ДЕРЖАВНИЙ УНИВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Кафедра системного аналізу

ЯРЦЕВ В.П.

Розподілені бази даних

Навчальній посібник

Oracle
PL/SQL

Київ – 2018

Рецензенти:

Циганок В.В., доктор технічних наук, с.н.с., завідувач лабораторії систем підтримки прийняття рішень Інституту проблем реєстрації інформації НАН України.

Гумен О.М, доктор технічних наук, професор, Національний технічній університет України «Київський політехнічний інститут ім. Ігоря Сікорського».

УДК 004.65
ББК 32.972.134

Ярцев В.П.
Розподілені бази даних: навчальний посібник. - К. ДУТ 2018. - 97с.

В навчальному посібнику розглядається систематизована сукупність відомостей що до понять розподілених баз даних, основні етапи та принципи побудови реляційних розподілених баз даних. Викладено математичні основи розробки об'єктів розподілених баз даних, оператори мови структурованих запитів PL/SQL, Розглянуті особливості розробки клієнтської та серверної частин інформаційної системи з реляційними базами та сховищами даних на сервері БД Oracle DB 11g. Приведено основи побудови додатків до баз даних у середовищі програмування Oracle Application.

Посібник призначений для студентів, що навчаються за усіма спеціальностями напрямку галузі - Інформаційні технології.

Обговорено та схвалено на засіданні кафедри системного аналізу
Протокол №__9___ від __12.04. 2018р.

Рекомендовано методичною радою факультету Інформаційних технологій ДУТ до друку
Протокол №_____ від _____2018р.

Зміст

1. Архітектура розподілених баз даних 6
   1.1 Основні поняття і визначення 6
   1.2 Різновиди архітектури РБД 7
   1.3 Гомогенні та гетерогенні розподілені СУБД 10
   1.4 Мультибазові системи 11
   1.5 Основні аспекти проектування розподілених БД 11
      1.5.1 Складові проектування розподілених БД 11
      1.5.2 Розподіл даних 12
      1.5.3 Фрагментація 13
      1.5.4 Реплікація 15
   1.6 Схеми володіння даними 16
   1.7 Збереження цілісності транзакцій 18
   1.8 Переваги та недоліки розподілених СУБД 19
   1.9 Забезпечення прозорості у розподілених системах керування БД 20
   1.10 Дванадцять правил Дейта для РСУБД 26
   Контрольні питання. 28
2. Структура, адміністрування та створювання баз даних у Oracle DB 11g 28
   2.1 Загальні відомості, склад, можливості СУБД 28
   2.2 Архітектура сервера Oracle 11g 29
   2.3 Фонові процеси Oracle 32
   2.4 Логічна та фізична структура бази даних 34
   2.5 Основні об'єкти БД 37
   2.6 Вбудовані типи даних 39
   2.7 Об'єктні типи даних в Oracle 40
   2.8 Приклад використання методів об'єктів 43
   2.9 Колекції 44
   2.10 Тип XMLTYPE 45
   2.11 Створення реляційної бази даних у сервері Oracle Database 11g 47
   2.11.1 Запуск сервера Oracle Database 11g 47
   2.11.2 Створення таблиць бази даних 50
   2.11. 3 Створення уявлень View для вибору даних з одної таблиці 54
   2.11.4 Створення уявлень View для вибору даних з декількох таблиць 56
3. Створення призначених для користувача додатків 58
   3.1 Реєстрація додатку, що строюється для перегляду даних таблиць 58
   3.2 Створення простого додатка для перегляду і редагуванню даних 59
   3.3 Запуск нової програми 65
   3.4 Розробка фільтра до таблиці 66
   3.5 Створення запиту 68
4. Створення WEB-додатку 71
   Рисунок 4.1 Вікно Websheet Applications 71
   Рисунок 4.3 Вікно Application builder 71
5. Утиліта SQL*Plus 76
   5.1 Настроювання середовища 76
   5.2 Програмне забезпечення SQL*Plus Instant Client 76
   5.3 Запуск сеансу SQL*Plus з командного рядка. 77
   5.4 Підключення до SQL*Plus через графічний інтерфейс Windows 79
   5.5 Команди SQL та SQL*Plus 80

5.6 Команди SQL*Plus і SQL 81
6. Призначення, основні оператори та синтаксис мови PL/SQL 81
6.1 Структура блока PL/SQL 82
6.2 Набір символів PL/SQL 83
6.3 Ідентифікатори 85
6.4 Структури програми PL/SQL 89
6.4.1 Структури керування обчисленнями 89
6.4.2 Умовні команди 89
6.4.3 Команда переходу 92
6.4.4 Цикли 93
6.5 Побудова запитів за допомогою утиліти SQLPlus 96

Список скорочень

ADO	- **(**Microsoft Active Data Objects) компоненти для доступу к БД
БД	- база даних
BDE	- (Borland Database Engine) процесор БД фірми Borland
ПЕОМ	- персональна електронна обчислювальна машина
СУБД	- система управління базами даних
ІС	- інформаційна система
ПБД	- персональна БД
КБД	- корпоративна БД
НФ	- нормальна форма
НД	- набір даних
ООП	- об’єктне орієнтоване програмування
ОС	- операційна система
ODBC	- (Open Data Base Connectivity) відчинений інтерфейс підключення до баз даних
CASE	- (Computer-Aided Software Engineering) система автоматизованого проектування
DLL	- бібліотека динамічно завантажуваемих процедур та функцій
DDL	- (Data definition language) - мова опису даних
SQL	- (Structured Queries Language) структурована мова запитів

1. Архітектура розподілених баз даних

1.1 Основні поняття і визначення

Технологія розподілених баз даних, що одержала в цей час широке поширення, сприяє зворотному переходу від централізованої обробки даних до децентралізованого. Створення технології систем керування розподіленими базами даних є одним із самих більших досягнень в області баз даних. Основною причиною розробки інформаційних систем, що використають бази даних, є прагнення інтегрувати всі оброблювані в компанії дані, у єдине ціле й забезпечити до них контрольований доступ.

Створення комп'ютерних мереж приводить до децентралізації обробки даних. Децентралізований підхід, по суті, відображає організаційну структуру підприємства, що логічно складає з окремих підрозділів, відділів, груп, які фізично розподілені по різних офісах, відділенням або філіям, причому кожна окрема одиниця має справу із власним набором оброблюваних даних. Розробка розподілених баз даних, що відображають організаційні структури підприємств, дозволяє зробити дані, підтримувані кожним з існуючих підрозділів, загальнодоступними, забезпечивши при цьому їхнє збереження саме в тих місцях, де вони найчастіше використаються. Подібний підхід розширює можливості спільного використання інформації, одночасно підвищуючи ефективність доступу до неї.

Розподілені системи вирішують проблему *островів інформації.*

Бази даних можна представити як якісь електронні острови, що представляють собою окремі, і в загальному випадку, важкодоступні місця, подібні вилученим друг від друга островам. Таке положення може бути наслідком географічної роз'єднаності, несумісності використовуваної архітектури комп'ютерів, несумісності використовуваних комутаційних протоколів. Інтеграція окремих баз даних в одне логічне ціле здатне змінити подібне положення справ.

Розподілена база даних — це набір логічно зв'язаних між собою поділюваних даних і їхніх описів, які фізично розподілені в деякій комп'ютерній мережі.

Розподілена система керування базою даних (РСУБД) — це програмна система, призначена для керування розподіленими базами даних і що дозволяє зробити розподіленість інформації прозорої для кінцевого користувача.

Розподілена система керування базами даних складається з єдиної логічної бази даних, розділеної на деяку кількість фрагментів. Кожний фрагмент бази даних зберігається на одному або декількох комп'ютерах (вузлах, sites), які з'єднані між собою комунікаційною мережею й кожний з яких працює під керуванням окремої СУБД. Будь-який користувач може виконати операції над даними на своєму локальному вузлу точно так само, як якби цей вузол зовсім не входив у розподілену систему (що створює певний ступінь локальної автономії). З іншого боку, будь-який вузол здатний обробляти дані, що зберігають на інших комп'ютерах мережі.

Користувачі взаємодіють із розподіленою базою даних через додатки. Локальні додатки не вимагають доступу до даних на інших вузлах, глобальні додатки вимагають подібного доступу. У розподіленої СУБД повинне існувати хоча б один глобальний додаток, тому будь-яка РСУБД повинна мати наступні особливості.

Дані, що зберігають, розбиті на деяку кількість фрагментів. Між фрагментами може бути організована реплікація даних. Фрагменти і їхні репліки розподілені по різних вузлах. Вузли зв'язані між собою мережними з'єднаннями. Робота з даними на кожному вузлу управляється СУБД.

СУБД на кожному вузлу здатні підтримувати автономну роботу локальних додатків.

Реплікація укладається в підтримці актуальної копії (репліки) деякого фрагмента бази даних на декількох різних вузлах. Немає необхідності в тім, щоб на кожному з вузлів системи існувала своя власна база даних, що показано на прикладі РСУБД, представленої на рис. 1.1.

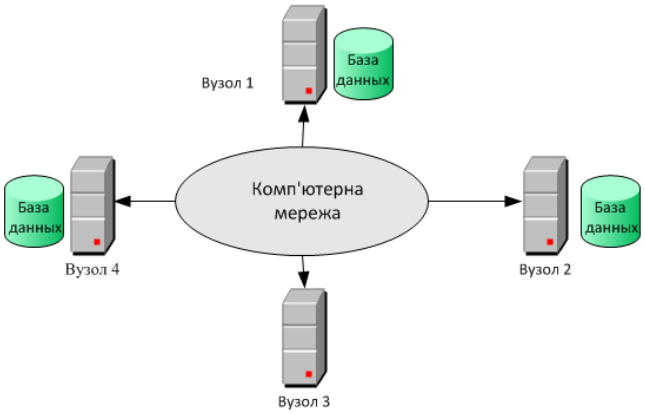


Рис.1.1 Варіант топології РСУБД

З визначення РСКБД треба, що для кінцевого користувача розподіленість системи повинна бути зовсім прозора (невидима), виглядати так само, як нерозподілена система. У деяких випадках цю вимогу називають фундаментальним принципом побудови розподілених СУБД .

1.2 Різновиди архітектури РБД

Розглянемо, які існують розходження між розподіленими СУБД і розподіленою обробкою даних.

Розподілена обробка — це обробка з використанням централізованої бази даних, доступ до якої може здійснюватися з різних комп'ютерів мережі.

Основним моментом у визначенні розподіленої бази даних є затвердження, що система працює з даними, *фізично розподіленими в мережі.*

Якщо дані зберігаються централізовано, то навіть у тому випадку, коли доступ до них забезпечується для будь-якого користувача в мережі, дана система просто підтримує розподілену обробку, але не може розглядатися як розподілена СУБД. Схематично подібна топологія представлена на рис.1.2. Ця топологія часто називається системою "клієнт/сервер".

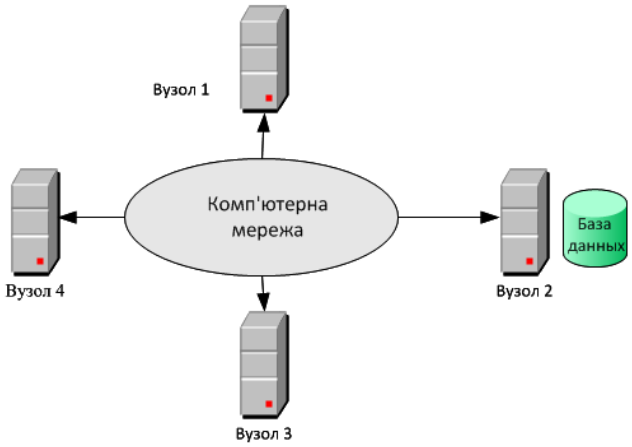


Рисунок 1.2 Система "клієнт/сервер" з розподіленою обробкою.

Система "**клієнт/сервер**" — це система, у якій одні вузли — ***клієнти***, а інших — ***сервери***. Всі дані розміщені на вузлах, які є серверами; всі додатки виконуються на вузлах-клієнтах і повній локальній незалежності не надається.

Клієнт — це процес, що посилає запит на обслуговування. Архітектура клієнт-сервер забезпечує прикладним програмам клієнта доступ до даних, якими управляє сервер. Це основне призначення цієї архітектури, тобто кілька клієнтів ефективно використають один сервер.

Можливі кілька варіантів основної схеми:

- кілька клієнтів можуть спільно використати один сервер.
- окремий клієнт може мати доступ до декількох серверів.

Така можливість, у свою чергу, ділиться на два випадки.

а) Клієнт обмежений доступом лише до одного сервера ***за один раз***, тобто кожний окремий запит до бази даних повинен бути орієнтованим на один сервер. Неможливо в межах одного запиту одержати дані із двох або більше різних серверів. Більше того, користувач повинен знати, на якому саме сервері зберігаються ті або інші частини даних.

б) Клієнт може мати одночасний доступ до декількох серверів, тобто окремий запит може сполучити дані з декількох серверів. А це означає, що кілька серверів надаються клієнтові так, начебто це насправді один сервер. Користувач не повинен знати, які частини даних зберігаються на кожному сервері. Але у випадку *b* фактично описаний принцип системи розподіленої бази даних. Це не зовсім те, що мають на увазі під терміном "клієнт/сервер".

Паралельна СУБД — це система керування базою даних, що функціонує з використанням декількох процесорів і пристроїв жорстких дисків, що дозволяє їй розпаралелювати виконання деяких операцій з метою підвищення загальної продуктивності обробки.

Застосування паралельних СУБД дозволяє об'єднати кілька малопотужних машин для одержання того ж самого рівня продуктивності, що й у випадку однієї, але могутнішої машини.

Для надання декільком процесорам спільного доступу до однієї й тій же базі даних паралельна СУБД повинна забезпечувати керування спільним доступом до ресурсів. Те, які саме ресурси розділяються і як цей поділ реалізований на практиці, безпосередньо впливає на показники продуктивності створюваної системи.

До основних типів архітектури паралельних СУБД (рис.1.3) ставляться:

- системи з поділами пам'яті;
- системи з поділами дисків;
- системи без поділів.

Схему без поділу в деяких випадках відносять до розподілених СУБД, у паралельних системах розміщення даних диктується винятково міркуваннями продуктивності. Вузли розподіленої СУБД звичайно розділені географічно, незалежно адмініструють та з'єднані між собою відносно повільними мережними з'єднаннями, тоді як вузли паралельної СУБД найчастіше розташовуються на тому самому комп'ютері або в межах того самого вузлу.

У **систему з поділом пам'яті** входить кілька процесорів, що розділяють загальну системну пам'ять (рис.1.3,а). Цю архітектуру називають також ***симетричної багатопроцесорної обробкою***. Вона застосовується для самих різних обчислювальних платформ, починаючи від персональних ЕОМ, що містять трохи паралельно працюючих мікропроцесорів, більших RISC-систем і аж до найбільших мейнфреймов. Ця архітектура забезпечує швидкий доступ до даних для обмеженого числа процесорів (не більше 64). При збільшенні числа процесорів мережні взаємодії починають обмежувати продуктивність всієї системи.

Системи без поділу інакше називають системами масової паралельної обробки (рис.1.3,в).У таких системах кожний процесор має свою власну оперативну й дискову пам'ять.

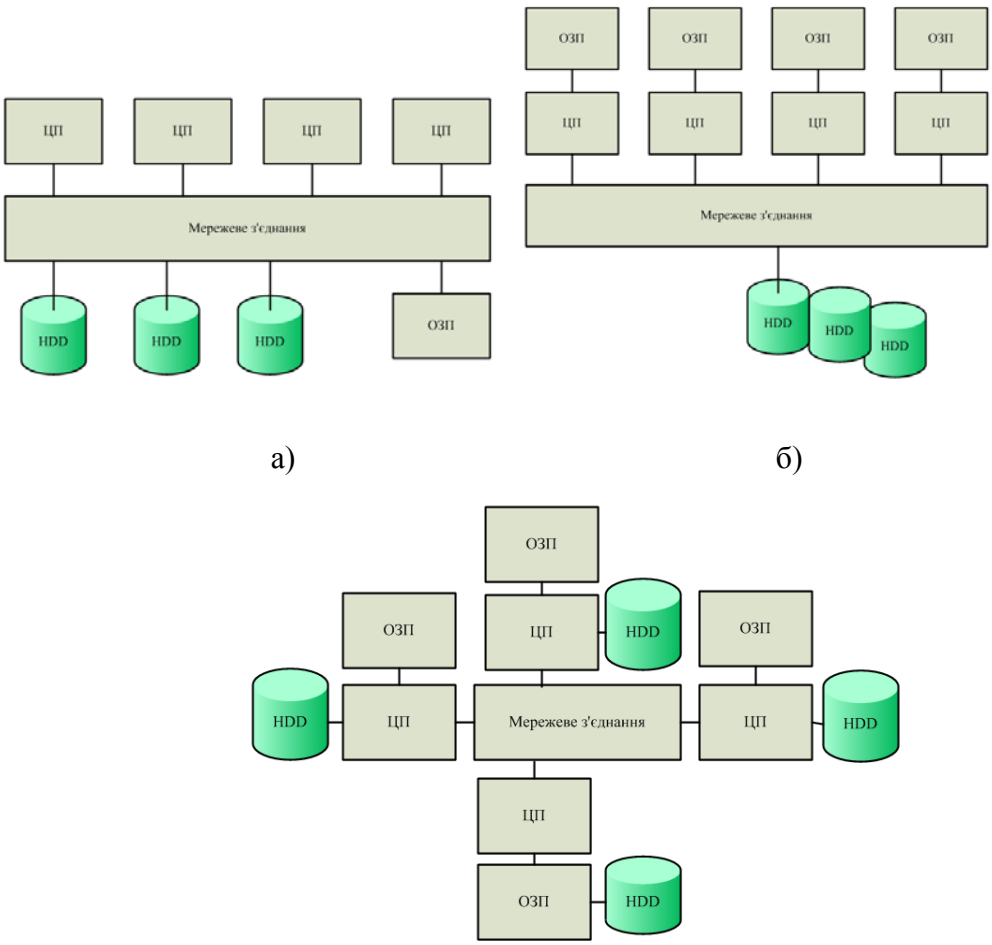


Рисунок 1.3 Архітектура з паралельною обробкою:
а) з розподілом пам’яті; б) з розподілом дисків; в) без розподілу

База даних розподілена між всіма дисковими пристроями обчислювальних підсистем, пов'язаних із цією базою даних. У результаті всі дані прозоро доступні користувачам кожної з обчислювальних підсистем. Така архітектура забезпечує більше високу масштабованість, чим системи з поділом пам'яті, і дозволяє легко організувати підтримку роботи великої кількості процесорів. Однак тільки у випадку, коли необхідні дані зберігаються локально, вдається досягти максимальної продуктивності.

У **системах з поділом дисків** кожний із процесорів має безпосередній доступ до всім спільно використовуваним дисковим пристроям, але має власну оперативну пам'ять (рис.1.3,б). Такі системи оптимальні для додатків з високою централізованою обробкою й забезпечують найвищі показники доступності й продуктивності. У системах з такою архітектурою, як і у випадку архітектури без поділу, виключаються вузькі місця, пов'язані з поділом пам'яті без додаткового навантаження по фізичному розподілі даних на різних пристроях. Поділювані дискові системи називають ***кластерами.***

Паралельні технології звичайно використаються в системах, які повинні підтримувати виконання тисяч транзакцій у секунду, або у випадку винятково більших баз даних (1-2 терабайту). Такі системи повинні забезпечувати малій час реакції на запит і мають потребу в доступі до великого обсягу даних.

1.3 Гомогенні та гетерогенні розподілені СУБД

Розподілені СУБД класифікуються як гомогенні й гетерогенні. Якщо всі вузли розподіленої системи використають той самий тип СУБД, то така система називається **гомогенної.** У **гетерогенних** системах на вузлах можуть функціонувати різні типи СУБД, що використають різні моделі даних (реляційні, мережні, ієрархічні і ін.).

Гомогенні системи простіше проектувати і супроводжувати. Такі РСУБД дозволяють поетапно нарощувати розміри системи, додаючи нові вузли до вже існуючої розподіленої системи. Більше того, організувати на різних вузлах паралельну обробку інформації, можна підвищити продуктивність всієї системи.

Гетерогенні системи виникають у випадках інтеграції в новостворювану розподілену систему незалежних вузлів зі своїми власними системами баз даних. У гетерогенних системах для організації взаємодії між різними типами СУБД буде потрібно організувати трансляцію переданих повідомлень. Користувачі кожного з вузлів повинні мати можливість уводити свої запити мовою тієї СУБД, що використається на цьому вузлу (прозорість у відношенні використовуваної СУБД). Система повинна забезпечити локалізацію необхідних даних і виконання трансляції переданих повідомлень. У загальному випадку дані можуть бути запитані з іншого вузла, що має:

- інший тип використовуваного встаткування;
- інший тип використовуваної СУБД;
- інший тип застосовувані встаткування й СУБД.

Якщо використається інший тип устаткування, але та ж СУБД, методи виконання трансляції укладаються в заміні кодів і зміні довжини слова. Якщо типи використовуваних на вузлах СУБД різні, трансляція ускладняється необхідністю відображення структури даних однієї моделі у відповідні структури даних іншої моделі. Наприклад, відносини в реляційної моделі даних повинні бути перетворені в записи і набори, типові для мережної моделі даних. Крім того, необхідно транслювати текст запитів з однієї використовуваної мови маніпулювання даними на іншій. Якщо відрізняються й тип устаткування, і тип використовуваної СУБД, буде потрібно виконувати обидва види трансляції.

Додаткові складності виникають при спробі вироблення єдиної концептуальної схеми, створюваної шляхом інтеграції незалежних локальних концептуальних схем.

Типове рішення, застосовуване в деяких реляційних системах для забезпечення прозорості у відношенні використовуваної СУБД, складається у використанні шлюзів. До складу окремих частин гетерогенних розподілених систем повинні входити шлюзи, призначені для перетворення мови й моделі даних кожного з використовуваних типів СУБД у мову й модель даних реляційної системи. Однак такий підхід не вільний від деяких серйозних недоліків. Наприклад, шлюзи не дозволяють організувати систему керування транзакціями навіть для окремих пар систем. Тобто шлюз між двома системами являє собою не більш ніж транслятор запитів. Тому шлюзи не дозволяють упоратися із проблемами, викликаними неоднорідністю структур і поданням даних у різних схемах.

Створена робоча група (Specification Working Group - SWG) повинна підготувати специфікації, що регламентують інфраструктуру середовища бази даних:

- уніфікований і потужний інтерфейс мови SQL (SQL API), що дозволяє створювати клієнтські додатки так, щоб вони не були прив'язані до конкретного типу використовуваної СУБД;
- уніфікований протокол доступу до бази даних, що дозволяє СУБД одного типу безпосередньо взаємодіяти із СУБД іншого типу, без необхідності використання якого-небудь шлюзу;
- уніфікований мережний протокол, дозволяє здійснювати взаємодія СУБД різного типу.

Найбільш важливим завданням цієї групи є пошук способу, що дозволяє в одній транзакції виконувати обробку даних, що втримуються в декількох базах, керованих СУБД різних типів, причому без необхідності використання яких-небудь шлюзів.

1.4 Мультибазові системи

Однієї з різновидів розподілених СУБД є **мультибазові системи**. Мультибазова система — розподілена система керування базами даних, у якій керування кожним з вузлів здійснюється зовсім автономно.

У мультибазових системах виконується інтеграція таких розподілених систем баз даних, у яких весь контроль над окремими локальними системами цілком і повністю здійснюється їхніми операторами. Повна автономія вузлів дозволяє не вносити які-небудь зміни в локальні СУБД. Отже, мультибазові СУБД вимагають створення поверх існуючих локальних систем додаткового рівня програмного забезпечення, призначеного для надання необхідної функціональності.

Мультибазові системи дозволяють кінцевим користувачам різних вузлів одержувати доступ і спільно використати дані без необхідності фізичної інтеграції існуючих баз даних. Вони забезпечують користувачам можливість управляти базами даних їхніх власних вузлів без якого-небудь централізованого контролю, що обов'язково є присутнім у звичайних типах РСУБД. Адміністратор локальної бази даних може дозволити доступ до певної частини своєї бази даних за допомогою створення *схеми експорту,* що визначає, до яких елементів локальної бази даних зможуть одержувати доступ зовнішні користувачі.

Мультибазова СУБД прозорим образом розташовується поверх існуючих баз даних і файлових систем, надаючи їх своїм користувачам як деяку єдину базу даних. Така підтримка глобальної схеми дозволяє користувачам на підставі цієї схеми будувати запити й модифікувати дані. Мультибазова СУБД *працює тільки із глобальною схемою*, тоді як локальні СУБД власними засобами забезпечують підтримку даних всіх їхніх користувачів.

Глобальна схема створюється за допомогою інтеграції схем локальних баз даних. Програмне забезпечення мультибазової СУБД попередньо транслює глобальні запити й перетворює їх у запити й оператори модифікації даних відповідних локальних СУБД. Отримані після виконання локальних запитів результати зливаються в єдиний глобальний результат, надаваний користувачеві. Крім того, мультибазова СУБД здійснює контроль за виконанням фіксації або відкоту окремих операцій глобальних транзакцій локальних СУБД, а також забезпечує збереження цілісності даних у кожній з локальних баз даних. Програми мультибазової СУБД управляють різними шлюзами, за допомогою яких контролюють роботу локальних СУБД.

1.5 Основні аспекти проектування розподілених БД

1.5.1 Складові проектування розподілених БД

Фрагментація. Будь-яке відношення може бути розділено на частини або *фрагменти* при організації фізичного зберігання цього відношення. Фрагменти розподіляються по різних вузлах. Якщо у фрагмент виділяється підмножина кортежів відношення, то він називається горизонтальним фрагментом. Фрагмент називається вертикальним, якщо в ньому використовується підмножина атрибутів відношення. Визначення і розміщення фрагментів повинні проводитися на основі аналізу найбільш важливих застосувань, що визначають особливості використання бази даних.

Реплікація. Одним із завдань РСУБД є підтримка актуальної копії деякого фрагмента на декількох вузлах.

Розподіл. Вузол для зберігання фрагмента вибирається виходячи з деякої оптимальної схеми розміщення фрагментів. При проектуванні повинні враховуватися як якісні, так і

кількісні показники майбутньої системи. Розподіл виконується на основі кількісної інформації, а базою при створенні схеми фрагментації є якісні показники. До кількісних показників відносяться наступні:

- частота запуску застосування на виконання;
- вузол, на якому запускається застосування;
- вимоги до продуктивності транзакцій і застосувань.

Якісна інформація може включати перелік транзакцій, що виконуються в застосуванні, використовувані в цих транзакціях відносини, атрибути і кортежі, тип доступу до цих об'єктів (читання або запис), предикати, використовувані в операціях читання.

Розділення відносин на фрагменти і розподіл фрагментів по вузлах виконується для досягнення наступної мети :

• *Локальність посилань.* Дані повинні зберігатися якомога ближче до місць їх використання. Якщо фрагмент використовується декількома вузлами, може виявитися доцільним розмістити на цих вузлах його репліки.

• *Підвищення надійності і доступності.* Надійність і доступність даних підвищуються за рахунок використання механізму реплікації. У разі відмови одного з вузлів завжди існуватиме копія фрагмента, що зберігається на іншому вузлу.

• *Достатній рівень продуктивності.* Неправильний вибір схеми розміщення даних може привести до виникнення в системі вузьких місць. Деякий вузол може бути переобтяжений запитами з боку інших вузлів, що приведе до зниження продуктивності всієї системи. З іншого боку, деякі вузли "простоюватимуть", тобто ресурси системи при неправильному розподілі використовуватимуться неефективно.

• *Прийнятне співвідношення між вартістю і місткістю зовнішньої пам'яті.* На всіх вузлах звичайно рекомендується використовувати дешевші пристрої масової пам'яті. Ця вимога повинна

бути узгоджена з вимогою підтримки *локальності посилань.*

• *Мінімізація витрат на передачу даних.* Вартість виконання в системі видалених запитів — одна з найважливіших характеристик системи. Витрати на вибірку будуть мінімальні при забезпеченні максимальної локальності посилань, тобто коли кожен вузол матиме свою власну копію даних. Проте при оновленні даних, що репліціруються, внесені зміни доведеться поширювати на всі вузли з репліками, що викличне збільшення витрат на передачу даних.

1.5.2 Розподіл даних

Існує чотири схеми розміщення даних в системі:

- централізоване
- роздільне (фрагментовано)
- розміщення з повною реплікацією
- розміщення з вибірковою реплікацією.

Розглянемо ці схеми докладніше з погляду досягнення мети, визначеної вище.

Централізоване розміщення. На одному з вузлів під управлінням СУБД створюється і зберігається єдина база даних. Доступ до цієї бази мають всі користувачі мережі. В цьому випадку локальність посилань мінімальна для всіх вузлів, окрім центрального, оскільки для отримання будь-якого доступу до даних потрібна установка мережевого з'єднання. Відповідно рівень витрат на передачу даних буде високий. Рівень надійності і доступності в системі низок, оскільки аварійна ситуація на центральному вузлу приведе до відмови всієї системи.

Локальність посилань - найнижча.
Надійність і доступність - найнижча.
Продуктивність - незадовільна.
Вартість пристроїв зберігання - найнижча.
Витрати на передачу даних – найвищі.

Роздільне (фрагментовано) розміщення. База даних розбивається на непересічні фрагменти, кожний з яких розміщується на одному з вузлів системи. Рівень локальних посилань буде високий, якщо на вузлах розміщені саме ті елементи даних, які найчастіше використовуються на цих вузлах. Якщо реплікація не використовується, то вартість зберігання даних буде мінімальна, але при цьому буде також невисокий рівень надійності і доступності даних в системі. Проте він буде вищий, ніж в попередньому варіанті, оскільки аварійна ситуація на будь-якому з вузлів викличне відмову в доступі тільки до тієї частини даних, яка на ньому зберігалася.

При правильно вибраній схемі розподілу даних рівень продуктивності в системі буде відносний високим, а рівень витрат на передачу даних — низьким.

Локальність посилань - висока.
Надійність і доступність - низька для окремих елементів.
Продуктивність - задовільна.
Вартість пристроїв зберігання - найнижча.
Витрати на передачу даних - низькі.

Розміщення з повною реплікацією. Повна копія всієї бази даних розміщується на кожному з вузлів системи. Отже, локальність посилань, надійність і доступність даних і рівень продуктивності системи будуть максимальні. Проте вартість пристроїв зберігання даних і рівень витрат на передачу даних в цьому випадку будуть також найвищими.

Локальність посилань - найвища.
Надійність і доступність - найвища.
Продуктивність - хороша для операцій читання.
Вартість пристроїв зберігання - найвища.
Витрати на передачу даних - високі для операцій оновлення, низькі для операцій читання.

Розміщення з вибірковою реплікацією. Ця схема є комбінацією методів фрагментації, реплікації і централізації. Одні масиви даних розділяються на фрагменти, що дозволяє добитися для них високої локальності посилань. Інші дані, використовувані на багатьох вузлах, але не часто оновлюються, репліцируються. Решта даних зберігається централізований. Така стратегія дозволяє об'єднати всі переваги, що існують в решті моделей, і виключити властиві їм недоліки. Завдяки своїй гнучкості саме ця стратегія використовується найчастіше.

Локальність посилань - висока.
Надійність і доступність - низька для окремих елементів, висока для системи в цілому.
Продуктивність -задовільна.
Вартість пристроїв зберігання - середня.
Витрати на передачу даних - низькі.

1.5.3 Фрагментація

Система підтримує **фрагментацію,** якщо дане відношення може бути розділено на частини або *фрагменти* при організації його фізичного зберігання. Фрагментація бажана для

підвищення ефективності системи. В цьому випадку дані можуть зберігатися в тому місці, де вони найчастіше використовуються. Це дозволяє досягти локалізації більшості операцій і зменшення мережевого трафіку. Крім того, виключається зберігання даних, які не використовуються локальними застосуваннями, а значить, неавторизовані користувачі не зможуть одержати до ним доступ.

З іншого боку, продуктивність застосувань, що вимагають доступу до даних з декількох фрагментів на різних вузлах, може виявитися недостатньою. Підтримка цілісності даних також може істотно ускладнювати, оскільки функціонально залежні дані можуть виявитися фрагментованими і розміщуватися на різних вузлах.

Для коректності фрагментації необхідне виконання наступних правил.

1. *Повнота.* Кожен елемент даних з початкового відношення повинен бути присутнім, принаймні, в одному із створених фрагментів. Це гарантує відсутність втрати інформації при фрагментації.

2. *Відновлюваність.* Початкове відношення повинно бути відновлено з його фрагментів за допомогою операцій реляційної алгебри. Це гарантує збереження функціональних залежностів.

3. *Непересікальність.* Один елемент даних не повинен бути присутнім в двох і більш фрагментах. Виняток становить вертикальна фрагментація, коли в кожному фрагменті повинні бути присутніми атрибути первинного ключа, необхідні для відновлення початкового відношення. Це правило гарантує мінімальну надмірність даних у фрагментах.

Існують два основних типу фрагментації: **горизонтальна** і **вертикальна.** Горизонтальні фрагменти є підмножинами кортежів відношення, а вертикальні — підмножини атрибутів відношення.

Горизонтальна фрагментація. Горизонтальним називається фрагмент, виділений з відношення по горизонталі і що складається з деякої підмножини кортежів цього відношення.

Для створення горизонтального фрагмента визначається предикат, за допомогою якого виконується відбір кортежів з початкового відношення.

Цей тип фрагмента визначається за допомогою операції *вибірки (селекції)* реляційної алгебри. Операція вибірки дозволяє відібрати групу кортежів, що мають деяку загальну для них властивість, — наприклад, всі кортежі, використовувані одним із застосувань, або всі кортежі, вживані на одному з вузлів. Якщо задано відношення Rb те його горизонтальний фрагмент може бути визначений формулою:

$$R=\delta_F=(R_1)$$

де F є предикатом, побудованим з використанням одного або більше атрибутів відношення R_b.

Побудова схеми фрагментації припускає пошук набору **мінімальних** (тобто повних і релевантних) предикатів для розбиття відношення на фрагменти. Набір предикатів є повним тоді і тільки тоді, коли вірогідність звернення до будь-яких двох кортежів одного і того ж фрагмента з боку будь-якого застосування існує, принаймні, одне застосування, яке по-різному звертається до виділених за допомогою цього предиката фрагментів.

Вертикальна фрагментація. Вертикальним називається фрагмент, виділений з відношення по вертикалі і що складається з підмножини атрибутів цього відношення.

При вертикальній фрагментації в різні фрагменти об'єднуються атрибути, використовувані окремими застосуваннями. Визначення фрагментів в цьому випадку виконується за допомогою операції проекції реляційної алгебри. Для заданого відношення R вертикальний фрагмент може бути обчислений за допомогою наступної формули:

$$R=\pi_{a1,\ldots,an}(R_1)$$

де a_1..., a_n є атрибутами відношення R_b.

Змішана фрагментація. Іноді для адекватного розподілу даних між застосуваннями тільки горизонтальної або тільки вертикальній фрагментації виявляється недостатньо. У таких випадках використовують **змішану** фрагментацію.

Змішаний фрагмент утворюється або після додаткової вертикальної фрагментації створених раніше горизонтальних фрагментів, або шляхом горизонтальної фрагментації визначених раніше вертикальних фрагментів.

Змішана фрагментація визначається поєднанням операцій вибірки і проекції реляційної алгебри. Для існуючого відношення R змішаний фрагмент можна визначити по формулах:

$$R=\delta_{F(}\pi_{al,\ldots,an}(R_1))$$

$$R=\pi_{al,\ldots,an}(\delta_F (R_1))$$

де F - предикат з використанням одного або більш атрибутів a_l..., a_n відносини Ri.

Якщо відношення містить невелику кількість кортежів, які відносно рідко оновлюються, то від фрагментації **відмовляються.** Таке відношення залишають не фрагментованим і просто розміщують на кожному з вузлів його репліцируємі копії.

Пошук відносин, які не потребують фрагментації — перший етап процедури визначення схеми фрагментації. Потім аналізують відносини, розташовані на одиничній стороні зв'язків типу "один до багатьох", і підбирають для них оптимальні схеми фрагментації. На останньому етапі аналізуються відносини, розташовані на множинній стороні тих же зв'язків.

1.5.4 Реплікація

Реплікацію можна визначити як процес генерації і відтворення декількох копій даних, що розміщуються на одному або декількох вузлах. Механізм реплікації дуже важливий, оскільки дозволяє організації забезпечувати доступ користувачам до актуальних і тоді, коли вони цього потребують. Використання реплікації дозволяє досягти багатьох переваг, включаючи підвищення продуктивності (у тих випадках, коли централізований ресурс виявляється переобтяженим), надійності зберігання і доступності даних, наявність "гарячої" резервної копії на випадок відновлення, а також можливість підтримки мобільних користувачів і сховищ даних.

Види реплікації.

Протоколи оновлення репліцируемих даних побудовані на допущенні, що оновлення всіх копій даних виконуються як частина самої транзакції оновлення. Іншими словами, всі копії репліцируемих даних оновлюються одночасно із зміною початкової копії і, як правило, за допомогою протоколу двофазної фіксації транзакцій [2]. Такий варіант реплікації називається *синхронною реплікацією.*

Хоча цей механізм може бути просто необхідний для деякого класу систем, в яких всі копії даних потрібно підтримувати в абсолютно синхронному стані (наприклад, у разі фінансових операцій), йому властиві певні недоліки. Зокрема, транзакція не зможе бути завершена, якщо один з вузлів з копією репліцируемих даних виявиться недоступним. Крім того, безліч повідомлень, необхідних для координації процесу синхронізації даних, створюють істотне додаткове навантаження на корпоративну мережу.

Багато комерційних розподілених СУБД надають інший механізм реплікації, що одержав назву *асинхронної реплікації.* Він передбачає оновлення цільових баз даних після виконання оновлення початкової бази даних. Затримка у відновленні узгодженості даних може варіюватися від декількох секунд до декількох годин або навіть днів. Проте рано чи пізно дані у всіх копіях будуть приведені в синхронний стан. Хоча такий підхід порушує принцип незалежності розподілених даних, він цілком може розумітися як прийнятний компроміс між цілісністю даних і їх доступністю, причому останнє може бути важливіше для організацій, чия

діяльність допускає роботу з копією даних, необов'язково точно синхронізованої на даний момент.

Функції служби реплікації. Як базовий рівень служба реплікації розподілених даних повинна копіювати дані з однієї бази даних в іншу синхронно або асинхронно. Крім того, існує безліч інших функцій, які повинні підтримуватися, включаючи наступні [2].

1. *Масштабованість.* Служба реплікації повинна ефективно обробляти як малі, так і великі об'єми даних.
2. *Відображення і трансформація.* Служба реплікації повинна підтримувати реплікацію даних в гетерогенних системах, що використовують декілька платформ. Це може бути пов'язано з необхідністю відображення і перетворення даних з однієї моделі даних в іншу чи ж для перетворення деякого типу даних у відповідний тип даних, але для середовища іншої СУБД.
3. *Реплікація об'єктів.* Повинна існувати можливість репліцировать об'єкти, відмінні від звичайних даних. Наприклад, в деяких системах допускається реплікація індексів і процедур, що зберігаються (або тригерів).
4. *Засоби визначення схеми реплікації.* Система повинна надавати механізм, що дозволяє привілейованим користувачам задавати дані і об'єкти, що підлягають реплікації.
5. *Механізм підписки.* Система повинна включати механізм, що дозволяє привілейованим користувачам оформляти підписку на дані і інші належні реплікації об'єкти.
6. *Механізм ініціалізації.* Система повинна включати засоби, що забезпечують ініціалізацію новостворюваної репліки.

1.6 Схеми володіння даними

Володіння даними визначає, якому з вузлів буде наданий привілей оновлення даних. Основними типами схем володіння є :

- " провідний / ведений ";
- "робочий потік";
- "повсюдне оновлення".

Останній варіант іноді називають *одноранговою,* або *симетричною реплікацією.*

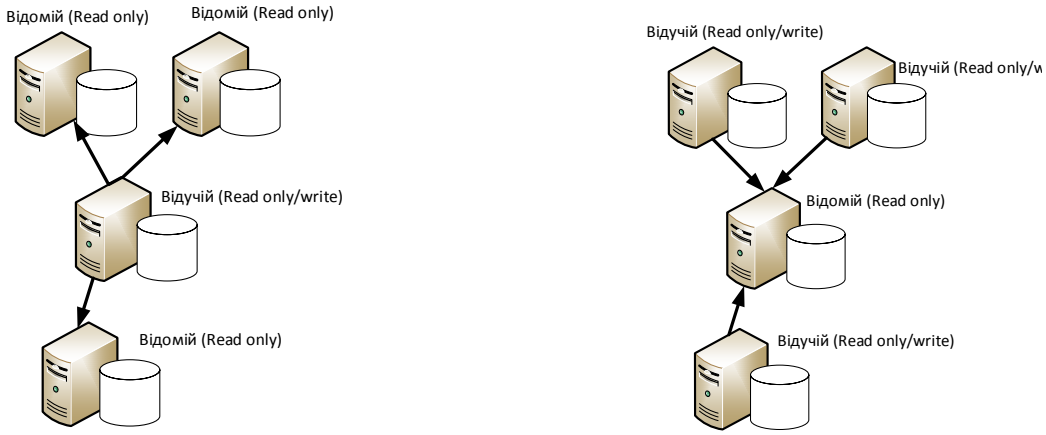
При організації володіння даними по схемі " провідний / ведений " асинхронно реплицируемі дані належать одному з вузлів, званому ведучим, або первинним, і можуть оновлюватися тільки на ньому. Тут можна провести аналогію між видавцем і підпісчікамі. Видавець (що веде вузол) публікує свої дані. Вся решта вузлів тільки підписується на дані, що належать провідному сайту, тобто мають власні локальні копії, доступні їм тільки для читання. Потенційно кожний з сайтів може грати роль ведучого для різних наборів даних, що не перекриваються. Проте в системі може існувати тільки один вузол, на якому розташовується провідна оновлювана копія кожного конкретного набору даних, а це означає, що конфлікти оновлення даних в системі повністю виключені. Нижче наводиться декілька прикладів можливих варіантів використання цієї схеми реплікації.

• *Системи, підтримка прийняття рішень* (ППР). Дані з однієї або більш розподілених баз даних можуть вивантажуватися в окрему, локальну систему ППР, де вони тільки прочитуватимуться при виконанні різних видів аналізу.

• *Централізований розподіл або розповсюдження інформації.* Розповсюдження даних має місце в тих випадках, коли дані оновлюються тільки в центральній ланці системи, після чого репліцируются їх копії, доступні тільки для читання. Цей варіант реплікації даних показаний на рис. 1.4,а.

• *Консолідація видаленої інформації.* Консолідація даних має місце в тих випадках, коли оновлення даних виконується локально, поле чого їх копії, доступні тільки для читання, відсилаються в загальне сховище. У цій схемі кожний з сайтів автономно володіє деякою частиною даних (рис. 1.4,б).

• *Підтримка мобільних користувачів.* Підтримка роботи мобільних користувачів набула останніми роками дуже широкого поширення. Співробітники багатьох організацій вимушені постійно переміщатися з місця на місце і працювати за межами офісів. Розроблено декілька методів надання необхідних даних мобільним користувачам. Одним з них і є реплікація. В цьому випадку на вимогу користувача дані завантажуються з локального сервера його робочої групи. Оновлення, виконані клієнтом для даних робочої групи або центрального сайту, обробляються схожим чином.



а) б)

• Рисунок 1.4 Схеми володіння даними

Провідний сайт може володіти даними всієї таблиці, і в цьому випадку вся решта сайтів є лише підпісчіками на копії цієї таблиці, доступні тільки для читання. У альтернативному варіанті багато сайтів володіють окремими фрагментами таблиці, а решта сайтів може виступати як підпісчіки копій кожного з цих фрагментів, доступних їм тільки для читання. Цей тип реплікації називають *асиметричною реплікацією.*

Як і у разі схеми " провідний / ведений ", в моделі "*робочий потік"* вдається уникнути появи конфліктів оновлення, хоча даній моделі властивий більший динамізм. Схема володіння "робочий потік" дозволяє передавати право оновлення реплицируемих даних від одного сайту іншому. Проте в кожен конкретний момент часу існує тільки один сайт, що має право оновлювати деякий конкретний набір даних. Типовим прикладом використання схеми робочого потоку є система обробки замовлень, в якій робота з кожним замовленням виконується у декілька етапів, наприклад оформлення замовлення, контроль кредитоспроможності, виписка рахунку, доставка.

Централізовані системи дозволяють застосуванням, що виконують окремі етапи обробки, діставати доступ і оновлювати дані в одній інтегрованій базі даних. Кожне застосування оновлює дані про замовлення по черзі тоді і тільки тоді, коли стан замовлення указує, що попередній етап обробки вже завершений. У моделі володіння "робочий потік" застосування можуть бути розподілені по різних сайтах, і коли дані реплицируются і пересилаються на наступний сайт в ланцюжку, разом з ними передається і право на їх оновлення.

У двох попередніх моделей є одна загальна властивість: у будь-який заданий момент часу тільки один вузол має право оновлювати дані. Всій решті сайтів доступ до реплік даних буде дозволений тільки для читання. В деяких випадках це обмеження виявляється дуже жорстким.

Схема володіння *з повсюдним оновленням* створює рівноправне середовище, в якому безліч вузлів мають однакові права на оновлення реплицируемих даних. В результаті локальні вузли дістають можливість працювати автономно, навіть в тих випадках, коли інші сайти недоступні.

Розділення права володіння може викликати виникнення в системі конфліктів, тому служба реплікації в цій схемі повинна використовувати той або інший метод виявлення і вирішення конфліктів.

1.7 Збереження цілісності транзакцій

Перші спроби реалізації механізму реплікації по самій своєї суті не передбачали збереження цілісності транзакцій. Дані копіювалися без збереження властивості атомарності транзакцій, що потенційно могло привести до втрати цілісності розподілених даних. Нагадаємо, що атомарність транзакції означає, що виконуються всі операції, що входять в неї, або нічого.

В даному випадку транзакція, що перебуває на початковому сайті з декількох операцій оновлення даних в різних таблицях, в процесі реплікації перетворювалася в серію окремих транзакцій, кожна з яких оновлювала дані в одній з таблиць. Якщо частина цих транзакцій на цільовому сайті завершувалася успішно, а решта частини — немає, то узгодженість інформації між двома базами даних втрачалася.

У протилежність цьому, реплікація із збереженням структури транзакції в початковій базі даних не порушує властивість атомарності транзакції.

Моментальні знімки таблиць. Метод *моментальних знімків таблиць* дозволяє асинхронно поширювати результати змін, виконаних в окремих таблицях, групах таблиць, уявленнях або фрагментах таблиць відповідно до наперед встановленого графіка.

Загальний підхід до створення моментальних знімків полягає у використанні файлу журналу відновлення бази даних, який дозволяє мінімізувати рівень додаткового навантаження на систему. Основна ідея полягає в тому, що файл журналу є кращим джерелом для отримання відомостей про зміни в початкових даних. Досить мати механізм, який звертатиметься до файлу журналу для виявлення змін в початкових даних, після чого поширювати виявлені зміни на цільові бази даних, не роблячи ніякого впливу на нормальне функціонування початкової системи. Різні СУБД реалізують подібний механізм по-різному: в деяких випадках даний процес є частиною самого сервера СУБД, тоді як в інших випадках він реалізується як незалежний зовнішній сервер.

Для відправки відомостей про зміни на інші сайти доцільно застосовувати метод організації черг. Якщо відбудеться відмова мережевого з'єднання або цільового сайту, відомості про зміни можуть зберігатися в чергах до тих пір, поки з'єднання не буде відновлено. Для гарантії збереження узгодженості даних порядок доставки відомостей на цільові сайти повинен зберігати початкову черговість внесення змін.

Тригері бази даних. Альтернативний підхід полягає в наданні користувачам можливості створювати власні застосування, що виконують реплікацію даних з використанням механізму *тригерів бази даних.*

В цьому випадку на користувачів покладається відповідальність за написання тих тригерних процедур, які викликатимуться при виникненні відповідних подій, наприклад при створенні нових записів або оновленні тих, що вже існують. Хоча подібний підхід надає більшу гнучкість, ніж механізм створення моментального знімка, йому також властиві певні недоліки.

• Відстежування запуску і виконання тригерних процедур створює додаткове навантаження на систему.

• тригері виконуються при кожній зміні рядка в провідній таблиці. Якщо провідна таблиця схильна до частих оновлень, виклик тригерних процедур може створити істотне

додаткове навантаження на застосування і мережеві з'єднання.

• У протилежність цьому, при використанні моментальних знімків всі виконані зміни пересилаються за одну операцію.

• Тригері не можуть виконуватися відповідно до деякого графіка. Вони виконуються в той момент, коли відбувається оновлення даних в провідній таблиці. Моментальні знімки можуть створюватися відповідно до встановленого графіка або навіть уручну. У будь-якому випадку це дозволяє виключити додаткове навантаження від реплікації даних в періоди пікової на вантаження на систему.

• Якщо репліцируется декілька зв'язаних таблиць, синхронізація їх реплікації може бути досягнута за рахунок використання механізму групових оновлень. Вирішити це завдання за допомогою тригерів істотно складніше.

• Анулювання результатів виконання тригерної процедури у разі відміни або відкоту транзакції — достатньо складне завдання.

Виявлення і вирішення конфліктів. Коли декілька вузлів можуть незалежно вносити зміни в репліцируемі дані, необхідно використовувати деякий механізм, що дозволяє виявляти конфліктуючі оновлення даних і відновлювати узгодженість інформації в базі. Простий механізм виявлення конфліктів в окремій таблиці полягає в розсилці початковим сайтом як нових, так і початкових значень змінених даних для кожного рядка, який був оновлений з моменту останньої синхронізації копій. На цільовому сайті сервер реплікації повинен порівняти з набутого значення кожен рядок в цільовій базі даних, яка була локально змінена за даний період. Проте цей метод вимагає установки додаткових угод для виявлення інших типів конфліктів, наприклад порушення посилальної цілісності між двома таблицями. Було запропоновано декілька різних механізмів вирішення конфліктів, проте найчастіше застосовуються наступні.

• *Найраніше або найпізніша тимчасова відмітка.* Змінюються відповідно дані з найважнішою або найпізнішою тимчасовою відміткою.

• *Пріоритети вузлів.* Застосовується оновлення, що поступило з вузла з найбільшим пріоритетом.

• *Доповнюючи і усереднені оновлення.* Введені зміни узагальнюються, може використовуватися в тих випадках, коли оновлення атрибуту виконується операціями, записаними у формі відхилень.

• *Мінімальне або максимальне значення.* Застосовуються оновлення, відповідні стовпцю з мінімальним або максимальним значенням.

• *За рішенням користувача.* Адміністратор бази даних створює власну процедуру вирішення конфлікту. Для усунення різних типів конфліктів можуть бути підготовлені різні процедури.

• *Збереження інформації для ухвалення рішення уручну.* Відомості про конфлікт записуються в журнал помилок для подальшого аналізу і усунення адміністратором бази даних уручну.

1.8 Переваги та недоліки розподілених СУБД

Основною причиною використання розподілених баз даних є те, що звичайно підприємства вже розподілені на підрозділи. Таким чином, інформація підприємства розбивається на частині (*острова) інформації)*.

Розподілена база даних забезпечує *мости* для їхнього з'єднання в ціле. У подібній базі даних персонал відділення компанії зможе виконувати необхідні йому локальні запити. Керівництву компанії може знадобитися виконувати глобальні запити, що передбачають одержання доступу до даних, що зберігається у всіх відділеннях компанії.

Розподілена система дозволяє структурі бази даних **відображати структуру організації.** Це є найбільш важливою перевагою розподілених СУБД.

У розподілених системах дані розміщаються на тім вузлу, на якому зареєстровані користувачі найчастіше їх використають. У результаті користувачі цього вузла одержують локальний контроль над необхідними їм даними й можуть регулювати локальні обмеження на їхнє використання. У цьому укладається **роздільність і локальна автономність** розподілених СУБД.

Розподілені СУБД проектуються так, щоб забезпечити працездатність системи, незважаючи на відмову одного з вузлів РСУБД або лінії зв'язку між вузлами. Це досягається організацією реплікації даних, так що дані і їхні копії будуть розміщені на декількох сайтах. Система буде перенаправляти запити до вузла, що відмовив, на адресу іншого сайту. Це приводить до **підвищення надійності системи й доступності даних.**

Виявляється, що набагато вигідніше встановлювати в підрозділах організації власні малопотужні комп'ютери, крім того, набагато дешевше додати в мережу нові робочі станції, чим модернізувати систему з мейнфреймом. Із цього випливають **економічні переваги** використання РСУБД.

Завдяки **модульності** розподіленого середовища **розширення** існуючої системи здійснюється набагато простіше. Додавання в мережу нового вузла не робить впливу на функціонування вже існуючих. Подібна гнучкість дозволяє організації легко розширюватися.

Недоліки. Найбільш істотним, є **підвищення складності програмного й апаратного забезпечення.** Розподілені СУБД є більше складними програмними комплексами, чим централізовані СУБД. Досить указати на той факт, що дані можуть піддаватися реплікації. Якщо реплікація даних не буде підтримуватися на необхідному рівні, система буде мати більше низький рівень доступності даних, надійності й продуктивності, чим централізовані системи.

У розподілених системах можуть виникнути **проблеми захисту** не тільки даних, реплицируемих на кілька різних сайтів, але й захисту мережних з'єднань самих по собі. У централізованих системах доступ до даних легко контролюється.

Ускладнення контролю за цілісністю даних. Вимоги забезпечення цілісності (коректності й погодженості даних) формулюються у вигляді обмежень. Виконання обмежень гарантує захист інформації в базі даних від руйнування. Реалізація таких обмежень цілісності вимагає доступу до великої кількості даних, використовуваних під час перевірок. У розподілених СУБД підвищена вартість передачі й обробки даних може перешкоджати організації ефективного захисту від порушень цілісності даних.

Ускладненні процедури проектування бази даних. Крім звичайних проблем, пов'язаних із проектуванням централізованих баз даних, розробка розподілених СУБД вимагає ухвалення рішення про фрагментації даних, розподілу фрагментів по окремих сайтах і організації процедур реплікації даних.

1.9 Забезпечення прозорості у розподілених системах керування БД

Прозорість розподілення. У визначенні РСУБД затверджується, що система повинна забезпечити прозорість розподіленого зберігання даних для кінцевого користувача.

Під прозорістю розуміється приховання від користувачів деталей реалізації системи. Інакше кажучи, користувачі розподіленої системи повинні мати можливість діяти так, ніби система *не* була розподілена. Всі проблеми розподілених систем повинні ставитися до внутрішніх проблем (проблемам реалізації), а не до зовнішніх проблем (проблемам користувальницького рівня). У централізованої СУБД забезпечення незалежності програм від даних також можна розглядати як одну з форм прозорості - від користувача ховаються зміни, що відбуваються у визначенні й організації зберігання даних.

Розподіленні СУБД можуть забезпечувати прозорість на різних рівнях.

При цьому переслідується одна мета: зробити роботу з розподіленою базою даних зовсім аналогічній роботі зі звичайної централізованої СУБД.

Виділяють чотири основних типи прозорості для системи з розподіленою базою даних.

- Прозорість розподіленості.
- Прозорість транзакцій.
- Прозорість виконання.
- Прозорість використання СУБД.

Прозорість розподіленості бази даних дозволяє кінцевим користувачам працювати з базою даних точно так, принаймні, з логічної точки зору, як якби дані в дійсності були зовсім не фрагментовано. Якщо РСУБД забезпечує прозорість розподіленості, то користувачеві не потрібно яких-небудь знань про фрагментації даних **(прозорість фрагментації)** або їхньому розміщенні **(прозорість розташування).** Прозорість фрагментації (як і прозорість розташування) - це досить важлива властивість, тому що вона дозволяє спростити розробку користувальницьких програм і виконання термінальних операцій.

Наприклад, це гарантує, що в будь-який момент дані можуть бути заново відновлені (а фрагменти перерозподілені) у відповідь на зміну вимог до ефективності роботи системи, причому ні користувальницькі програми, ні термінальні операції при цьому не зачіпаються.

С іншої сторони, якщо користувачеві необхідно мати відомості про фрагментації даних і розташуванні фрагментів, те цей тип прозорості називають **прозорістю локального відображення.** Далі ми розглянемо всі згадані типи прозорості.

Для ілюстрації обговорюваних концепцій буде використатися відношення СПІВРОБІТНИК, фрагментовано так, як описано в прикладі, а саме:

$C_1=\pi_{Таб№,\ Пол,\ ДатаНародж,\ Посада,\ Оклад}$ (СПІВРОБІТНИК) – фрагмент таблиці розташовано на сайті 1.

$C_1=\pi_{Таб№,\ Відділ№,\ ПІБ}$ (СПІВРОБІТНИК) – фрагмент таблиці розташовано на сайті 1.

$C_{21}= \delta_{Відділ№} =’Д_1’(C_2)$ – фрагмент таблиці розташовано на сайті 1.

$C_{22}=\delta_{Відділ№} =’Д_2’(C_2)$ – фрагмент таблиці розташовано на сайті 2.

$C_{23}=\delta_{Відділ№} =’Д_3’(C_2)$ – фрагмент таблиці розташовано на сайті 3.

Прозорість фрагментації

Прозорість фрагментації (або незалежність від фрагментації) є самим верхнім рівнем прозорості розподілення. Якщо РСУБД забезпечує прозорість фрагментації, то користувачеві не потрібно знати, як саме фрагментовані дані. У цьому випадку доступ до даних здійснюється на основі глобальної схеми й користувачеві немає необхідності вказувати імена фрагментів або розташування даних. Наприклад, для вибірки відомостей про всіх керівників відділень (у них атрибут Посада має значення 'менеджер'), при наявності в системі прозорості фрагментації, можна користуватися наступним SQL-оператором:

```
SELECT ПІБ
FROM СПІВРОБІТНИК
WHERE Посада = 'менеджер';
```

Це той же самий SQL-оператор, що треба було б увести для одержання зазначених результатів у централізованій системі.

Таким чином, якщо забезпечується прозорість фрагментації, користувачі одержують дані у вигляді деякого подання, у якому фрагменти логічно скомбіновані за допомогою відповідних операцій з'єднання й об'єднання. До обов'язків *системного оптимізатора* ставиться визначення фрагментів, до яких потрібен фізичний доступ для виконання кожного із запитів, що надійшли, користувача.

Прозорість розташування. Прозорість розташування являє собою середній рівень прозорості розподілення. У цьому випадку користувач повинен мати відомості *про* способи фрагментації даних у системі, але не має потреби у відомостях про розташування даних. При наявності в системі прозорості розташування той же самий запит варто переписати в такому виді:

```
SELECT ПІБ
FROM C21
WHERE Таб#  IN
     (SELECT Таб#  FROM Cl  WHERE Посада ='менеджер')  UNION
SELECT ПІБ FROM C22 WHERE Таб#  IN
     (SELECT Таб#  FROM Cl  WHERE Посада ='менеджер')  UNION
SELECT ПІБ FROM C23 WHERE Таб#  IN
(SELECT Таб#  FROM Cl  WHERE Посада ='менеджер');
```

У цьому випадку в запиті необхідно вказувати імена використовуваних фрагментів. Крім того, додатково необхідно скористатися операціями з'єднання (або підзапитами), оскільки атрибути Посада і ПІБ перебувають у різних вертикальних фрагментах. Основна перевага прозорості розташування в тім, що база даних може бути піддана фізичної реорганізації, і це не зробить ніякого впливу на програми додатків, що здійснюють до неї доступ. Зокрема, дані можуть бути перенесені з одного вузла на інший, і це не повинне зажадати внесення яких-небудь змін у їхні програми, що використають, або дії користувачів. Така переносімось бажана, оскільки вона дозволяє переміщати дані в мережі відповідно до змін вимог до ефективності роботи системи.

Прозорість реплікації. Із прозорістю розташування дуже тісно зв'язаний ще один тип прозорості - прозорість реплікації. Він означає, що користувачеві не потрібно мати відомості про існуючу реплікацію фрагментів. Під прозорістю реплікації мається на увазі прозорість розташування реплік. Для користувачів повинна бути створена таке середовище, щоб вони, принаймні, з логічної точки зору, могли вважати, що дані не дублюються. Прозорість реплікації (як і прозорість фрагментації, і прозорість розташування) є досить бажаної, оскільки вона спрощує створення користувальницьких програм і виконання термінальних операцій. Зокрема, прозорість реплікації дозволяє створювати й знищувати дублікати в будь-який момент відповідно до змін вимог, не зачіпаючи при цьому ніякі з користувальницьких додатків або, термінальних операцій.

З вимог прозорості реплікації треба, що до обов'язків системних оптимізаторів також ставиться визначення, який саме з фізичних дублікатів буде застосований для доступу до даних при виконанні кожного уведеного користувачем запиту.

Треба сказати, що можуть існувати системи, які не забезпечують прозорості розташування, але підтримують прозорість реплікації.

Прозорість локального відображення. Це найнижчий рівень прозорості розподілення. При наявності в системі прозорості локального відображення користувачеві необхідно вказувати як імена використовуваних фрагментів, так і розташування відповідних елементів даних. Той же самий запит у системі із прозорістю локального відображення здобуває наступний вид:

```
SELECT ПІБ FROM C21 AT SITE 1 WHERE Таб# IN.
(SELECT Таб#  FROM Cl  AT  SITE 1
WHERE Посада ='менеджер')  UNION
SELECT ПІБ FROM C22  AT  SITE 2 WHERE  Таб#  IN
SELECT Таб#  FROM Cl  AT  SITE  1 WHERE Посада ='менеджер')  UNION
SELECT ПІБ FROM C23  AT «ITE  3 WHERE  Таб#  IN
     (SELECT Таб#  FROM Cl  AT  SITE  1
WHERE  Посада='менеджер');
```

З міркувань наочності мова SQL доповнена новим ключовим словом AT SITE, що дозволяє вказати, де саме розташований необхідний фрагмент даних. Очевидно, що в цьому випадку запит має більше складний вид і на його підготовку буде потрібно більше часу, чим у двох попередніх випадках. Малоймовірно, щоб система, що надає тільки такий рівень прозорості розподілення, була прийнятна для кінцевого користувача.

Прозорість іменування. Прямим наслідком варіантів, що обговорювалися вище, прозорості розподілення є **вимога наявності прозорості іменування.** Як і у випадку централізованої бази даних, кожний елемент розподіленої бази даних повинен мати унікальне ім'я. Отже, РСУБД повинна гарантувати, що ніякі два вузли системи не зможуть створити деякий об'єкт бази даних, що має те саме ім'я. Одним з варіантів рішення цієї проблеми є створення центрального **сервера імен,** що буде відповідати за повну унікальність всіх існуючих у системі імен. Однак подібному до підходу властиві такі недоліки, як:

- рангова та певної частини локальної автономії;
- поява проблем із продуктивністю (оскільки вузол із сервером імен перетворюється у вузьке місце всієї системи);
- зниження доступності - якщо вузол із сервером імен з якої-небудь причини стане недоступним, всі інші сайти системи не зможуть створювати нових об'єктів бази даних.

Альтернативне рішення складається у використанні префіксів, що поміщають в імена об'єктів як ідентифікатор вузла, що створювали цей об'єкт.

Наприклад, відношення ДЕТАЛІ, створене на вузлу S_1, могло б одержати ім'я S_1.ДЕТАЛІ. Аналогічним образом, необхідно мати можливість ідентифікувати кожний фрагмент і кожну його копію. Тому другої копії третього фрагмента відносини ДЕТАЛІ, створеного на вузлу S_1 можна було б привласнити ім'я S_1.ДЕТАЛІ.Ф3.К2. Однак подібний підхід приводить до втрати прозорості розподілення.

Підхід, що дозволяє перебороти недоліки, що належать обом згаданим методам, складається у використанні **аліасів** (псевдонімів), створюваних для кожного з об'єктів бази даних. У результаті об'єкт S_1.ДЕТАЛІ.Ф3.К2 користувачам вузла може бути відомий під ім'ям ДЕТАЛІ_ЛОКАЛЬНИЙ. Завдання перетворення аліасу в щире ім'я відповідного об'єкта бази даних покладають на РСУБД.

Прозорість транзакцій. Прозорість транзакцій у середовищі розподілених СУБД означає, що при виконанні будь-яких розподілених транзакцій гарантується збереження цілісності й погодженості розподіленої бази даних.

Розподілена транзакція здійснює доступ до даним, що зберігають більш ніж в одному місці розташування. Наприклад, окрема транзакція може містити в собі виконання коду на багатьох сайтах, зокрема це можуть бути операції відновлення, виконувані на декількох вузлах. Кожна із транзакцій розділяється на трохи **субтранзакцій** — по однієї для кожного вузла, до даних якого здійснюється доступ.

На вилучених вузлах субтранзакции представляються **агентами,** де під агентом розуміється процес, що виконується для даної транзакції на окремому вузлу. Система повинна знати, що два агенти є елементами однієї й тієї ж транзакції, не повинні виявлятися в стані взаємного блокування.

Розподілена транзакція. Розглянемо виконання транзакції Т, що виконує роздруківку імен усього персоналу компанії, при використанні схеми фрагментації, певної вище у вигляді фрагментів C_1 C_2, C_{21}, C_{22} і C_{23}.

Транзакція буде включати три субтранзакции T_{C1}, T_{C2} і T_{C3}, представлені агентами на вузлах 1, 2 і 3 відповідно. Кожна із субтранзакцій друкує імена працівників локального відділення компанії.

Графік розподіленої транзакції показаний у табл. 1.1. Природна паралельність, властиву системі, - кожна із субтранзакцій виконується на своєму вузлу паралельно з іншими.

Таблиця 1.1 Приклад розподіленої транзакції

Час	Транзакція T_{C1}	Транзакція T_{C2}	Транзакція T_{C3}

T_1	Begin_transaction	Begin_transaction	Begin_transaction
T_2	Read(ПІБ)	Read(ПІБ)	Read(ПІБ)
T_3	Print(ПІБ)	Print(ПІБ)	Print(ПІБ)
T_4	End_ transaction	End_ transaction	End_ transaction

Атомарність залишається фундаментальною концепцією поняття транзакції й у випадку розподілених транзакцій, однак додатково РСУБД повинна гарантувати атомарність і кожної з її субтранзакцій . Отже, РСУБД повинна гарантувати не тільки синхронізацію субтранзакцій з іншими локальними транзакціями, що виконуються паралельно з ними на одному вузлу, але й забезпечити синхронізацію субтранзакцій із глобальними транзакціями, що виконуються одночасно з ними на цьому й іншому вузлах системи. Прозорість транзакцій у розподілених СУБД додатково ускладняється за рахунок наявності фрагментації, розподілу даних і використання реплікації. Існує два додаткових аспекти прозорості транзакцій, таких як **прозорість паралельності** й **прозорість відмов.**

Прозорість паралельності. Прозорість паралельності забезпечується РСУБД у тому випадку, якщо результати всіх паралельно виконуваних транзакцій (як розподілених, так і нерозподілених) генеруються незалежно і є логічно погодяться з результатами, які були б отримані в тому випадку, якби всі ці транзакції виконувалися послідовно в деякому довільному порядку, по одній у кожний момент часу. Існують деякі фундаментальні принципи (наприклад, використання механізму блокування), які використаються для керування паралельним доступом як у централізованих СУБД, так і в розподілених системах. Однак у випадку розподілених СУБД мають місце додаткові ускладнення, пов'язані з необхідністю гарантувати, що як глобальні, так і локальні транзакції не можуть робити впливу один на одного. Крім того, РСУБД повинні гарантувати погодженість всіх субтранзакцій кожної глобальної транзакції.

Наявність у системі реплікації ще більше ускладнює проблему організації паралельної обробки в системі. Якщо одна з копій реплицируемих даних піддається відновленню, відомості про це, в остаточному підсумку, повинні бути представлені в кожній з існуючих копій. У цьому випадку найбільш очевидна стратегія - зробити поширення відомостей про зміну частиною вихідної транзакції, оформивши його як ще одну атомарну операцію. Однак якщо один з утримуючу копію змінених даних вузлів виявиться в момент внесення зміни недоступним через відмову на самому вузлу або в каналі зв'язку, то виконання транзакції буде відкладено доти, поки цей вузол знову не стане доступним.

Якщо існує велика кількість копій даних, то ймовірність успішного завершення транзакції зменшується в експонентній залежності від їхнього числа. Така ж ситуація може виникнути не тільки для операцій відновлення, але й для операцій вибірки. Таким чином, описана стратегія має небажаний побічний ефект - зниження рівня продуктивності й доступності як для вибірки, так і для відновлення. Альтернативною стратегією є обмеження поширення відомостей про зміну тільки тими вузлами, які в цей момент доступні. На інші вузли відомості про зміну надійдуть, як тільки вони знову стануть доступними.

Додатковою стратегією могла б бути видача дозволу обновляти копії асинхронно, через якийсь час після внесення вихідного відновлення. Затримка у відновленні цілісності може варіюватися від декількох секунд до декількох годин. У більшості розподілених систем керування паралельністю базується на механізмі блокування, точно так, як і в не розподілених системах.

Прозорість відмов. Будь-яка централізована СУБД повинна включати механізм відновлення, що у підданої різним порушенням і відмовам середовищу буде гарантувати атомарність виконання транзакцій - або всі операції транзакції будуть успішно завершені, або жодна з них (принцип "всі або нічого"). Більше того, якщо результати виконання транзакції були зафіксовані, внесені нею зміни здобувають тривалий' (або постійний) характер. У централізованих СУБД можуть мати місце різні типи відмов: збої системи, відмова носіїв, помилки в програмах, недбалість персоналу, стихійні лиха й дії зловмисників. У розподіленому середовищі РСУБД повинна додатково враховувати наступне:

- можливість втрати повідомлення;
- можливість відмови лінії зв'язку;
- аварійний останов одного з вузлів;
- розчленовування мережних з'єднань.

Щоб забезпечити атомарність глобальної транзакції в розподіленому середовищі, РСУБД повинна гарантувати, що вся безліч стосовних до даної транзакції агентів або зафіксувало свої Результати, або виконало відкіт. Отже, РСУБД повинна синхронізувати виконання глобальної транзакції таким чином, щоб мати гарантії, що всі її субтранзакции були успішно j завершені до того, як почалася фінальна операція фіксації результатів всієї глобальної транзакції. Наприклад, розглянемо глобальну транзакцію, що виконує відновлення даних на двох сайтах, C_1 і C_2. Нехай субтранзакция на сайті C_1 завершується успішно й фіксує свої результати, однак субтранзакция на сайті C_2 не може успішно завершити своє виконання й робить відкіт внесених змін для збереження цілісності локальної бази даних. У результаті розподілена транзакція виявляється в неузгодженому стані, оскільки через властивість ; тривалості транзакцій результати виконання транзакції на C_1 скасувати вже не можна. Одним з коректних методів відновлення розподілених баз даних є протокол двофазної фіксації транзакції.

Прозорість виконання. Прозорість виконання вимагає, щоб робота в середовищі РСУБД виконувалася точно так само, як і в середовищі централізованої СУБД. У розподіленому середовищі робота системи не повинна демонструвати ніякого зниження продуктивності, пов'язаного з її розподіленою архітектурою, наприклад із присутністю повільних мережних з'єднань. Прозорість виконання вимагає, щоб РСУБД була здатна знаходити найбільш ефективні стратегії виконання запитів.

У централізованої СУБД оброблювач запитів повинен оцінювати кожний запит на доступ до даних і знаходити оптимальну стратегію його виконання, що представляє собою впорядковану послідовність операцій з базою даних. У розподіленому середовищі оброблювач розподілених запитів (системний оптимізатор) перетворить запит на доступ до даних в упорядковану послідовність операцій локальних баз даних. Додаткова складність виникає через необхідність ураховувати наявність фрагментації, реплікації й певної схеми розміщення даних. Системний оптимізатор повинен з'ясувати:

- до якого фрагмента варто звернутися;
- яку копію фрагмента використати, якщо його дані репліцируются;
- яке з місць розташування повинне використатися .

Оброблювач розподілених запитів виробляє стратегію виконання, що є оптимальною з погляду деякої вартісної функції. Звичайно розподілені запити оцінюються за такими показниками:

- час доступу, що включає фізичний доступ до даних на диску;
- час роботи CPU, затрачуване на обробку даних у первинній пам'яті;
- час, необхідне для передачі даних по мережних з'єднаннях.

Перші два фактори аналогічні тим, які враховуються в централізованих системах. Однак у розподіленому середовищі РСУБД необхідно враховувати й витрати на передачу даних, які в багатьох випадках серед інших витрат виявляються домінуючими. Останнє зауваження буде особливо справедливо у випадку використання повільних мережних з'єднань, характерних для глобальних мереж. Швидкість передачі даних у таких каналах може становити лише трохи кілобайт у секунду. У подібних ситуаціях при оптимізації можна ігнорувати витрати на ввід-висновок і використання ЦП. Однак деякі мережні з'єднання мають швидкість передачі даних, порівнянну зі швидкістю доступу до дисків (наприклад, локальні мережні з'єднання). У цьому випадку при оптимізації повинні враховуватися всі три показники витрат.

Прозорість використання СУБД . Прозорість використання СУБД дозволяє сховати від користувача РСУБД той факт, що на різних сайтах можуть функціонувати різні локальні

СУБД. Тому даний тип прозорисгі застосуємо тільки у випадку гетерогенних розподілених систем. Як правило, це один із самих складних у реалізації типів прозорості. У дійсності, усе, що необхідно, щоб екземпляри СУБД на різних вузлах **усе разом підтримували той самий інтерфейс,** і зовсім необов'язково, щоб це були копії однієї й тієї ж версії СУБД. Наприклад, СУБД Ingres і Oracle обидві підтримують офіційний стандарт мови SQL, а виходить, можна домогтися, щоб вузол із СУБД Ingres і вузол із СУБД Oracle обмінювалися повідомленнями між собою в контексті розподіленої системи.

Підтримка гетерогенності досить приваблива. На практиці сучасне ПЗ використається не тільки на багатьох різних комп'ютерах і в середовищі різних операційних | систем. Воно досить часто використається й з різними СУБД, і було б дуже добре, якби різні СУБД можна було б включити в розподілену систему. Інакше кажучи, ідеальна] розподілена система повинна забезпечувати прозорість використання СУБД.

Концепція прозорості не може бути віднесена до типу "всі або нічого", оскільки можливо її організацію на декількох різних рівнях. Кожний з рівнів має на увазі наявність певного набору угод між вузлами системи. Так, при повній прозорості вузли повинні додержуватися загальної угоди з питань моделі даних, інтерпретації схем, подання даних. У непрозорих системах єдиною угодою є формат обміну даними й набір функціональних можливостей кожного вузла системи.

З погляду кінцевого користувача повна прозорість бажана. Але з погляду адміністратора локальної бази повністю прозорий доступ пов'язаний із труднощами здійснення контролю. Традиційний спосіб роботи з поданнями, використовуваний для побудови захисного механізму, у такій ситуації може виявитися неефективним. Наприклад, механізм роботи з поданнями мови SQL дозволяє обмежувати доступ до базових таблиць або підмножини даних базової таблиці й дозволяти його тільки конкретним користувачам. Однак він не дозволяє так само легко обмежувати доступ до даних на основі набору критеріїв, відмінних від імені користувача.

Простіше забезпечити подібний тип функціональності за допомогою процедур, які будуть запускатися вилученим користувачем. У цьому випадку локальні користувачі зможуть працювати з даними так само, як вони працювали колись, при використанні стандартного механізму захисту СУБД. Однак вилучені користувачі зможуть одержувати тільки такий тип доступу до даних, що реалізований у наданому їм наборі процедур, так, як це робиться в об’єктно - орієнтованої системі. Подібний тип архітектури реалізувати простіше, ніж забезпечити повну прозорість системи, до того ж він надає більше високий рівень локальної автономності.

1.10 Дванадцять правил Дейта для РСУБД

Основою для побудови всіх цих правил є те, що розподілена СУБД повинна сприйматися кінцевим користувачем точно так само, як і звична йому централізована СУБД. Даного правила подібні із дванадцятьома правилами Кодда для реляційних систем.

0. Основний принцип. З погляду кінцевого користувача розподілена система Повинна виглядати в точності так, як і звичайна, нерозподілена система.

1. Локальна незалежність .Вузли в розподіленій системі повинні бути незалежні, або автономні. У даному контексті автономність означає наступне:

- локальні дані належать локальним власникам і виконуються локально;
- всі локальні процеси залишаються чисто локальними;
- всі процеси на заданому вузлу контролюються тільки цим вузлом.

2. Відсутність опори на центральний вузол. У системі не повинне бути жодного вузла, без якого система не зможе функціонувати. Це означає, що в системі не повинне існувати центральних серверів таких служб, як керування транзакціями, виявлення взаємних блокувань, оптимізація запитів і керування глобальним системним каталогом.

3. Безперервне функціонування. В ідеалі, у системі ніколи не повинна виникати потреба в плановому останове її функціонування для виконання таких операцій, як:

- додавання або видалення вузла із системи;
- динамічне створення або видалення фрагментів з одного або декількох вузлів.

4. Незалежність від розташування. Незалежність від розташування еквівалентна прозорості розташування. Користувач повинен одержувати доступ до бази даних з кожного з вузлів. Більше того, користувач повинен одержувати доступ ***до*** будь-яким даних так, ніби вони зберігалися на його вузлу, незалежно від того, де вони фізично зберігаються.

5. Незалежність від фрагментації. Користувач повинен одержувати доступ до даних незалежно від способу їхньої фрагментації.

6. Незалежність від реплікації. Користувач не повинен мати потребу у відомостях про наявність реплікації даних. Це значить, що користувач не буде мати засобів для одержання прямого доступу до конкретної копії елемента даних, а також не повинен піклуватися про відновлення всіх наявних копій елемента даних.

7. Обробка розподілених запитів. Система повинна підтримувати обробку запитів, що посилаються на дані, розташовані на більш ніж одному сайті.

8. Керування розподіленими транзакціями. Система повинна підтримувати виконання транзакцій як одиниці відновлення. Система повинна гарантувати, що виконання як глобальних, так і локальних транзакцій буде відбуватися зі збереженням чотирьох основних властивостей транзакцій, а саме: атомарности, погодженості, ізольованості й тривалості.

9. Апаратна незалежність

РСУБД повинна бути здатна функціонувати на встаткуванні з різними обчислювальними платформами.

10. Незалежність від операційної системи

Прямим наслідком попереднього правила є вимога, відповідно до якого СУРБД повинна бути здатна функціонувати під керуванням різних операційних систем.

11. Незалежність від мережі. РСУБД повинна бути здатна функціонувати в мережах з різною архітектурою та типами носія.

12. Незалежність від типу СУБД. РСУБД повинна бути здатна функціонувати поверх різних локальних СУБД, можливо, з різним типом використовуваної моделі даних. Інакше кажучи, РСУБД повинна підтримувати гетерогенність.

Контрольні питання.

1. Визначення розподіленої інформаційної системи і її можливості.
2. Призначення РСУБД.
3. Різновиди розподіленої системи СУБД.
4. Основні архітектури побудови РБД.
5. Структура розподіленої системи з поділом дисків.
6. Що визначає поняття кластер?
7. Яка різниця між гетеро та гомогенними РБД.
8. Принцип побудови мультибазової розподіленої системи
9. Переваги розподіленої системи обробки БД.
10. Недоліки розподіленої системи обробки БД.

2. Структура, адміністрування та створювання баз даних у Oracle DB 11g

2.1 Загальні відомості, склад, можливості СУБД

Корпорація Oracle є найбільшим в світі постачальником корпоративного програмного забезпечення і пропонує повний комплекс технологій для побудови IT-інфраструктури та управління сучасним підприємством: сімейство базових програмних технологій Oracle11g, Oracle12g, готове рішення для колективної роботи Oracle Collaboration Suite, повнофункціональний комплекс бізнес-додатків Oracle E-Business Suite і інтеграційне рішення для управління даними Oracle Data Hub. Корпорація надає свої продукти і послуги в області консалтингу, навчання та технічної підтримки більш ніж в 150 країнах світу. Офіційний сайт корпорації - www.oracle.com.

На сьогоднішній день СУБД Oracle - це інтегроване програмних забезпечення для мережі розподілених обчислень Grid. Для реалізації складних комплексних рішень є широкий набір програмних продуктів, який можна розбити на кілька розділів по їх функціональному призначенню.

У всі варіанти СУБД Oracle включені мови і інтерфейси, що дозволяють отримувати дані з бази і маніпулювати ними. Для доступу до даних в Oracle можна використовувати SQL, ODBC, JDBC, SQLJ, OLE DB, ODP.NET, SQL / XML, XQuery і WebDAV. Програми, що зберігаються в самій базі даних, можуть бути написані на мовах PL / SQL і Java.

PL / SQL- це розроблене Oracle процедурне розширення мови SQL. Зазвичай на ньому реалізуються логічні програмні модулі для додатків. Мовою PL / SQL можна писати збережені процедури, тригери, цикли, умовні пропозиції і обробку помилок. Процедури на PL / SQL можна скомпілювати і зберегти в базі даних. Блоки, написані на PL / SQL, можна також виконувати безпосередньо за допомогою інтерактивного інструменту SQL * Plus, наявного у всіх версіях Oracle. Програмні блоки на PL / SQL можна скомпілювати заздалегідь.

У Oracle8 мова Java вперше почав використовуватися для написання збережених процедур, а віртуальна Java-машина (JVM) була вбудована безпосередньо в СУБД. JVM забезпечує підтримку написання на Java збережених процедур, методів і тригерів, а також технологій Enterprise JavaBeans ™ (EJB), CORBA, HOP і HTTP.

Включення Java в СУБД Oracle дозволяє програмістам, що володіє Java, застосувати свої знання у розробку додатків для Oracle. Java-програми можна розгортати на стороні клієнта, всередині сервера додатків або в базі даних - в залежності від конкретних обставин. Oracle Database llg включає JIT-компілятор Java, по замовчуванням активований.

СУБД Oracle славиться вмінням обробляти дуже великі обсяги даних і підтримувати безліч одночасно працюючих користувачів. Oracle не тільки добре масштабується для розгортання на все більш потужних одиночних системах, але може бути розгорнута і в розподіленої конфігурації. Примірники Oracle, розгорнуті на декількох платформах, можна об'єднати, представивши у вигляді логічно єдиної розподіленої бази даних.

Oracle включає (рис.2.1):

• засоби розробки додатків бази даних;

• засоби встановлення з'єднання з базою даних;
• розподілені бази даних;
• засоби переміщення даних;
• засоби підвищення продуктивності;
• засоби управління базою даних;
• засоби забезпечення безпеки бази даних.

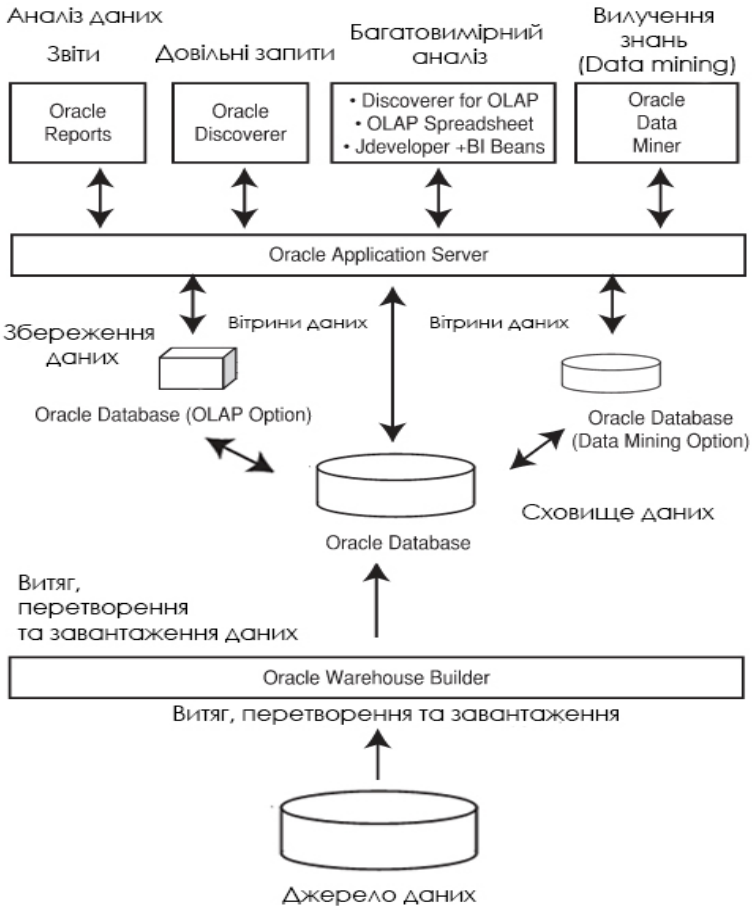


Рисунок 2.1 Складові програм Oracle для збереження та обробки даних

Переваги Oracle:
правильну стратегію оновлення портфеля хмарних продуктів, підтримка сумісності зі старими рішеннями високі показники задоволеності продуктами.

Недоліки Oracle:
складнощі ліцензування, проблеми, пов'язані з підтримкою клієнтів, випуск оновлень, щоб спростити установку виправлень, випускає їх один раз в квартал, щоб залучити клієнтів до своїх хмарних сервісів, Oracle збільшила кількість процесорних ліцензій, необхідних для запуску програмного забезпечення компанії в конкуруючих хмарних інфраструктурах, що призвело до подвоєння вартості використання продуктів Oracle, обмежила окремі функції Oracle Cloud і локальних інженерних систем.

2.2 Архітектура сервера Oracle 11g

Архітектура серверу Oracle складається з логічних структур, процесів та файлів для збереження даних та параметрів конфігурації (рис.2.2).

Кожна база даних Oracle має пов'язаний з нею ***екземпляр.*** Організація екземпляра дозволяє СУБД обслуговувати безліч типів транзакцій, забезпечувати високу

продуктивність, цілісність даних і безпеку. Екземпляр Oracle - складний комплекс структур пам'яті та процесів операційної системи (рис.2.3). Термін ***процес*** означає будь-яке завдання, виконувану без втручання користувача.

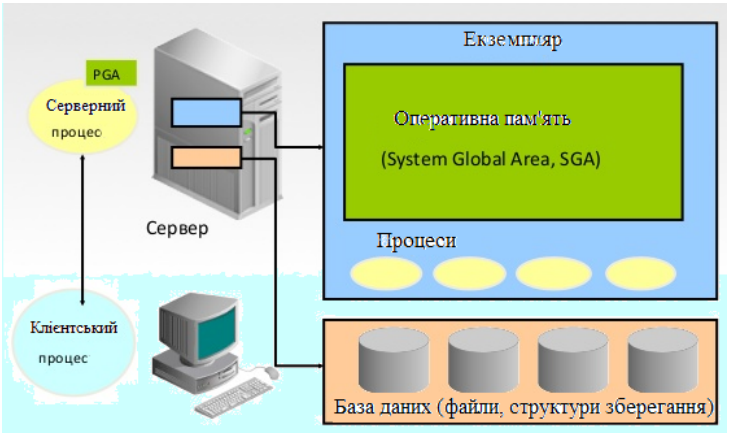


Рисунок 2.2 Архітектура серверу Oracle 11g

Відкриття БД Oracle включає три стадії:

- формування екземпляра Oracle (передустановча стадія).
- установка бази даних екземпляром (настановна стадія).
- відкриття БД (стадія відкриття).

Екземпляр, у якому немає бази даних, називається ***незайнятим (idle),*** але він займає пам'ять і не виконує ніякої роботи.

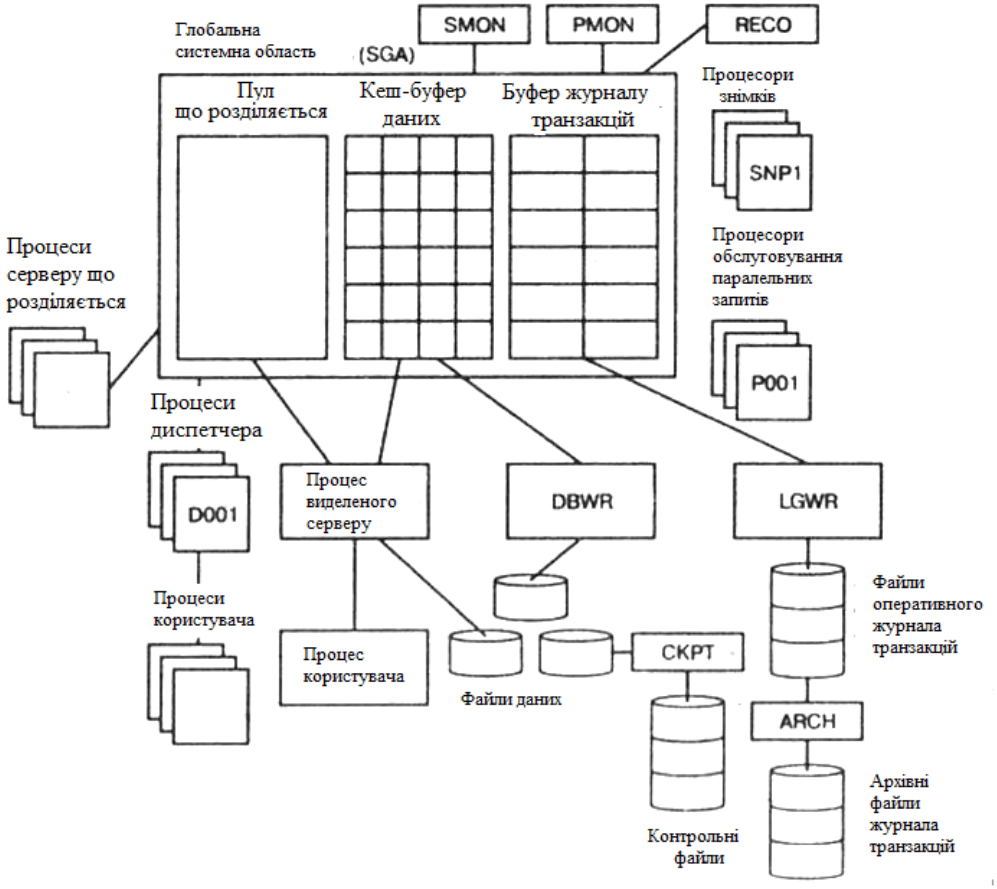


Рисунок 2.3 Структура екземпляра Oracle 11g

Екземпляр може приєднатися тільки до однієї БД, а доти, поки не буде використано ***Parallel Server***, БД може бути підключена тільки до одного екземпляра ***Oracle.***

Екземпляр - це мозок системи обробки даних. В екземплярі виконуються всі операції, у той час як у БД зберігаються всі дані.

Більшість настроювань екземпляра пов'язане з компонентами в глобальній системній області. Але крім них існують і деякі опції настроювання, що стосуються фонових процесів.

Системна глобальна область (System Global Area - SGA) Oracle (рис.2.4) - це група спільно використовуваних областей пам'яті, де зберігається інформація для примірника (кеш виразів, буфери журналів повторного виконання і кеш-буфер даних).

Програмна глобальна область (Program Global Area- PGA) і глобальна область користувача (User Global Area - UGA) - спільно використовувані області пам'яті, що містять дані і керуючу інформацію для серверних процесів і призначених для користувача сеансів.

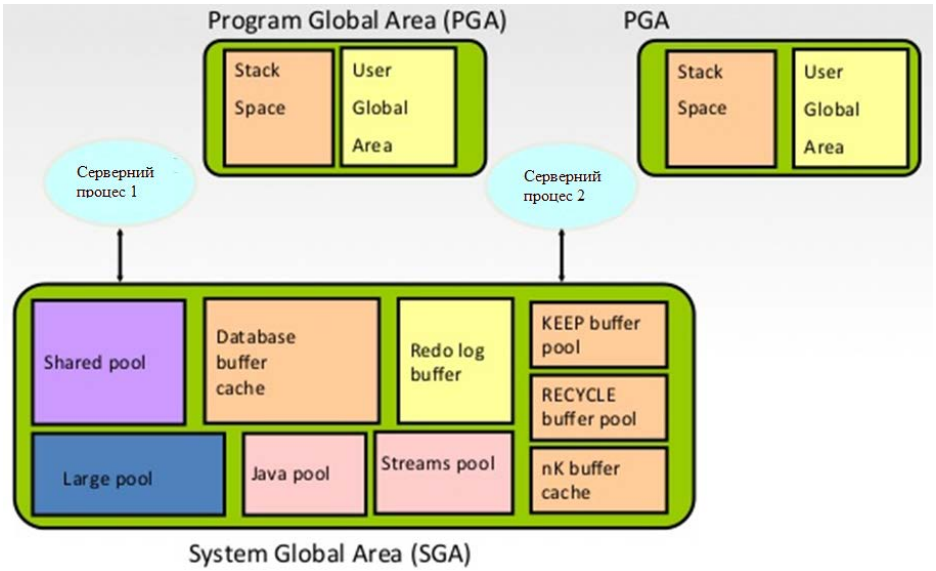


Рисунок 2.4 Структура системної глобальної область

Oracle підтримує кілька примірників на одній машині, але фонові процеси не є загальними. Наприклад, для трьох примірників на одній машині потрібно три набору фонових процесів. Тому на одній машині зазвичай рекомендується мати одну базу даних, один екземпляр і кілька схем. В SGA (System Global Area) зберігаються структури пам'яті, необхідні для маніпулювання даними, аналізу пропозицій SQL і кэширования транзакцій. До цієї області одночасно має доступ безліч процесів, які можуть зчитувати дані з її або модифікувати їх. Всі операції із БД використають інформацію, що перебуває в SGA. SGA виділяється відразу ж після створення екземпляра ще на передустановчої стадії. Звільняється ця область тільки після повного вимикання екземпляра.

Екземпляр Oracle являє собою складний комплекс взаємодіючих процесів. SGA складається з наступних компонентів:

- поділюваний пул (SHARED POOL);
- кэш-буфер даних (DATABASE BUFFER CACHE);
- буфер журналу транзакцій (REDO LOG BUFFER);
- структури сервера багатозадачного середовища (MULTI-THREADED SERVER - MTS).

Поділюваний пул містить кэш бібліотеки, кэш словника й керуючі структури сервера. Кэш бібліотеки зберігає план виконання пропозицій SQL. Тут утримуються заголовки пакетів PL/SQL і процедур, що виконувалися раніше. Кэш словника зберігає рядка словника даних, які були використані для лексичного аналізу пропозицій SQL. Сервер Oracle використає кэш бібліотеки для підвищення швидкості виконання операторів SQL.

Розмір поділюваного пула задається параметром SHARED POOL SIZE у файлі INIT.ORA. Розмірність значення параметра - байти. Обсяг пула необхідно замовляти.

Кэш-буфер даних складається із блоків пам'яті того ж розміру, що й блоки ORACLE. Всі дані, з якими працює СУБД, першою справою завантажуються в кэш-буфер. У цих же блоках пам'яті виконується й будь-яке відновлення даних. Тому дуже важливо правильно встановити розмір буфера.

СУБД переноси дані на диск - відповідно до порядку їхнього розміщення в списку LRU (LEAST RECENTLY USED/ Цей список відслідковує звертання до блоків даних і враховує частоту обігів. Якщо виконується звертання до блоку даних, що зберігається в кэш-буфере, то воно міститься в той кінець списку, що зветься MRU (MOST RECENTLY USED/ Якщо серверу потрібне місце в кэш-буфере для завантаження нового блоку, то він вирішує цей момент за допомогою списку LRU - який із блоків перенести на диск, щоб звільнити місце для нового. Блоки, найбільш віддалені в списку від MRU, - кандидати на видалення з кэш-буфера. Таким чином, довше всього залишаються в кэш-буфере ті блоки, до яких найчастіше виконується обіг.

Модифіковані блоки називаються ***брудними (dirty)*** і містяться у відповідний ***dirty-список.***

Буфер журналу транзакцій. Дані про транзакції зберігаються в цьому буфері доти, поки не будуть переписані у файл оперативного Журналу транзакцій. Після заповнення буфера його вміст переноситься у файл журналу транзакцій.

Розмір буфера журналу транзакцій задається параметром LOGJBUFFER. Значення параметра вказує розмір буфера в байтах.

У результаті ви одержите час, що користувальницькі процеси перебували в стані очікування при звертанні до буфера журналу транзакцій.

2.3 Фонові процеси Oracle

У процесі роботи СУБД ORACLE переглядає тисячі записів у таблицях даних, відповідає на сотні запитів користувачів одночасно. Вся робота розподіляється між безліччю програм, які працюють незалежно одна від іншої, причому в кожної своя роль. У сукупності ці програми називаються фоновими процесами ORACLE.

Безліч фонових процесів Oracle включає: SMON, PMON, RECO, DBWR, SNPn, LGWR, Dnnn, ARCH, LCKn, KPT, Pnnn

Процеси USER і SERVER виконують обробку транзакцій кінцевого користувача БД. Хоча ці процеси й не кваліфікуються як фонові, вони відіграють значну роль у функціонуванні системи.

Процеси PMON і SMON — це процеси моніторів **БД.** Після запуску **БД** процес **SMON** (System Monitor) виконує автоматичне відновлення екземпляра. Якщо при останнім вимиканні БД що-небудь не було завершено, SMON автоматично запускає незавершені операції. Крім того, він стежить за сегментами БД, фіксує звільнення простору в тимчасових сегментах і автоматично поєднує їх.

PMON — Process Monitor (Монітор процесів) виконує автоматичну "збирання" після раптово припинилися або завершилися аварійно процесів. Ця "збирання" включає видалення сеансу, блокувань, які були їм установлені, неприйнятих транзакцій, звільнення ресурсів глобальної системної області, виділених цьому процесу.

Фонові процеси SMON і PMON повинні бути неодмінно пущені при запуску системи. У противному випадку система функціонувати не буде.

Процес DBWR (DataBase Writer) — відповідає за перенос обновлених блоків, занесених в ***dirty-список*** з кэш-буфера даних у файли даних. Замість того, щоб записувати кожний блок на диск відразу, DBWR чекає, поки не буде виконано одне з умов, а потім переглядає ***dirty-список*** і всі відзначені в ньому блоки переписує на диск.

Процес LGWR (Log Writer) — четвертий і останній фоновий процес, запуск якого обов'язковий для роботи СУБД ORACLE. Цей процес відповідає за перезапис інформації з буфера журналу транзакцій, що перебуває в глобальній системній області, у файли оперативного журналу. ORACLE не вважає транзакцію виконаної доти, поки процес LGWR не перезапише дані про неї з буфера журналу транзакцій у файл. Цей процес рідко служить причиною ускладнень у роботі.

Фонові процеси-диспетчери Dnnn. Зупинимося докладніше на цих процесах. Всі процеси сервера можуть закріплюватися за певними користувальницькими процесами, або використатися декількома процесами користувача. В останньому випадку вони називаються "поділюваними" (shared) процесами або серверами. Для роботи із застосуванням поділюваного сервера необхідна установка Multi Threaded Server (MTS). При використанні поділюваних процесів, у системі повинен існувати як мінімум один процес диспетчер. Процес-диспетчер передає запити користувачів у чергу SGA і повертає відповіді сервера відповідному процесу користувача. Фоновий процес Dnnn, де nnn - це число, що задає у файлі init.ora. Параметр указує протокол, що буде використаний диспетчером і кількість процесів.

Додатковий фоновий процес ARCH (Archiver) відповідає за копіювання повністю заповненого оперативного журналу транзакцій в архівні файли журналів транзакцій. Під час перезапису в архів ніякий інший процес не може одержати доступ до журналу. У цей час система повинна перебувати в стані очікування.

Процес СКРТ (Check Point) - це додатковий фоновий процес, що відповідає за обробку контрольних крапок. Необхідність у ньому виникає, коли треба знизити навантаження на LGWR.

Процес RECO (Recovery) відповідає за відновлення незавершених транзакцій у розподіленій системі БД. Коли виникає підозріла транзакція, RECO виконує свої функції автоматично без втручання адміністратора БД.

Процес SNPn (Snapshot) виконує автоматичні відновлення знімків БД і запускає процедури відповідно до розкладу, зафіксованим у пакеті DBMS_JOB. У файлі initora задається кількість процесів, що запускають, SNPn і тривалість інтервалу, протягом якого процес "засипає" перш ніж виконати завдання.

Процесс LCKn (Parallel Server Lock) — у середовищі, що робить паралельне обслуговування декількох екземплярів БД, на LCKn покладає відповідальність за координацію блокувань установлюваних різними екземплярами БД. Якщо в системі немає паралельного обслуговування, то запуск LCKn не потрібний.

Процеси паралельних запитів (Parallel Query) одержали в системі найменування Pnnn. Сервер Oracle запускає й зупиняє процеси Pnnn залежно від активності роботи із БД і настроювання опцій паралельних запитів. Ці процеси приймають участь у формуванні індексів, таблиць і запитів. Кількість процесів, що запускають, визначається двома параметрами: PARALLEL_MIN SERVERS і PARALLEL MAX SERVERS, що визначають, відповідно, мінімальна й максимальна кількість процесів, що запускають.

Додатки й клієнтські частини зв'язуються із сервером Oracle за допомогою "процесів користувача". Кожний процес користувача має підключення до процесу сервера, що може бути або жорстко пов'язаний з одним процесом користувача, або розподілятися між багатьма. Процес сервера, аналізує й виконує передані йому оператори SQL і повертає результат процесу користувача. Так само процес сервера зчитує блоки даних з файлу даних і розміщає їх у кэш-буфере даних.

Кожному процесу користувача виділяється область пам'яті, що називається "глобальною областю процесу" - PGA (Process Global Area). Уміст PGA залежить від режиму підключення процесу користувача до процесу сервера. Якщо процес користувача має виділений процес сервера, то в PGA розміщається інформація про поточний сеанс роботи користувача, стек і інформація про стан курсору. Якщо ж процес користувача пов'язаний з

поділюваним процесом сервера, то інформація про поточний сеанс і поточний стан курсору зберігається в SGA. Загалом, це не дуже істотно, хоча деяке збільшення розміру SGA все-таки буде потрібно. У цьому й складається різниця між двома типами взаємодії клієнта й сервера в БД Oracle.

2.4 Логічна та фізична структура бази даних

Структура бази даних складається з логічних та фізичних елементів (рис.2.5). До логічної структури відноситься:

Блок даних (data block) - дрібний блок бази даних Oracle, що складається з певної кількості байт на диску.

Екстент (extent) -два або більше послідовних блоків даних Oracle, що представляє собою одиницю виділення місця на диску.

Сегмент (segment) - набір екстентів, які виділяються логічну структуру, як таблиця або індекс (або інший об'єкт).

Табличний простір (tablespace) - набір з одного або більше файлів даних, зазвичай складається з пов'язаних сегментів. Все це, взяте разом, являє собою безліч взаємопов'язаних логічних об'єктів бази, що називається схемою.

Фізична структура складається з файлів даних, що містять в собі всі логічні структури, які є частиною табличного простору, на зразок таблиць і індексів.

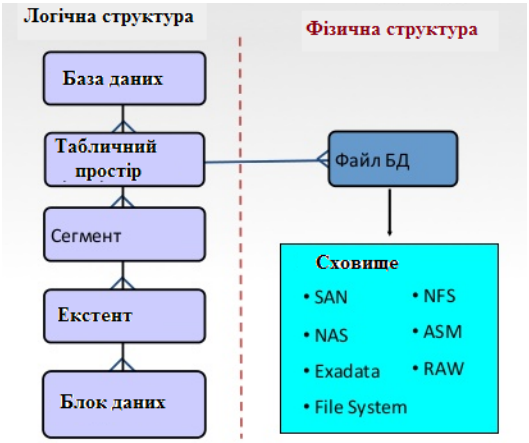


Рисунок 2.5 Логічна та фізична структура БД

Блок даних Oracle - це дрібний логічний компонент бази даних, основа ієрархії зберігання даних і основа сховища бази даних Oracle. Блок даних складається з певного числа байтів дискового простору в системі зберігання операційної системи. База даних Oracle виділяє вільний простір для даних в термінах блоків даних Oracle. Наприклад, ви можете встановити розмір блоку даних в 2, 4, 8, 16 або 32 Кбайт (або навіть більше), і ці блоки даних прийнято називати блоками Oracle. Диски сховища, на яких розташовуються блоки Oracle, самі діляться на блоки даних дискового простору, які представляють собою безперервні області, що зберігають певну кількість байт, наприклад, 4096 або 32768 байт (тобто 4 або 32 Кбайт).

Розмір блоків задається в параметрі DB_BLOCK_SIZE в файлі ініціалізації (init.ora). Розмір блоку - мінімальна одиниця поновлення, вибору або вставки даних. Коли користувач вибирає дані з таблиці, оператор SELECT "читає" дані з файлів бази в одиницях блоків Oracle. Якщо вибрати розмір блоку Oracle в 8 Кбайт, блоки даних міститимуть в точності 8192 байт. Якщо вибрати розмір блоку в 64 Кбайт (65536 байт), то якщо треба витягти ім'я довжиною в чотири символи, доведеться прочитати весь блок розміром 64 Кбайт, в якому містяться цікаві для чотири літери.

Якщо встановити розмір блоку Oracle кратним розміром блоку операційної системи, то можна за рахунок цього виграти 5% продуктивності. У Oracle пропонують керуватися наступними принципами при виборі розміру блоку бази даних. Виберіть мінімальний розмір блоку, якщо збережені записи малі і доступ до них найчастіше довільний. Вибирайте більший розмір блоку, якщо рядки малі, але доступ до них в основному послідовний (або довільний і послідовний), або ж якщо рядки великі. Малий розмір блоку зручний, якщо ви працюєте з дрібними записами і виконуєте великий обсяг пошуку за індексом.

Більший розмір блоків підходить для додатків, які формують звіти, виконуючи сканування великих таблиць. Якщо ви не впевнені у виборі розміру блоку, Oracle рекомендує розмір блоку в 8 Кбайт для систем, що виконують велику кількість транзакцій.

У Oracle створюється п'ять табличних просторів (рис.2.6) які представляють собою табличні простори за замовчуванням, що повинні мати всі бази даних:

- табличний простір System;
- табличний простір Sysaux;
- табличний простір Undo;
- табличний простір Temporary;
- постійне табличний простір за замовчуванням.

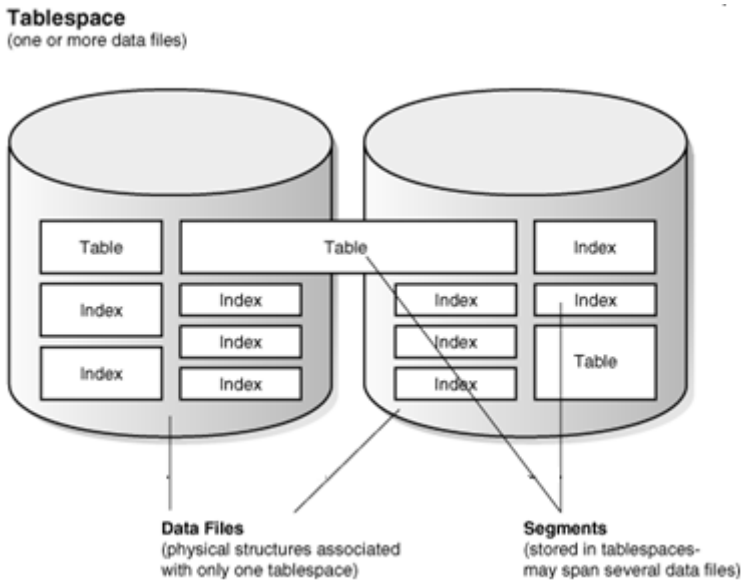


Рисунок 2.6 Табличні просторі Oracle

Під час створення бази даних тільки два табличних простору повинні бути присутніми обов'язково. Це табличний простір System (ключове табличний простір Oracle, що містить дані словника Oracle) та табличний простір Sysaux (яке є допоміжним табличним простором для System і містить дані, які використовуються різними продуктами і засобами Oracle).

За розміром табличні простори класифікуються наступним чином. Табличні простору Bigfile з єдиним великим файлом, чий максимальний розмір знаходиться в межах від 8 до 128 Тбайт, в залежності від розміру блоку бази даних. Таким чином, ваша база даних може імовірно зберігатися тільки в одному табличному просторі Bigfiile. Табличні простору Smallfile може включати безліч файлів даних, але файли не можуть бути такими ж великими, як в Bigfile. Табличні простору Smallfile є табличними просторами, що створюються за замовчуванням в Oracle Database 11g, тому Oracle створює System і Sysaux саме як табличні простору Smallfile. Табличні простору Temporary містять дані, які існують лише на час сеансу роботи користувача. Зазвичай Oracle використовує ці табличні просторі для сортування та подібних дій користувачів.

Також табличні простори класифікуються як табличні простори Permanent включають всі табличні простору, які не є Temporary. Табличні простору Undo містять записи скасування, які Oracle використовує для відкату, або скасування змін, в базі даних. Табличні простору Read-only не допускають операцій записи в свої файли даних. Ви можете перетворити будь-яке нормальне табличний простір в простір тільки для читання, щоб захистити дані або виключити необхідність у виконанні резервного копіювання та відновлення великих файлів даних, які не змінюються.

База даних складається з файлів - файлів даних і системних файлів Oracle. Ці файли самі по собі не приносять користі, якщо ви якимось ніяк не контактуєте з ними, що вимагає допомоги з боку операційної системи, яка забезпечує можливості і ресурси обробки, такі як пам'ять, щоб дозволити маніпулювати даними на дискових пристроях. Коли ви комбінуєте специфічний набір процесів, створених Oracle на сервері з пам'яттю, виділеної йому операційною системою, ви отримуєте екземпляр Oracle. Коли говорять "база даних запущена", насправді "екземпляр запущений". Сама база даних в формі набору фізичних файлів, з яких вона складається, ніяк не використовується, якщо не запущено екземпляр. Примірник виконує всю необхідну роботу в базі даних. Зазвичай між базою і екземпляром існує відношення "один до одного", на відміну від Microsoft SQL Server, де кожен екземпляр може підтримувати кілька баз даних.

База даних Oracle складається з наступних трьох основних типів файлів:

- Файли даних (.dbf). Ці файли зберігають дані таблиць і індексів.
- Контрольні файли. Ці файли записують зміни всіх структур баз даних.
- Файли журналів повторного виконання. Ці онлайнові файли містять зміни, проведені в табличних даних.

На додаток до цих трьох типів файлів база даних Oracle використовує кілька інших системних файлів операційної системи для управління своїми операціями.
- ініціалізаціоні файли (init.ora і файл параметрів сервера [SPFILE]),
- файли мережевого адміністрування (tnsnames.ora і listener.ora),
- файли журналів попереджень,
- файли трасування,
- файл паролів,
- файли резервних копій, з яких виконується відновлення бази в разі збою носія.

Кілька комп'ютерів можуть розділяти доступ до даних, встановлюючи кластери, Oracle Real Application Clusters (RAS). Oracle RAS складається з безлічі екземплярів, які працюють на безлічі групових комп'ютерів, що взаємодіють один з одним по мережі (Рис.2.7).

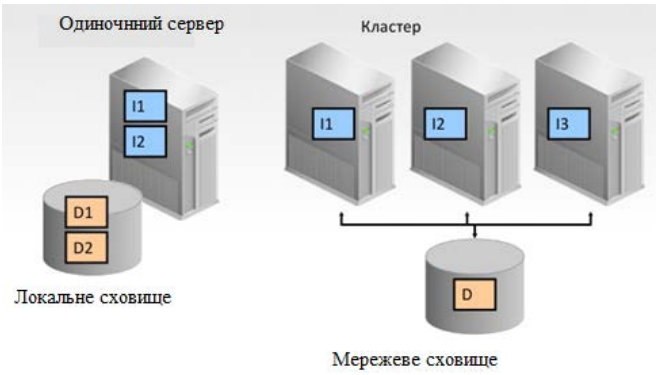


Рисунок 2.7 Варіанти конфігурацій БД Oracle

Для встановлення кластера використовується програма Oracle Clusterware для доступу до бази даних, запущеної на розділеної дискової системі. Забезпечуючи обчислювальний процес міццю безлічі серверів, Oracle RAS забезпечує стійкість, масштабованість і високу

готовність. Ви можете легко підвищувати обчислювальну потужність, просто додаючи нові вузли для доступу до бази даних.

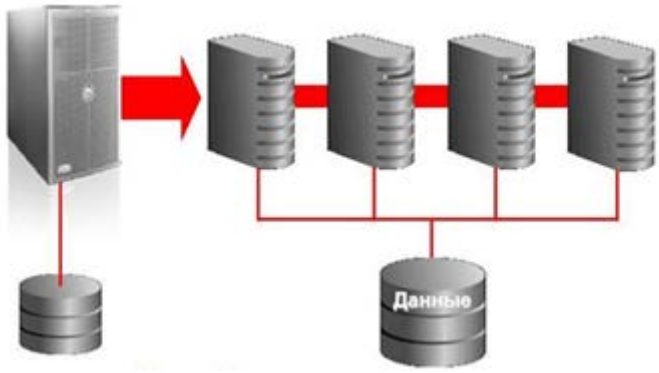


Рисунок 2.8 Масштабованість Oracle Real Application Clusters

Основні переваги використання Oracle RAC:

- масштабованість і висока доступність;
- поступове нарощування обсягу інформації, що зберігається;
- додаткові ресурси по запиту з нульовим простоєм.

Використання Oracle RAC забезпечує паралельну обробку даних в залежності від розміру таблиці яка створена SQL-запитом (рис.2.9).

Рисунок 2.9 Варіанти обробки запитів у Oracle RAC

2.5 Основні об'єкти БД

Oracle підтримує реляційну модель даних, тому, природно, що до числа основних об'єктів Oracle ставляться: ***таблиця, подання, користувач.***

Таблиця (TABLE) є базовою структурою реляційної моделі. Вся інформація в базі даних зберігається в таблицях. Таблиці складаються з безлічі пойменованих стовпців або атрибутів. Безліч значень стовпця визначено за допомогою обмежень цілісності, тобто підтримується обмежена концепція домена. Повне ім'я таблиці в базі даних складається з імені схеми, у даній реалізації співпадаючому з ім'ям користувача, що створив таблицю, і властиво імені таблиці. Таблиця може бути порожній або складатися з однієї або більше рядків значень атрибутів. Рядка значень атрибутів таблиці називаються також ***кортежами.***

Уявлення(VIEW) — це пойменована, динамічно підтримувана сервером вибірка з однієї або декількох таблиць. Оператор SELECT, що визначає вибірку, обмежує видимі користувачем дані. Сервер гарантує актуальність подання, тобто формування подання (матеріалізація відповідного запиту) виробляється щораз при використанні подання. Використовуючи подання, адміністратор безпеки обмежує доступну користувачам частина бази даних тільки тими даними, які реально необхідні для виконання його роботи.

Користувач (USER) — об'єкт, що володіє можливістю створювати й використати інші об'єкти ORACLE, а також запитувати виконання функцій сервера. До числа таких функцій ставляться організація сесії, зміна стану сервера й бази даних, запит на виконання операторів SQL і ін. Для спрощення рішення завдань ідентифікації й іменування в базі даних ORACLE підтримує об'єкти: ***послідовність*** і ***синонім.***

Послідовність (SEQUENCE) — це об'єкт, що забезпечує генерацію унікальних номерів в умовах багатокористувального асинхронного доступу. Звичайно елементи послідовності використаються для вставки унікальних ідентифікаційних номерів для елементів таблиць бази даних.

Синонім (SYNONYM) — це альтернативне ім'я-псевдонім об'єкта ORACLE, що дозволяє користувачам бази даних мати доступ до даного об'єкта. Синонім може бути ***приватним*** і ***загальним.*** Загальний (public) синонім дозволяє всім користувачам бази даних звертатися до відповідного об'єкта по альтернативному імені. Для керування ефективністю доступу до даних Oracle підтримує об'єкти: ***індекс, таблична область*** і ***кластер.***

Індекс (INDEX) — це об'єкт бази даних, створюваний для підвищення продуктивності вибірки даних. Індекс створюється для стовпців таблиці або для подання в просторі бази даних і забезпечує більше швидкий доступ до даних бази даних за рахунок зберігання прямих посилань на місце розташування рядків, що містять необхідні дані.

Таблична область (TABLESPACE) — іменована частина бази даних, використовувана для розподілу пам'яті для таблиць і індексів.

Кластер (CLUSTER) — об'єкт, що задає спосіб спільного зберігання даних декількох таблиць, що містять інформацію, звичайно оброблювану спільно. Кластеризация стовпців таблиць дозволяє зменшити час виконання вибірки. Рядка таблиць, що мають однакове значення в кластеризованных стовпцях, зберігаються в базі даних спеціальним образом.

Для ефективного керування розмежуванням доступу до даних Oracle підтримує об'єкт ***роль.***

Роль (ROLE) — іменована сукупність привілеїв, які можуть бути надані користувачам або іншим ролям. ORACLE підтримує кілька стандартних або визначених ролей. Специфічними для розподілених систем є об'єкти ORACLE: ***знімок*** і ***зв'язок бази даних.***

Знімок (SNAPSHOT) — локальна копія таблиці вилученої бази даних, що використається або для тиражування (копіювання) всієї або частини таблиці, або для тиражування результату запиту даних з декількох таблиць. Знімки можуть бути що ***модифікують*** або ***призначеними тільки для читання.*** Знімки тільки для читання можливо періодично обновляти, відбиваючи зміни основної таблиці. Зміни, зроблені в модифікує знімку, що, поширюються на основну таблицю й інші копії.

Зв'язок бази даних (DATABASE LINK) — це об'єкт бази даних, що дозволяє звернутися до об'єктів вилученої бази даних. Ім'я зв'язку бази даних можна розглядати як посилання на параметри механізму доступу до вилученої бази даних (ім'я вузла, протокол і ін.). Використання одного імені спрощує роботу з об'єктами вилученої бази даних.

Для програмування алгоритмів обробки даних, реалізації механізмів підтримки цілісності бази даних Oracle використає об'єкти: ***процедура, функція, пакет*** і ***тригер.***

Процедура (PROCEDURE) - це пойменований, структурований набір змінних і операторів SQL і PL/SQL, призначений для рішення конкретного завдання.

Функція (FUNCTION) — це пойменований, структурований набір змінних і операторів SQL і PL/SQL, призначений для рішення конкретного завдання й возвращающий Значення змінної.

Пакет (PACKAGE) — це пойменований, структурований набір змінних, процедур і функцій, зв'язаних єдиним функціональним задумом. Наприклад, ORACLE поставляє пакет DBMS_OUTPUT, у якому зібрані процедури й функції, призначені для організації вводу-виводу.

Тригер (TRIGGER) — це збережена процедура, що запускається (автоматично виконується) тоді, коли відбувається пов'язане із тригером подія. Звичайно події пов'язані з виконанням операторів INSERT, UPDATE або DELETE у деякій таблиці.

2.6 Вбудовані типи даних

Для зберігання інформації у таблицях БД використовують наступні вбудовані типи даних (таблиця 2.1)

Таблиця 2.1. Вбудовані типи даних Oracle.

№	Тип даних	Опис
1.	VARCHAR2(size)	Рядок байтів або символів змінної довжини, але не більше ніж size. Максимум 4000 байт.
2.	NVARCHAR2(size)	Рядок символів національного алфавіту змінної довжини, але не більше ніж size. Максимум 4000 байт.
3.	NUMBER [(p [, s])]	Число змінної довжини. Значення p – відповідає за максимальну точність числа (до 38 цифр), s – кількість цифр після крапки. Якщо s – від'ємне, то ціле число округлюється до (–s+1) цифри. Розмірність від 1 до 22 байт.
4.	FLOAT [(p)]	Число з плаваючою крапкою змінної довжини. Значення p – відповідає за максимальну точність числа (до 38 цифр). Розмірність від 1 до 22 байт.
5.	LONG	Символьні дані змінної довжини до 2 гігабайтів, або 2 1 31 - байти. Призначений для зворотної сумісності.
6.	DATE	Діапазон дати від 1-го січня 4712 р. до Різдва, до 31 грудня 31 9999 р. після Різдва. Дата може зберігатися з точністю до секунд. Розмірність 7 байт
7.	BINARY_FLOAT	4-х байтне число з плаваючою крапкою
8.	BINARY_DOUBLE	8-ми байтне число з плаваючою крапкою
9.	TIMESTAMP [(precision)] TIMESTAMP WITH TIME ZONE	Дата й час, з точністю до часток секунди. precision- число десяткових знаків часток секунди, приймає значення від 0 до 9. За замовчуванням 6.
10.	INTERVAL YEAR [(year_precision)] TO MONTH	Інтервал часу, виміряний в роках та місяцях. year_precision- кількість цифр в полі року Year.
11.	INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds)]	Інтервал часу, виміряний в днях, годинах, хвилинах, секундах та частках секунди. day_precision- кількість цифр під дні (від 0 до 9). fractional_seconds – кількість цифр під частки секунди (від 0 до 9).
12.	RAW(size)	Двійкові дані довжиною size – байт. Максимальне значення size -2000 байт.

13.	LONG RAW	Двійкові дані довжиною до 2-х гігабайт.
14.	ROWID	Рядок символів, що містить закодовану інформацію про розміщення рядку таблиці на
15.	UROWID [(size)]	Рядок символів, що містить закодовану інформацію про логічну адресу рядку в індексованій організованій таблиці.
16.	CHAR [(size)]	Рядок байтів або символів постійної довжини size. Максимум 2000 байт.
17.	NCHAR[(size)]	Рядок символів національного алфавіту постійної довжини size. Максимум 2000 байт.
18.	CLOB	Символьний тип даних для дуже великих об’єктів (до 128 терабайт).
19.	NCLOB	Символьний тип даних в кодуванні Unicode (розміром до 128 терабайт).
20.	BLOB	Двійкові дані великого розміру (до 128 терабайт)).
21.	BFILE	Тип даних для роботи з зовнішніми файлами розміром до 4 гігабайт. Містить фізичний шлях до файлу. Відкриває доступ тільки для читання.
22.	XMLType	Для зберігання XML-документу як символьного об’єкту

2.7 Об'єктні типи даних в Oracle

Крім порівняно простих убудованих типів даних — як перейшли зі стандартів *SQL*, так і власних, — в *Oracle* є можливість використати складові. Це типи об'єктів, що конструюються, розраховані на зберігання даних, що мають внутрішню структуру. Ця структура відома *СУБД*, і дозволяє з нею працювати. Об'єктні типи дозволяють зберігати та обробляти засобами *СУБД* "складно влаштовані дані" більше просунутим образом, ніж це дозволяє техніка "більших неструктурованих об'єктів" типів LOB. Через наявність цілком певного типу (*тип колекції*), одиничне об'єктне значення можна уявити за скаляр, хоча воно і не буде атомарним.

Зберігання в стовпцях таблиці значень у вигляді об'єктів, у змісті об'єктного підходу Oracle уперше забезпечила в рамках "об’єктно - реляційної моделі" починаючи з версії Oracle 8. Об'єктні можливості Oracle у загальному додержуються визначень SQL:1999, однак роблять це не пунктуально.

Приклад використання програмувальних (об'єктних) типів. Спочатку потрібно створити "тип", як різновид збережених елементів БД:

```
CREATE TYPE address_type AS OBJECT
 (  zip      CHAR ( 6),
 location  VARCHAR2 ( 200 ) ) ;
```

Типу ADDRESS_TYPE приписані дві "властивості": ZIP і LOCATION. Визначення типу нагадує визначення таблиці, однак на відміну від таблиці тип об'єкта в Oracle не має права містити обмежень цілісності. Якщо необхідно їх указати, зробити це прийде тільки по місцю вживання типу, тобто в описі таблиці.

Відповідно до традицій об'єктного підходу Oracle дозволяє використати тип для створення "буквальних значень" і властивостів об'єктів. "Буквальні значення" фактично дозволяють працювати зі значеннями, що володіють відомої СУБД структурою й однозначно визначаються набором значень елементів своєї структури.

Приклади використання типу ADDRESS_TYPE для визначення стовпця у звичайній таблиці:

```
CREATE TABLE odept1
(
 dname    VARCHAR2 ( 20),
 deptno   NUMBER ( 2)  CONSTRAINT pk_odept1 PRIMARY KEY,
 addr     address_type
);

CREATE TABLE oemp1
(
 ename   VARCHAR2 ( 20),
 empno   NUMBER   ( 4)  CONSTRAINT pk_oemp1 PRIMARY KEY,
 deptno  NUMBER   ( 2)  CONSTRAINT fk_oemp1 REFERENCES dept,
 home    address_type
);
```

Стовпці ADDR і HOME можна назвати "об'єктними атрибутами". Вони не дозволяють зберігати об'єктні значення у вигляді самостійної сутності і посилатися на них посиланнями. Локалізувати такі значення можна тільки за звичайними правилами пошуку даних у таблиці. У виразах явно зазначені об'єктні значення формулюються за допомогою конструктора. На відміну від інших об'єктних систем, наприклад, від Java, в Oracle конструктор має список параметрів, що відповідають властивостям типу.

Приклади застосування в операціях додавання даних:

```
INSERT INTO odept1( deptno, dname, addr )
VALUES
 ( 10, 'RESEARCH', address_type ( '123456', Kiev' )
);
INSERT INTO oemp1 ( empno, ename, deptno, home )
VALUES
 ( 20, 'SMITH', 10, address_type ( '789012', 'Lviv' ) );
```

Oracle допускає певні синтаксичні вільності в записі вираження над об'єктними даними. Тут і далі використаються окремі випадки можливих формулювань. Приклад застосування в запиті про співробітників, "працюючих за місцем проживання", за винятком конкретно зазначеної адреси:

```
SELECT e.ename, d.dname
FROM  oemp1 e INNER JOIN odept1 d
  ON e.home = d.addr
WHERE  e.home <> address_type ( '345678', 'Kiev' );
```

Приклад показує легкість формулювання порівняння складених величин, якимись є адреси. Порівняння здійснюється поелементно, шляхом порівнянням всіх властивостей по черзі.

В виразах можна звертатися до буквального об'єктного значення як до цілого, а можна й до його окремих властивостей. У другому випадку, як правило, потрібно прибігати до псевдоніма:

```
COLUMN home FORMAT A35
SELECT e.ename, e.home, e.home.zip FROM oemp1 e ;
```

Таблиці об'єктів

Створений у БД тип можна вжити та для створення "таблиць об'єктів":

CREATE TABLE addresses1 OF address_type;

Така таблиця завжди містить рівно один стовпець об'єктного типу.

Запис занесення "рядків" у таку таблицю та приклад запиту:

INSERT INTO addresses1 VALUES ('123456', 'Kiev');

SELECT a.*, UPPER (location) FROM addresses1 a;

Об'єкти в таких таблицях зберігаються як самостійні сутності, у яких є автоматично породжуваний СУБД внутрішній унікальний ідентифікатор object ID, відповідно до класичного об'єктного підходу що дозволяє посилатися на конкретні об'єкти з інших таблиць або із програми. Порівняння елементів-"рядків" у таблиці об'єктів один з одним відбувається вже не за значеннями властивостей, як у випадку об'єктного стовпця у звичайній таблиці, а за значенням object ID. Перейти на порівняння значень властивостей дозволяє функція VALUE, наприклад:

SELECT dname FROM odept1 d, addresses1 a WHERE d.addr = VALUE (a);

Зроблено запит про відділи, розташованих по адресах з таблиці ADDRESS1: створення значення зі структурою з об'єкта. Буквальні значення рівняються один з одним за значеннями їхніх властивостей, а об'єкти - за значеннями object ID.

Посилання на об'єкти

Завдяки наявності внутрішніх ідентифікаторів object ID в об'єктах з таблиць об'єктів і можливості на них посилатися локалізувати такі об'єкти стає можливим із застосуванням не тільки SQL запитом, але й навігації за допомогою посилань.

Приклад створення звичайної таблиці зі стовпцем для посилання на збережений у таблиці об'єктів (типу ADDRESS_TYPE) елемент-об'єкт:

```
CREATE TABLE odept2 (
  dname    VARCHAR2 ( 50),
 deptno   NUMBER CONSTRAINT pk_odept PRIMARY KEY,
 addr     REF address_type SCOPE IS addresses1);
```

Опис посилання звужений можливістю адресуватися тільки до об'єктів з таблиці ADDRESSES1. Допускаються варіанти опису посилання: її націленість на вміст конкретної таблиці можна не застосовувати або ж, навпроти, підсилити до аналогії з посилальною .

Приклад заповнення поля ADDR значенням-посиланням:

```
INSERT INTO odept2
   ( dname, deptno, addr )
VALUES
```

```
  ( 'RESEARCH', 10, ( SELECT REF ( a )
                      FROM addresses1 a
                      WHERE a.location = 'Kiev'   ) );
```

Приклад припустимих оформлень звертання до властивостей об'єкта через посилання:

```
COLUMN deref(d.addr) FORMAT A40
SELECT d.dname, DEREF ( d.addr ), d.addr.zip FROM odept2 d ;
```

Навігація по об'єктах у БД за допомогою посилань і в тому числі витяг об'єкта із БД можливі не тільки в запитах SQL, але й у програмі на PL/SQL.

2.8 Приклад використання методів об'єктів

Більше складні конструкції в описі типу дозволяють задавати методи об'єктів і типів. Приклад вказівки в типі методу:

```
CREATE TYPE employee_type AS OBJECT
(  name     VARCHAR2 ( 50),
hiredate DATE,
 home     REF address_type,
  MEMBER FUNCTION days_at_company RETURN NUMBER
);
```

Коли методів багато, їхні заголовки перераховуються по черзі через кому, загальним списком із властивостями.

В описі *типу* приводиться тільки *заголовок* методів. Для опису *тіла* методу необхідно створити *тіло типу* (повна аналогія пари *пакет — тіло пакета*, наявної в PL/SQL):

```
CREATE TYPE BODY employee_type AS
MEMBER FUNCTION days_at_company RETURN NUMBER IS
   BEGIN
   RETURN TRUNC ( SYSDATE - hiredate );
   END;
END;
```

Приклад використання типу в таблиці, що створено:

```
CREATE TABLE sailors
( ship VARCHAR2 ( 30 ),
emp EMPLOYEE_TYPE );
```

в цьому контексті EMPLOYEE_TYPE - це ім'я об’єктного типу.

Використання конструктора типу для заповнення таблиці:

```
INSERT INTO sailors
VALUES
   ( 'Ninna', employee_type ( 'Frank Naude', SYSDATE, NULL ) );
```

в цьому контексті EMPLOYEE_TYPE - це конструктор.

Використання властивостей і методу типу для запиту до таблиці:

```
COLUMN emp FORMAT a60
SELECT *
FROM sailors;
SELECT x.ship, x.emp.name, x.emp.days_at_company ( )
FROM sailors x;
```

і вказівка дужок після DAYS_AT_COMPANY повідомляє, що це метод, а не властивість

2.9 Колекції

Колекції дозволяють зберігати в поле рядка таблиці відразу безліч значень: скалярних, об'єктів або посилань на об'єкт. Таким чином, у БД вони являють собою ще один спосіб угруповання елементів, додатково до об'єктних таблиць. Oracle відносить колекції до об'єктних можливостей своєї СУБД, і тому допускаються багаторівневі колекції.

Формально колекції визначаються через тип, і тому явного порушення скалярности даних у таблицях Oracle колекції не створюють: у стовпці таблиці як і раніше зберігаються значення певного типу. У діалекті SQL Oracle є два види колекцій: вкладені таблиці (неупорядкована безліч) і масиви VARRAY(упорядкований список). Вони відповідають двом видам колекцій у стандарті SQL, але реалізовані з деякими вільностями.

Вкладені таблиці (nested tables) в Oracle є термін для позначення можливості зберігати в поле рядка відразу безліч значень ("таблицю" значень). Звичайну таблицю в Oracle, деякі стовпці якої описані як "вложеные таблиці", завжди можна перепроектувати в інформаційно рівносильний набір звичайних таблиць із "одиничними" стовпцями.

Приклад застосування вкладених таблиць:

```
CREATE TYPE colourset_typ AS TABLE OF VARCHAR2 ( 32);

CREATE TABLE colour_models
 (  model_type VARCHAR2 ( 12),
colours    colourset_type)
NESTED TABLE colours STORE AS colour_model_colours_tab;
```

фраза NESTED TABLE змушена, але може містити вказівки оптимізації зберігання

```
INSERT INTO colour_models
VALUES (
  'RGB', colourset_typ ( 'RED', 'GREEN', 'BLUE' )
);
```

в цьому контексті COLOURSET_TYP - конструктор колекції

```
COLUMN colours FORMAT A60
SELECT * FROM colour_models;
```

Приклад відкриває та обставина, що технічно перелік самих значень зі збережених у стовпці наборів розташовується не безпосередньо в основній таблиці, а в окремої допоміжній (службової), і їй ми *зобов'язані* дати щонайменше ім'я. Однак крім цього у фразі NESTED

TABLE при створенні основної таблиці ми *маємо право* вказати деякі подробиці організації доступу до цієї службової таблиці й особливості зберігання (наприклад, *табличний простір*).

Массивы VARRAY в Oracle повнутрішній технічній організації зберігають списки в полях типів LOB за правилами, що допускають для цих типів: як разом з рядком таблиці (короткі списки), так і в окремому сегменті LOB (довгі списки). У кожному разі спеціальної допоміжної таблиці для зберігання значень масиву тут не потрібно. Приклад:

```
CREATE TYPE addresslist_typ IS VARRAY ( 5) OF address_type;

CREATE TABLE participants
( name      VARCHAR2 ( 20),
locations addresslist_typ
);
```

При створенні таблиці зі стовпцем-масивом VARRAY не потрібно приводити додаткових вказівок (як для стовпця вложеной таблиці), однак є можливість їх застосувати при необхідності в тім.

Додавання й вибірка даних зовні не відрізняється від здійснюваних для вкладеної таблиці, наприклад:

```
INSERT INTO participants
VALUES
('Einstein', addresslist_typ ( address_type ( '123456', 'Kiev' ), address_type ( '789012', 'Lviv' );
```

Колекція в таблиці може здатися зручним для моделювання даних предметної області, працювати з такими даними в SQL не просто. Використовують особливу функцію ***TABLE***. Вона придумана для "розгортання" елементів колекції в список рядків, до якого можна вже звичним образом застосовувати запити, що охоплюють.

```
SELECT *
FROM   TABLE ( SELECT colours
               FROM   colour_models
               WHERE  model_type = 'RGB');

SELECT *
FROM   TABLE ( SELECT locations
               FROM   participants
               WHERE  name = 'Einstein');
```

Для розгортання багаторівневих колекцій передбачений особливий випадок уживання функції TABLE у сполученні із з'єднанням (join). Крім того, ряд можливостей по програмній обробці колекцій передбачений в PL/SQL.

2.10 Тип XMLTYPE

Цей убудований об'єктний тип для роботи в БД із документами XML з'явився у версії 9.0. До цього найбільш підходящим для зберігання документів XML був тип *CLOB*. Тип XMLTYPE технічно може або як і раніше базуватися на *CLOB*, або мати в БД структуру об'єкта. Крім користувальницької спрямованості тип XMLTYPE активно застосовується в останніх версіях Oracle для внутрішньої організації БД. СУБД і БД Oracle пропонують

широкий спектр можливостей по використанню типу XMLTYPE у зв'язку з документами XML.

Створення та поповнення таблиці, у якій вирішено зберігати описів книг у форматі XML - заради збереження формату джерела або ж у чинність відсутності фіксованої структури в опису книги:

```
CREATE TABLE books
(   id          NUMBER,
description XMLTYPE);

INSERT INTO books
VALUES (100, xmltype ( '<?xml version="1.0"?>
    <cover>
      <title>Java Programming with Oracle JDBC</title>
      <author>Donald Bales</author>
      <publisher>OReilly and Associates</publisher>
      <pubdate>December 2001</pubdate>
      <isbn>0-596-00088-x</isbn>
      <pages>496</pages>
    </cover>'  ));
```

Oracle дозволяє замість конструктора XMLTYPE ('...') написати просто '...'

```
SET LONG 1000
SELECT id, description
FROM books;

SELECT id, b.description. XMLDATA FROM books b;
```

XMLDATA — спеціально створений для XMLTYPE "псевдостолбец". У даному прикладі його може замінити метод GETCLOBVAL() типу XMLTYPE, один з багатьох існуючих.

Приклад вибірки з використанням умови відбору мовою XPath:

```
SELECT id, b.description.GETCLOBVAL ( )
FROM   books b
WHERE  b.description.EXISTSNODE('/cover[author="Donald Bales"]')=1;
```

Таблиці даних XMLTYPE. За аналогією з таблицями об'єктів, проектованих самостійно, можна створювати таблиці документів XMLTYPE:

```
CREATE TABLE xbooks OF XMLTYPE;
```

Працювати з ними можна як і з іншими таблицями об'єктів:

```
INSERT INTO xbooks
VALUES (
  xmltype ( '<?xml version="1.0"?>
     <cover>
       <title>Java Programming with Oracle JDBC</title>
       <author>Donald Bales</author>
```

```
    <publisher>OReilly and Associates</publisher>
    <pubdate>December 2001</pubdate>
    <isbn>0-596-00088-x</isbn>
    <pages>496</pages>
  </cover>'   ));

SELECT x.GETCLOBVAL ( ) FROM xbooks x;
```

У цьому прикладі дані XML будуть зберігатися як *CLOB*. Більше складний приклад — створення таблиць об'єктів типу XMLTYPE, де документи XML зберігаються у вигляді таблиці об'єктів, а не як *CLOB*. От як це могло б виглядати в який-небудь БД типу особами об'єктивно,і в чинність відсутності фіксованої структури опису книгою області:

```
CREATE TABLE oxbooks OF XMLTYPE
  XMLSCHEMA "http://www.oracle.com/xbooks.xsd"
  ELEMENT "Book" ;
```

Щоб таблиця OXBOOKS була в такий спосіб заведена в дійсності, потрібно, щоб схема XML http://www.oracle.com/xbooks.xsd була попередньо визначена ("зареєстрована") в "репозитарии" XML DB (сама XML DB зі своїм репозитарием автоматично включена до складу типовим образом створеної БД). Достоїнство такого опису стовпця XMLTYPE у тім, що він дозволяє зберігати не довільні документи XML, а тільки типізовані схемою XML. Тим самим зареєстрована схема XML використаються як засоб обмеження цілісності збережених даних XML, що накладає в таблиці Oracle за правилами технології XML.

2.11 Створення реляційної бази даних у сервері Oracle Database 11g

2.11.1 Запуск сервера Oracle Database 11g

Після установки Oracle при кожному запуску операційної системи буде автоматично запускатися сервер Oracle - процес oracle.exe, що займає більше 400 Мб оперативної пам'яті.

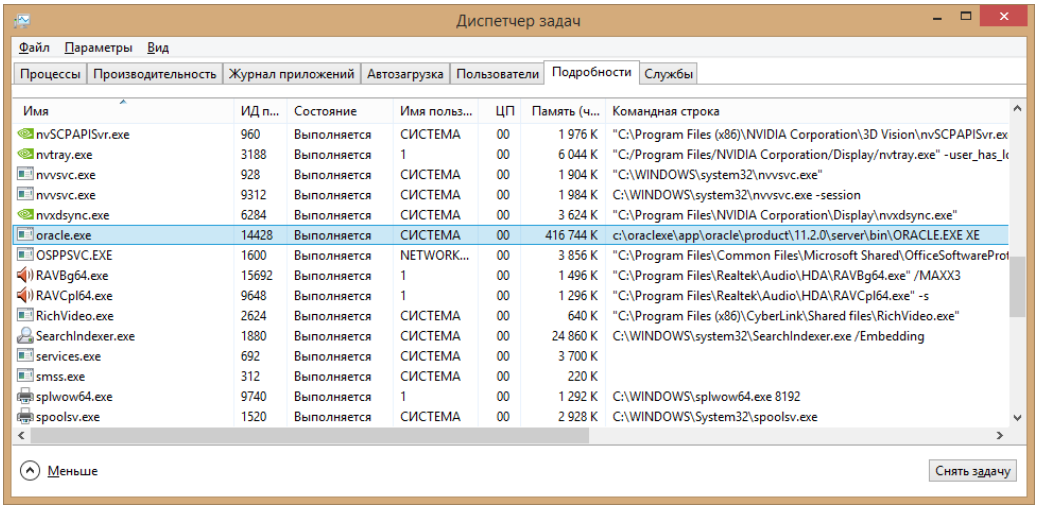


Рисунок 2.10

Він потрібен тільки при роботі з БД, в решту часу він буде лише займати місце в оперативній пам'яті.

Є можливість запускати сервер в ручному режимі тільки тоді, коли він потрібен. Для цього на панелі управління виберемо «Адміністрування» -> «Служби». Знайдемо в списку службу «OracleServiceXE», в контекстному меню виберемо «Властивості» і вкажемо тип запуску «Вручну»:

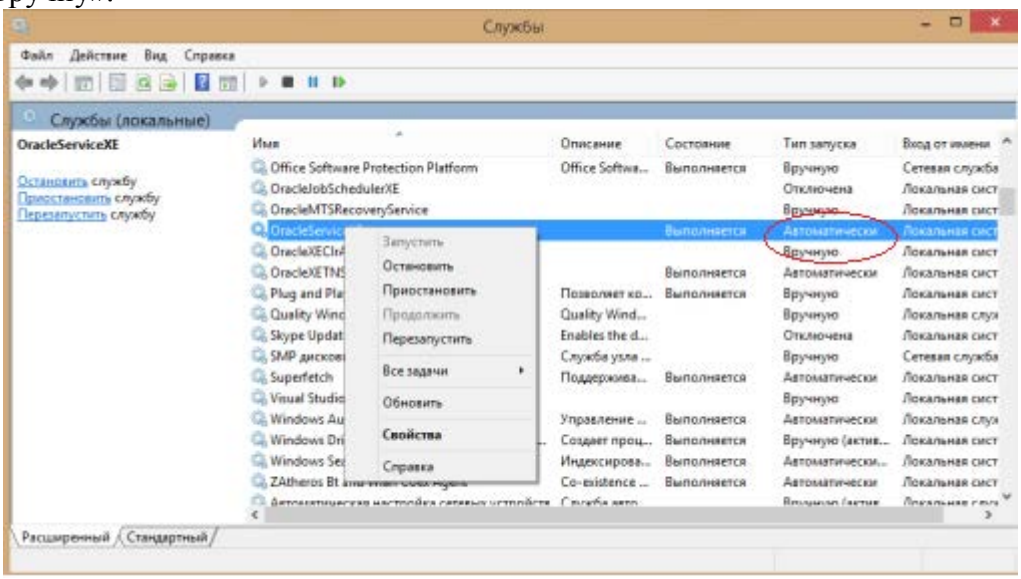


Рисунок 2.11

Для зміни режиму запуску правою клавішею миші покличте контекстне меню та виберіть строку « Свойства». У закладці «Общее» розділу «Тип запуска» встановить «Ручное» (рис.2.12).

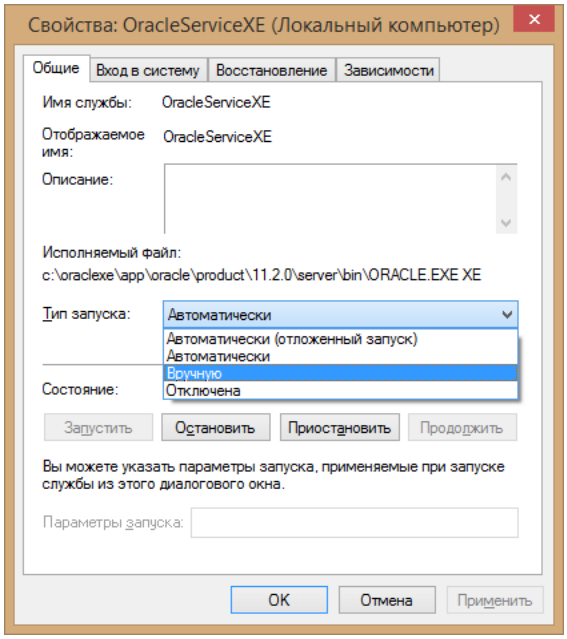
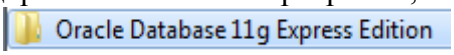
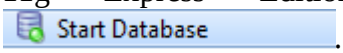


Рисунок 2.12

Варіант «Відключено» в списку вибирати не слід, так як при цьому робота з базою даних буде повністю неможлива. Після цього перед запуском Oracle необхідно в меню «Пуск» вибрати «Oracle 11g Express Edition» і потім «Start Database».

Для запуску серверу відкрийте меню «Усі програми», виберіть у папці Oracle Database 11g Express Edition 

строку Start Database 

. Сервер послідовно запускає усі процеси і у разі успішного старту з'явитися запрошення C:\Windows\system32 (рис.2.10). Якщо об'єкти БД будуть створюватися у інтерактивному редакторі браузера «Get Started», дане вікно можна закрити.

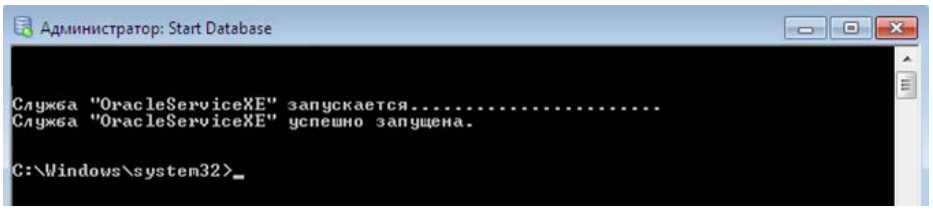


Рисунок 2.10 Запуск серверу Oracle

Для створення об'єктів БД потрібно запустити утиліту «Get Started», для чого с панелі робочого стола виконати команди: ***Пуск \ Всі програми \ Oracale Database 11g \ Get Started*** або відкрить браузер та набрати адресу: http://127.0.0.1:8080/apex/f?p=4950.

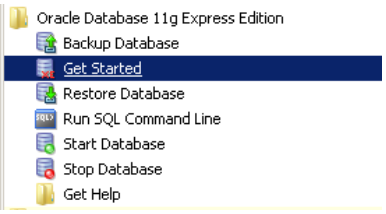


Рис.2.11 Запуск браузера розробника об'єктів серверу Oracle ***Get Started***

Відкриється вікно Oracle Database XE 11.2 у якому ви побачити п'ять закладок (рис.2.12):

- Storage – дозволяє переглянути усі створені БД.
- Session – дозволяє переглянути активні схеми БД.
- Parametrs – дозволяє переглянути і відредагувати параметри логічної структури екземпляру.
- Application Express – для розробки, редагування складу таблиць, уявлень, запитів та інших елементів БД, а також для введення та редагування даних. У даному редакторі можна розробити різні Windows та WEB-форми (додатки щодо роботи з БД) для супроводу таблиць.

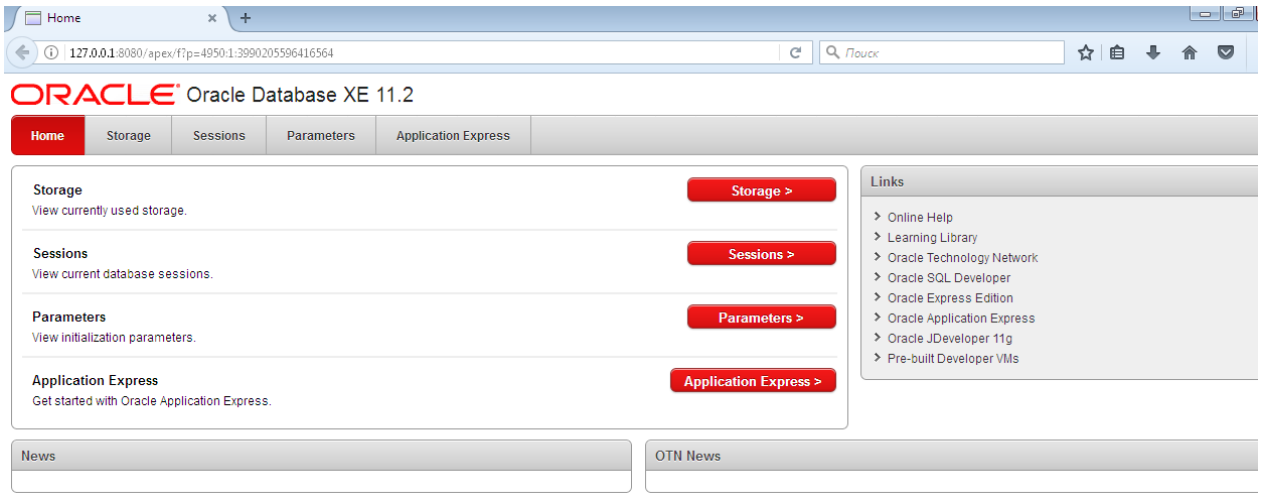


Рис.2.12 Вікно браузера серверу Oracle Database XE 11.2

Для початку розробки таблиць у сервері Oracle потрібно натисніть кнопку ***Application Express***. Відкриється вікно для введення ім'я користувача та паролю (рис.2.13).

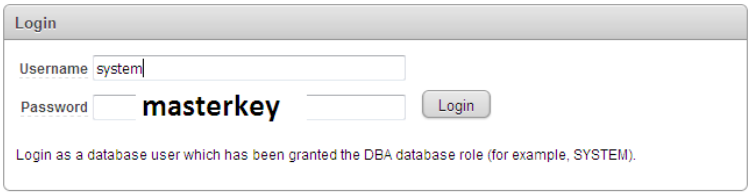


Рисунок 2.13 Реєстрація користувача у Oracle Database XE 11.2

Потрібно заповнити поля: Username **SYSTEM, а у полі** Password уведіть наприклад **masterkey** та натисніть кнопку ***Login***.

2.11.2 Створення таблиць бази даних

Розглянемомо приклад створення БД (**UspStud**) яка представлена на ER-діаграмі. Схема БД містить 3 таблиці: Student, Predmet, USP.

Для створення БД **`UspStud`** потрібно виконати наступні дії. Виберіть команду Create New. Додайте ім'я користувача БД: CAD21**`UspStud`**. Додайте ім'я додатку БД: **`UspStud.`** Введіть пароль: masterkey (рис.2.14).

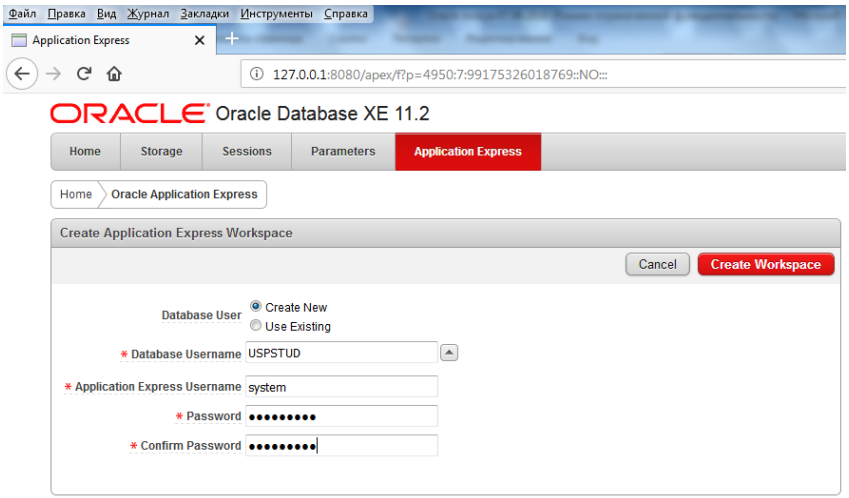


Рисунок 2.14 Реєстрація БД **`UspStud.`**

Натисніть копку Create Workspace. Буде створено робочу область, натисніть на посилання click here to login (рис.2.15).

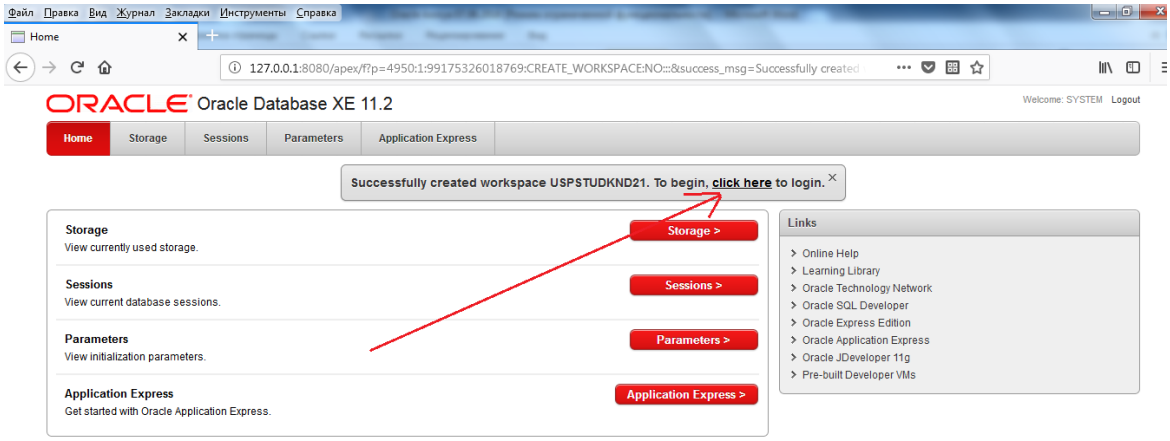


Рисунок 2.15 Створення робочій області БД **UspStud.**

Відкриється вікно у якому введіть пароль , який був записано у попередньому вікні - masterkey та натисніть кнопку Login (рис.2.16).

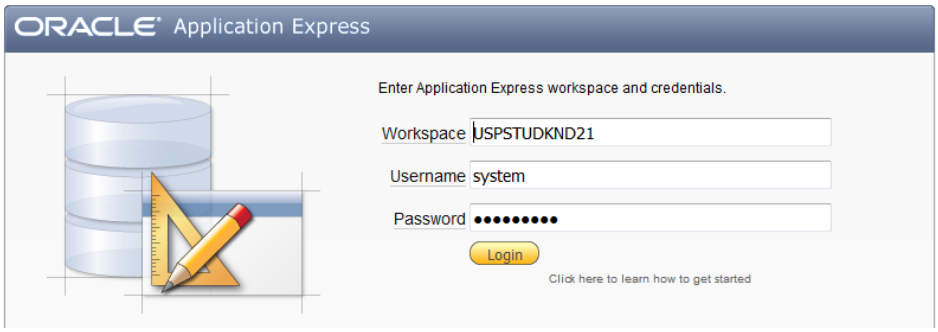


Рисунок 2.16 Запуск БД **UspStud.**

В наступному вікні виберіть редактор об'єктів (рис.2.17)., натиснув піктограму SQL Workshop.

Рисунок 2.17 Вікно майстра для створення об'єктів БД **UspStud.**

Відкриється вікно майстра створення об'єктів. Праворуч, у розділу Schema виберіть ім'я схеми що створюється. Потім натисніть кнопку ***Set*** (рис.2.18). У цьому майстру п'ять піктограм за допомогою яких можливо виконати наступні дії.
Object Browser – перелік об'єктів БД;
SQL Commands – редактор команд DDL, для створення таблиць, уявлень та ін.

SQL Script – редактор тексту команд для створення об’єктів БД, імпортованих з різних систем CASE.
Query Builder – редактор команд DDL, для створення запитів.
Utilities – вікно налаштування властивості БД.

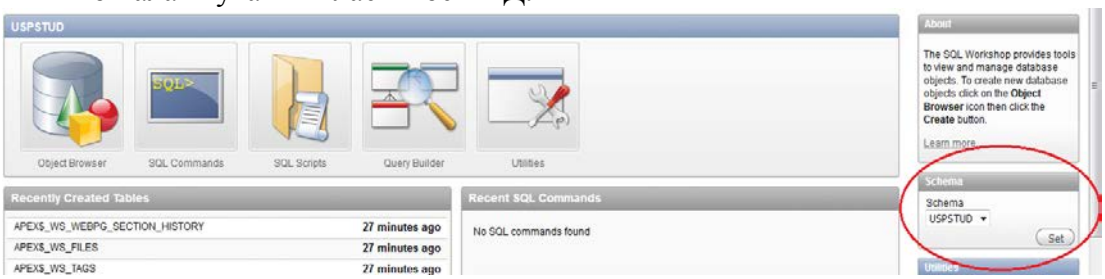


Рисунок 2.18 Вибір схеми БД **UspStud.**

Далі почніть створювати таблицю, для чого праворуч у розділу ***Create Object*** натисніть строчку ***Table*** (Рис.2.19).

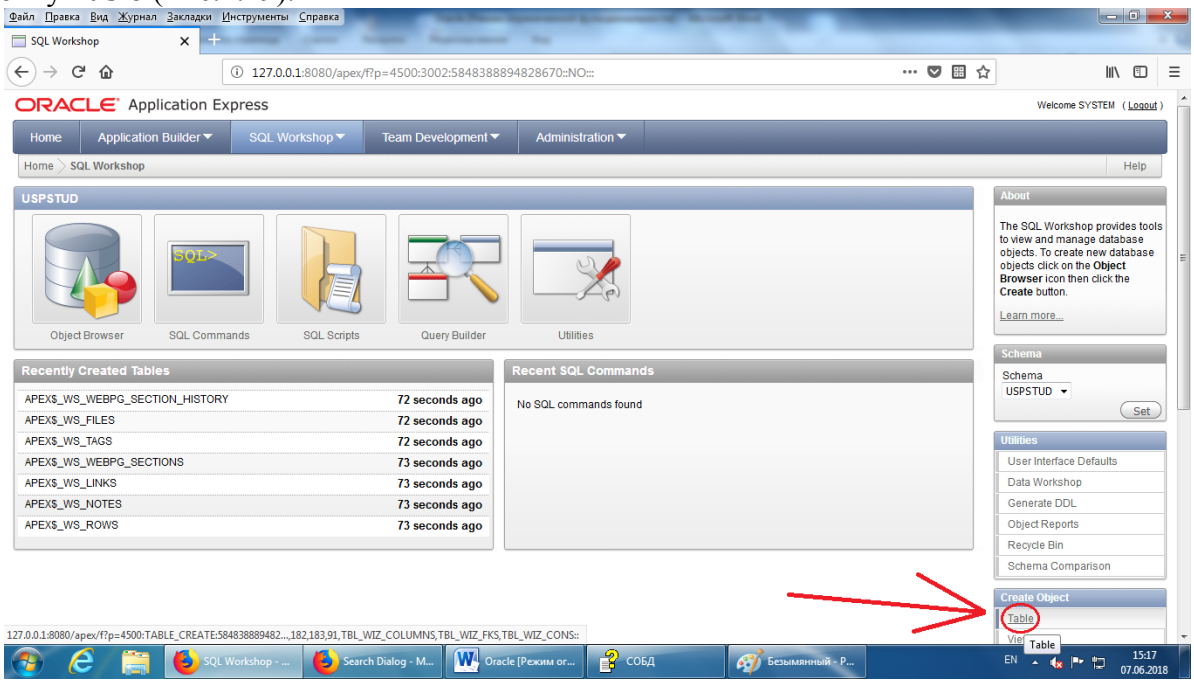


Рисунок 2.19 Вибір для створення об’єкту таблиця БД **UspStud.**

Створить таблицю «STUDENT». У колонці Column Name потрібно ввести назву полів таблиці. На першому місті завжди повинні бути ключові поля, спочатку Primary key, потім - Foreign key . Ці поля повинні мати тип даних числовій (наприклад, Number) та обмеження Not null. Для цього у колонці ***Not null*** потрібно встановити відмітку. Якщо потрібно перенести поле на інше місця у таблиці виконуйте кнопки ***Move*** (Рис.2.20).

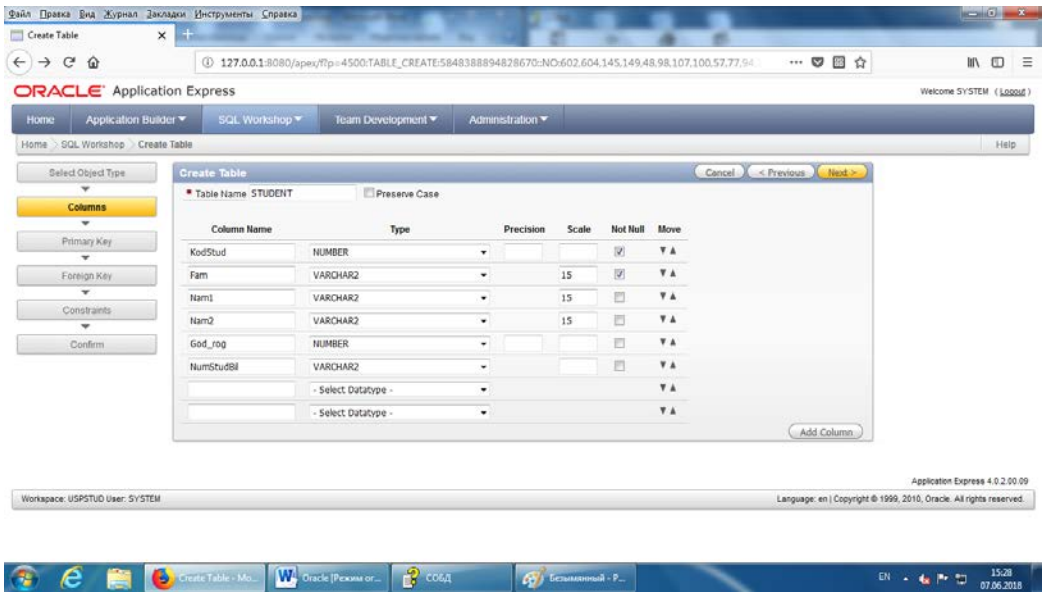


Рисунок 2.20 Створення таблиці STUDENT.

Натисніть у списку ліворуч наступну кнопку «Primary key», виберіть команду «from a new sequence», натисніть групову кнопку у строчці «Primary Key» та виберіть поле «**KodStud**» (Рис.2.21).

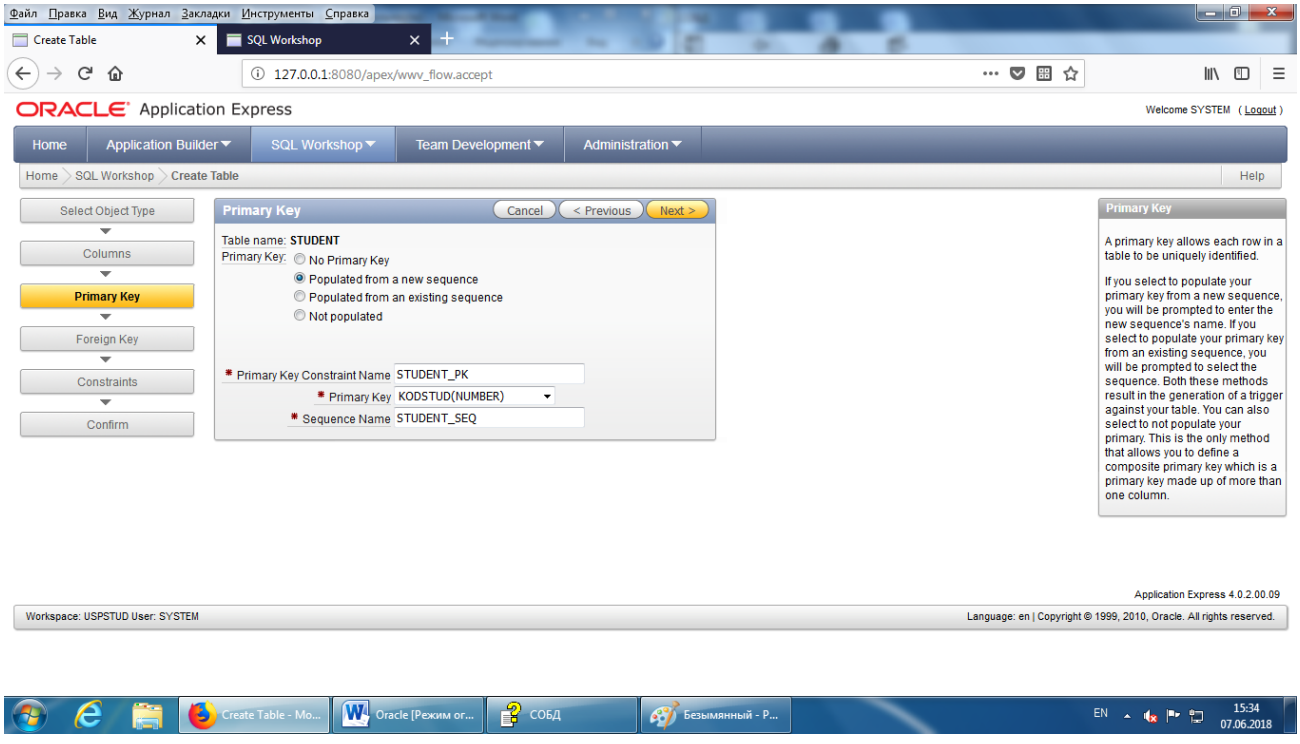


Рисунок 2.21 Створення первинного ключа таблиці STUDENT.

Ця таблиця батьківська, тому не потрібно створювати Foreign key, обмеження «Constraints» також у даної таблиці створювати не будемо.

Натисніть кнопку «Confirm» дослідити текст запиту для створення таблиці, якщо потрібно зробити зміни, повернувшись у команду «Columns» (Рис.2.22). Для створення таблиці натисніть кнопку «Create».

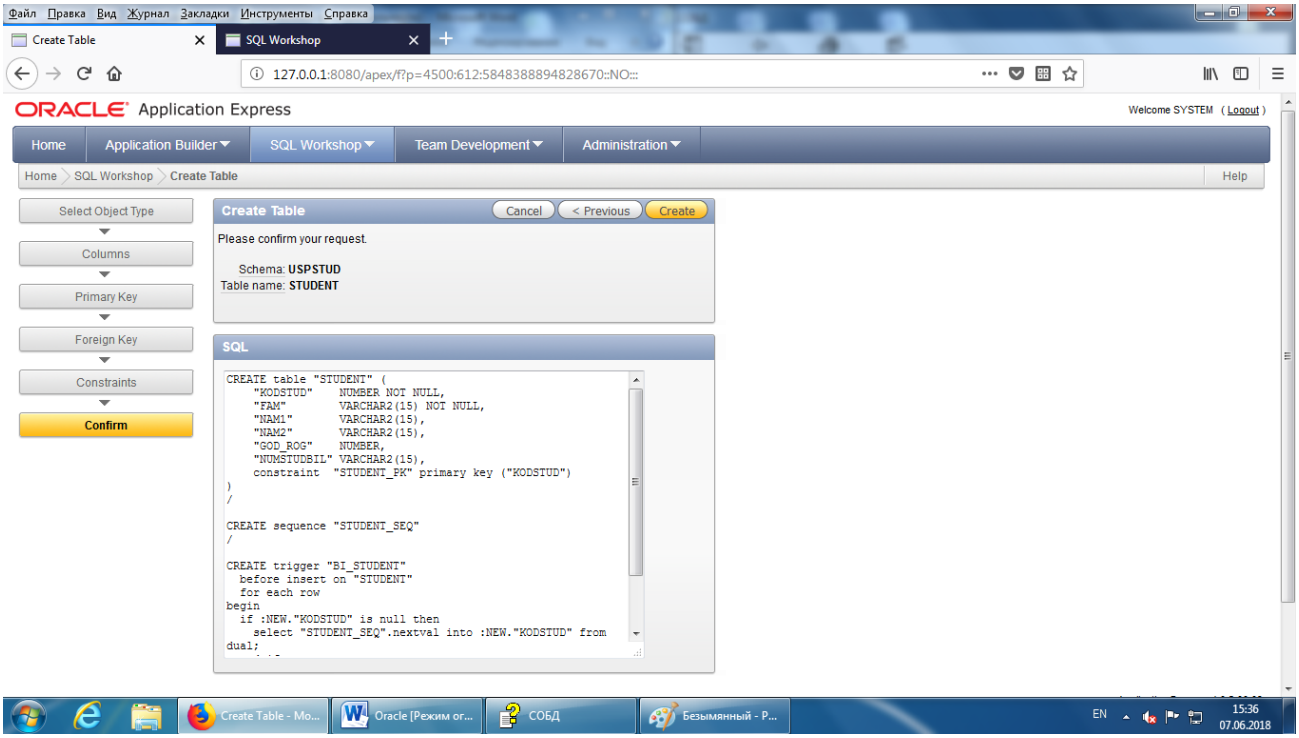


Рисунок 2.22 Перегляд команд SQL, що створюють таблицю STUDENT.

Що би додати записі у таблицю, виберіть закладку «Data», натисніть команду «Insert Row» (Рис.2.23).

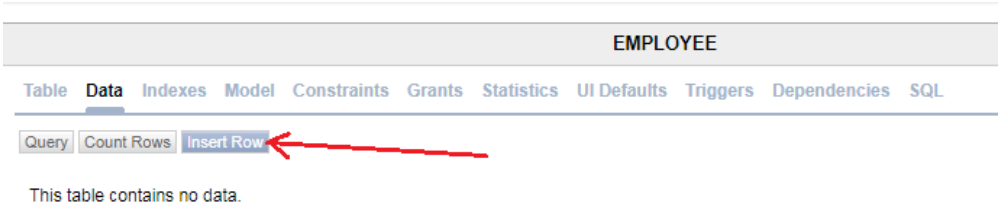


Рисунок 2.23 Додавання записів у таблицю STUDENT.

2.11. 3 Створення уявлень View для вибору даних з одної таблиці

Для пошуку даних що зберігаються у таблицях БД частіше використовують об’єкт уявлення (View).

Розглянемо приклад як створити запит якій сформує список успішності студентів за допомогою утілити Query Builder. Для цього натисніть кнопку «Create» , виберіть команду «View» (Рис. 2.23).

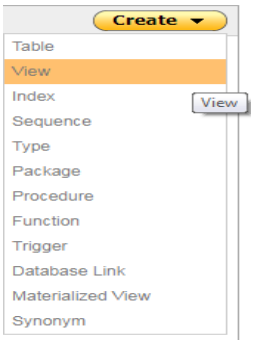


Рисунок 2.23 Меню вибору об’єктів, які потрібно створити

Відкриється вікно ***Create View*** (Рис.2.24). У полі View Name введіть назву уявлення. Натисніть на посилання «Query Builder».

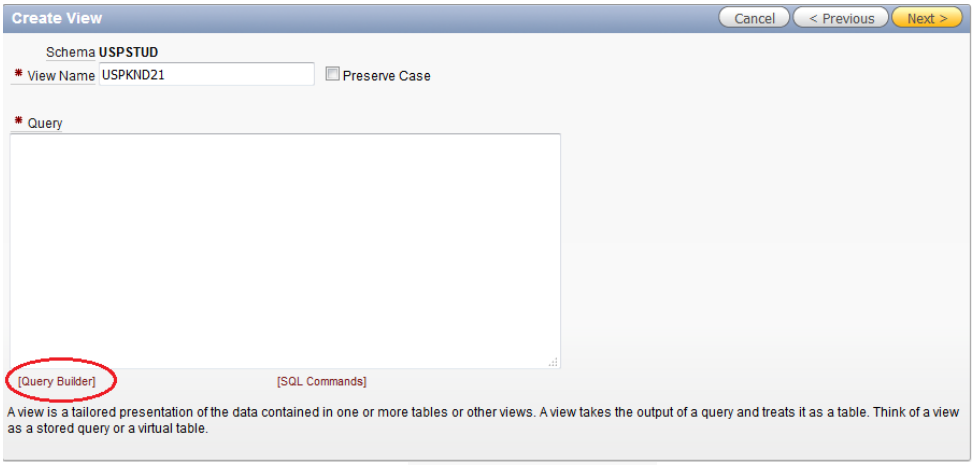


Рисунок 2.24 Вікно ***Create View***

У списку ліворуч виберіть ім’я таблиці, наприклад «USP», праворуч буде відображено структуру таблиці у якій потрібно відмітити потрібні поля та натиснути кнопку «***Run***» (Рис.2.25). Результат виконання запиту буде відображено у нижній частині вікна, якщо натиснуть на вкладку «Results». Перегляд структури команд SQLбуде відображено якщо натиснуть на вкладку «SQL». Для збереження запиту потрібно відкрити вкладку «Save SQL».

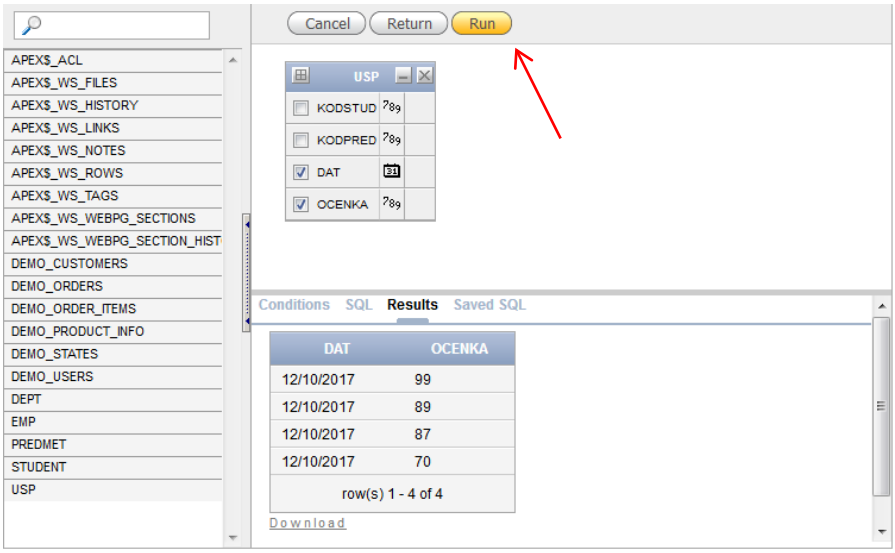


Рисунок 2.25 Вікно для вибору таблиці, полів та перегляду результатів виконання ***View***

2.11.4 Створення уявлень View для вибору даних з декількох таблиць

У разі потреби вибору даних з декількох таблиць, наприклад Predmet, Student, USP, додаймо ці таблиці з списку об'єктів та вибираємо поля. Використовував вкладку «Conditions» можна додати у запит команду сортування, групових операцій (Avg, Count …), GroupBy (Рис.2.26, 2.27).

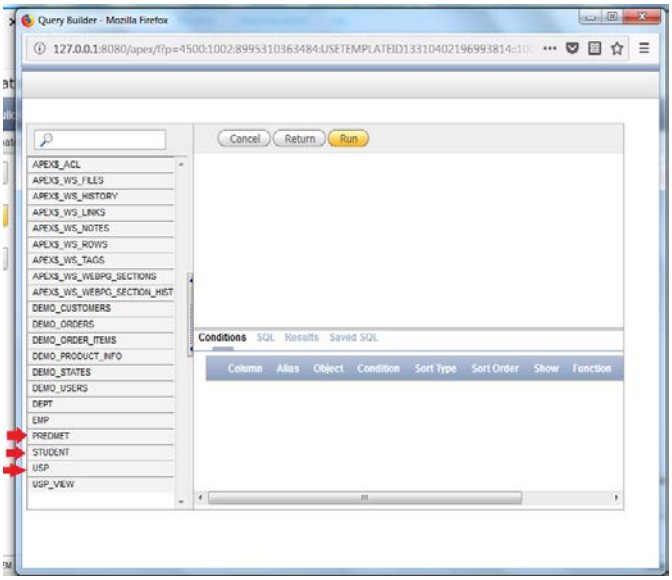


Рисунок 2.26 Вікно для вибору таблиць дляперегляду результатів виконання ***View***

Натиснув кнопку «Return», перейдемо у вікно створення уявлення (***View***), де побачимо структуру запиту (Рис.2.28).

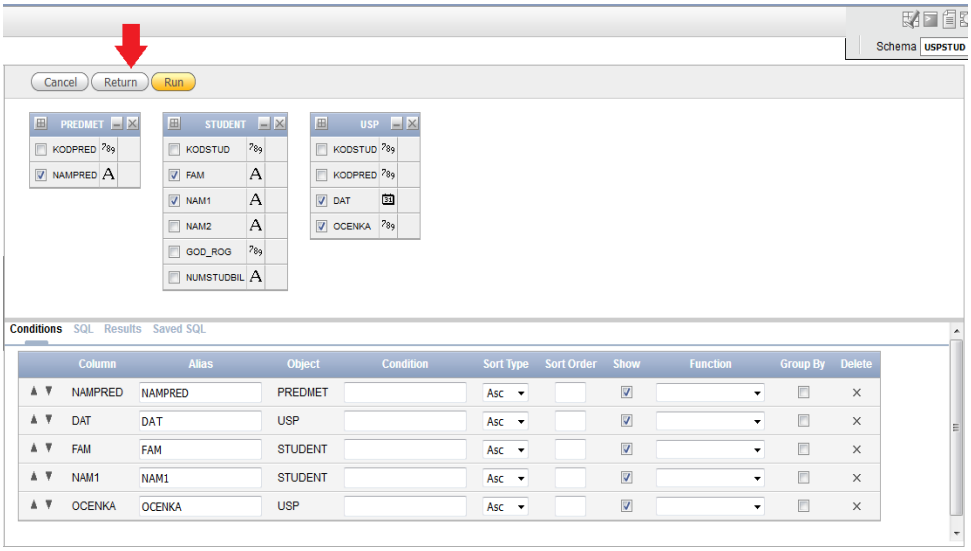


Рисунок 2.27 Вікно для вибору полів та налагодження параметрів запиту команди ***View***

Для збереження уявлення натиснім кнопку «Next» а потім « Create» (Рис. 2.28).

Рисунок 2.28 Вікно для перегляду та збереження запиту команди ***View***

Уявлення, що створено, відображається у списку об’єктів ліворуч, якщо вибрати з меню строчку ***Views*** (Рис.2.29). Вибрав потрібно уявлення праворуч відображається структура таблиці запиту уявлення якщо натиснута кнопка «***View***». Якщо вибрано закладку «Data», буде відображено інформацію з полій таблиць.

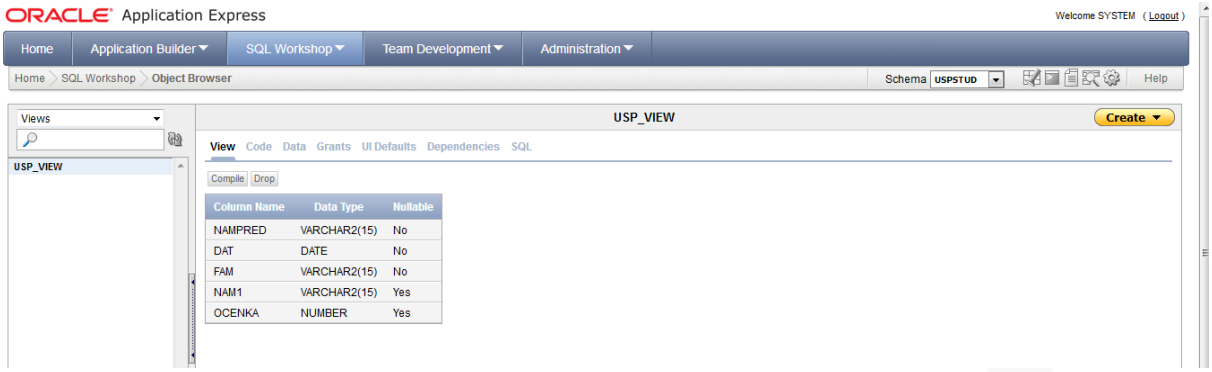


Рисунок 2.29 Вікно для перегляду структури та даних що виконує ***View***

3. Створення призначених для користувача додатків

3.1 Реєстрація додатку, що строюється для перегляду даних таблиць

Для пошуку та виведення потрібної для користувача інформацію з бази даних в Oracle DB 11g є можливість розробляти **звіти** і **форми**, виконані як Web-сторінки.

Побудуємо форми звіту для таблиць, що входять в навчальну базу даних користувача з ідентифікатором **UspStud**. Для цього потрібно відкрити базу даних UspStud. Щоб створити додаток, увійдіть у сервер Oracle DB 11g як користувач **UspStud**CAD21.

Цей користувач володіє декількома таблицями бази даних у схемі **UspStud**CAD21, яка може бути використана при створенні додатків для вигаданого Журналу **UspStud** .

Введіть логін і пароль для входу у сервер: ***system/masterkey*** (Рис.3.1).

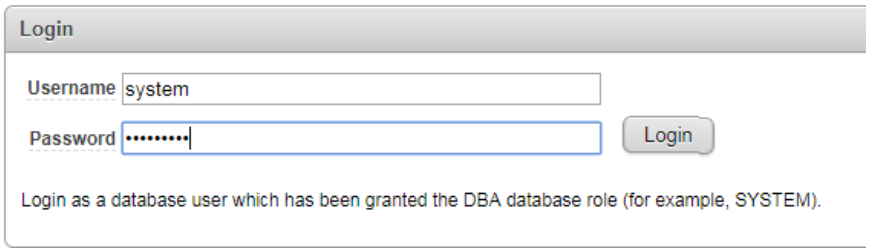


Рисунок 3.1 Вхід у сервер БД Oracle DB 11g

У формі, що відкриється, виберіть закладку **Application Express** за допомогою якою можно створити нову або відкрити БД ,що існує, для редагування або покращення складу форми у якої буде відображатися інформація з таблиць (Рис.3.2).

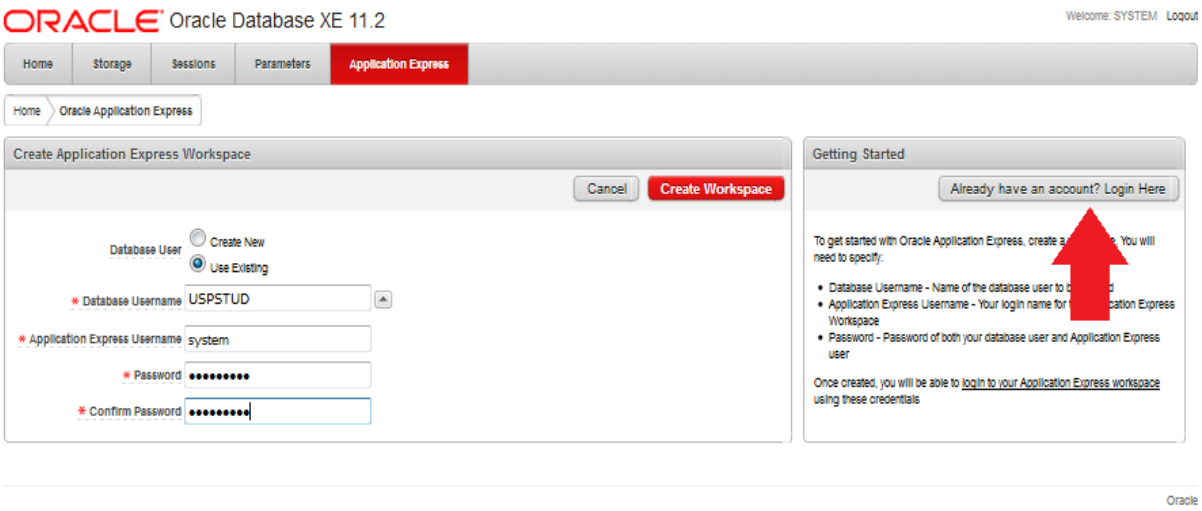


Рисунок 3.2 Вікно для реєстрації додатку серверу БД Oracle DB 11g

Виберіть у розділу Крпистувач БД (**Database User)** радіокнопку Використовуйте існуючу (**Use Existing).**

У полі **Database Username** з списку, що випадає, виберіть базу даних яка була створена раніше **UspStud**CAD21.

Введіть ім'я користувача додатку, що буде створено, **Application Express Username**: **system** та введіть пароль : **masterkey**.

Натисніть праворуч кнопку **Login Here** (Рис.3.2).

3.2 Створення простого додатка для перегляду і редагуванню даних

Розглянемо порядок створення додатку на основі таблиці ***USP***, що є частиною схеми БД **`UspStud`**. Щоб створити додаток для перегляду або редагуванню даних що зберігаються у таблиці USP виконуємо наступні дії.

На головній сторінці бази даних клацніть іконку **Application Builder** (Рис.3.3).

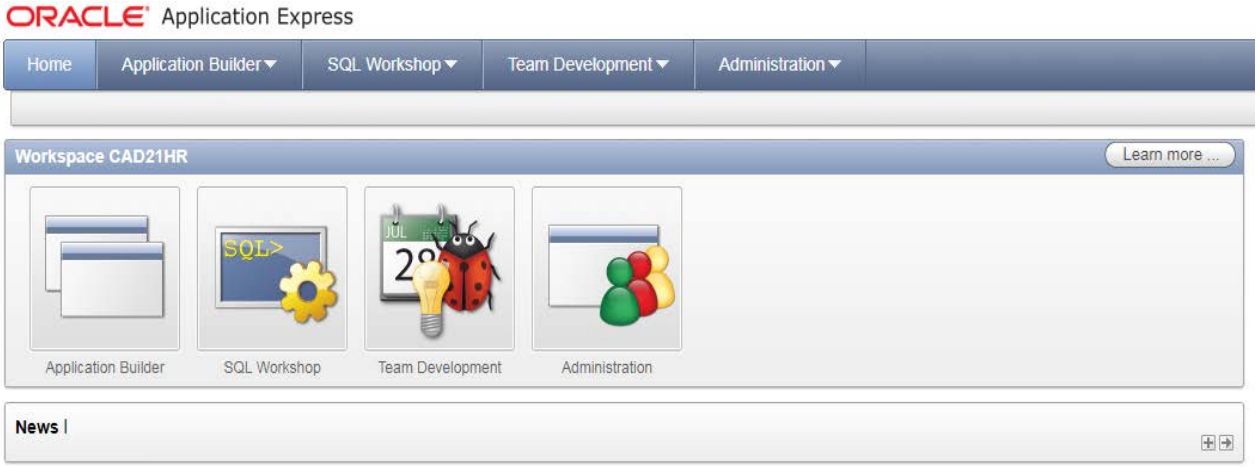


Рисунок 3.3 Вікно для відкриття способу побудови додатку серверу БД Oracle DB 11g

Відкриється закладка форми **All Applications** у якої відображаються усі додатки, що розроблені для бази даних **`UspStud`** (Рис.3.3). У цій формі є три вкладки:

All Application – вікно для перегляду додатків, які створені;

Database Applications – вікно для розробки Windows додатків;

Webssheet Applications - вікно для розробки Web додатків.

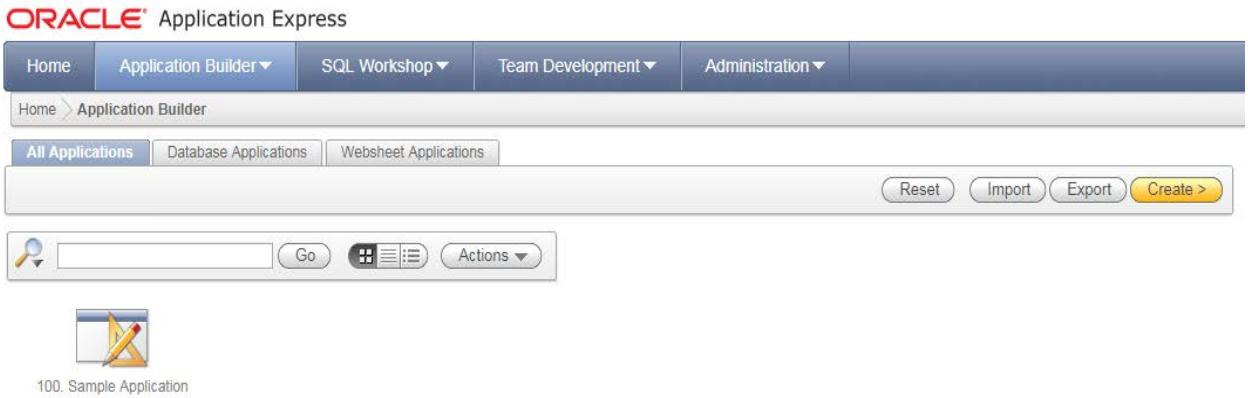


Рисунок 3.3 Вікно для відкриття способу побудови додатку серверу БД Oracle DB 11g

Праворуч форми натисніть кнопку **Create.** Відкриється вікно **Create Application** (Рис.3.4).

Можна побудувати 3 варіанту додатку: форму для редагування таблиць БД (Database), таблицю, яка буде редагуватися за допомогою браузеру (Websheet) та вибрати шаблон (Sample Applications).

На цій сторінці, виберіть метод побудови додатку **Application Type - Database** і натисніть Next.

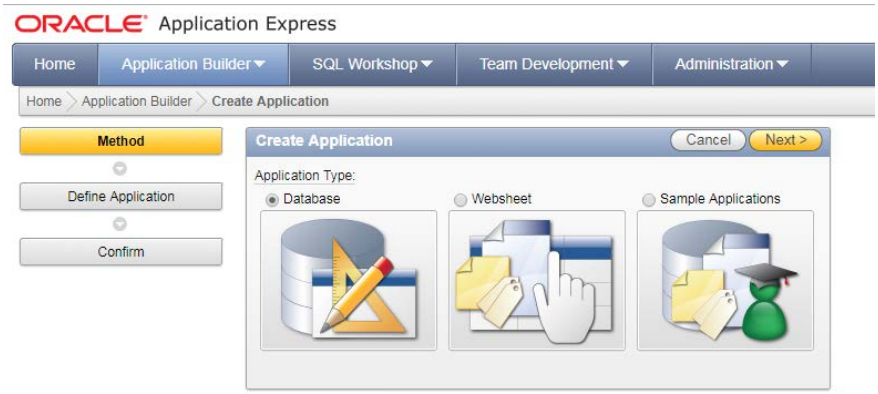


Рисунок 3.4 Вікно **Create Application**

Метод *Для електронної таблиці (***For spreadsheet)** формує додаток у вигляді електронної таблиці, аналогічно таблицям Excel.

Метод *З нуля* **(From Scratch)** формує додаток у вигляді форми як у Delphi або C#. Режим From Scratch дозволяє: створіть програму, визначивши сторінки, вибравши схему автентифікації та вказавши користувальницький інтерфейс. Сторінки можуть ґрунтуватися на таблицях, запитах чи запитах, що розгортаються.

В новому вікні (Рис.3.5)виберіть метод **From Scratch** (З нуля).

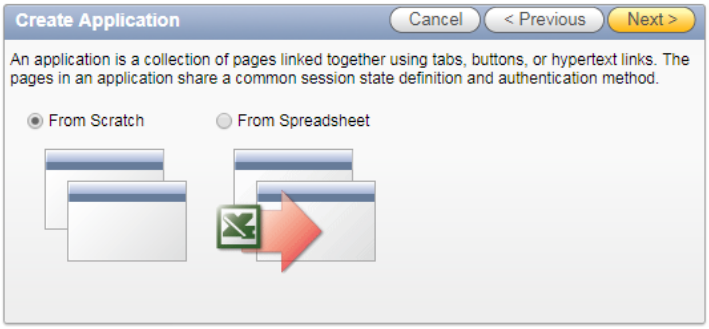


Рисунок 3.4 Вікно вибору методу побудови додатку

Введіть ім'я програми та унікальний ідентифікатор програми. Потім виберіть метод створення програми та схему (Рис.3.5).

У формі **Create Application** введіть наступні параметри. В поле **Name** введіть **UspStud**CAD21, полі **Shema** введіть – **UspStud**CAD21 (ім’я користувача БД). Решта поля залиште без змін. Натисніть **Next.**

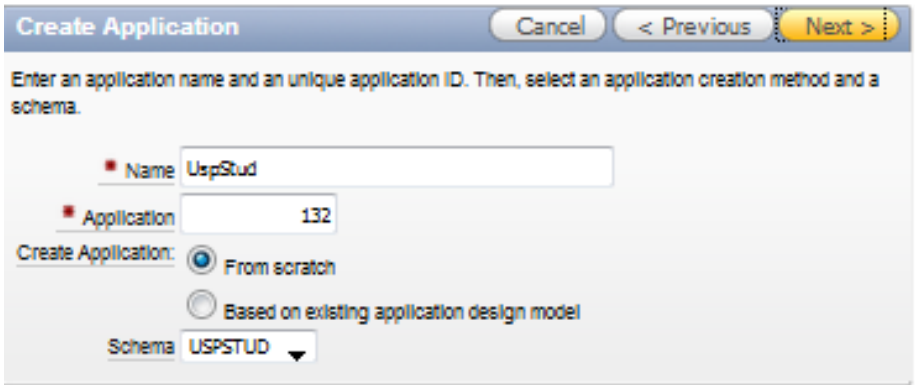


Рисунок 3.5 Вікно реєстрації додатку

У наступному вікні потрібно вибрати тип сторінки, яку хочете створити**:**

Blank створює сторінку без вбудованої функціональності. **Звіт** створює сторінку, яка містить форматований результат запиту SQL. Ви можете вибрати для створення звіту на основі вибраної вами таблиці або на основі індивідуального SQL SELECT або функції PL / SQL, що повертає висловлене висновок SQL SELECT.

Форма створює форму для оновлення одного рядка в таблиці бази даних.

Звіт та форма створює звіт про дві сторінки та комбінацію форм. На першій сторінці користувачі вибирають рядок для оновлення. На другій сторінці користувачі можуть оновлювати обрану таблицю чи перегляд. Таблична форма створює форму для виконання оновлення, вставки та видалення операцій на декількох рядках у таблиці бази даних.

Master Detail створює звіт про дві сторінки та комбінацію форм. На першій сторінці користувачі вибирають рядок для оновлення. На другій сторінці користувачі можуть оновлювати вибрану таблицю або представлення та пов'язані з ним деталі. Пов'язана таблиця або представлення деталей повинна мати відношення зовнішньої ключа до обраної основної таблиці або перегляду.

Діаграма створює сторінку з флеш-схемою, яка представляє результат запиту SQL.

В опції **Select Page Type** виберіть **Report and Form**. В поле відображається тип сторінки, яку ви додаєте (Рис.3.6). В поле Ім’я Таблиці (**Table Name**) виберіть таблиці **STUDENT**. В поле Реалізація (Implementation) виберіть **Interactive**. Натисніть кнопку **Add Page.**

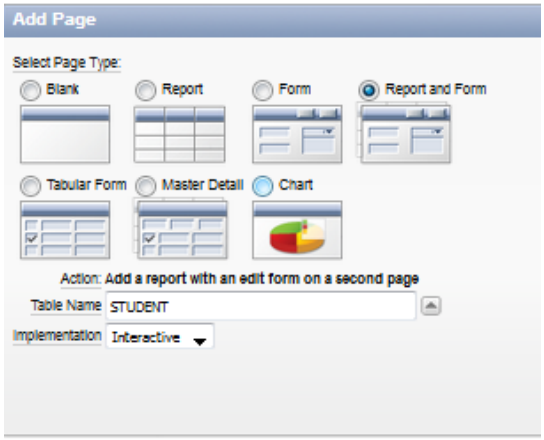


Рисунок 3.6 Вікно додавання елементів додатку

Дві нові строчці відображаються вгорі сторінки у розділі **Створення додатка.** Натиснуть кнопку **Next** (Рис.3.7).

Рисунок 3.7 Вікно перегляду складу додатку

Опцію Tabs залиште без змін (**One Level of Tabs**) і натисніть **Next** (Рис.3.8).

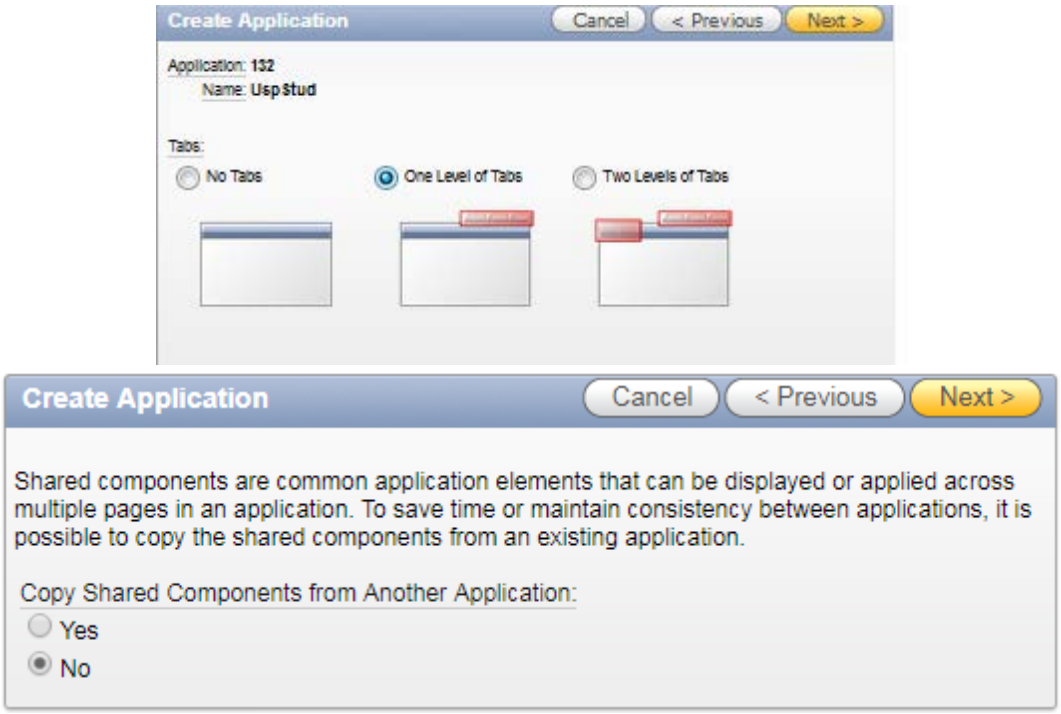


Рисунок 3.8 Вікно вибору рівня додатку

Спільні компоненти - це звичайні елементи програми, які можуть відображатися або застосовуватися на кількох сторінках у програмі. Щоб заощадити час або підтримувати узгодженість між додатками, можна скопіювати спільні компоненти з існуючої програми.

Опцію **Shared Components** залиште без змін (No)і натисніть **Next**. Ця опція дозволить вам імпортувати загальні компоненти з інших додатків. Загальні компоненти - це стандартні елементи, які можуть бути відображені або застосовані на будь-якій сторінці програми.

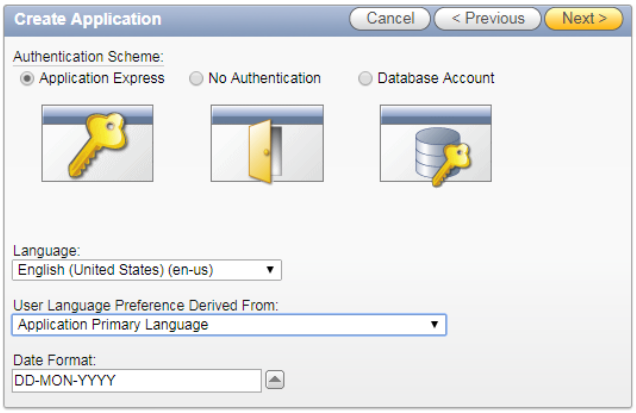


Рисунок 3.9 Вікно вибору мови додатку

Параметри полів **Authentication Scheme, Language і User Language Preference Derived From** залиште без змін і натисніть **Next.** У опціях User Interface виберіть Theme 2 і натисніть **Next** (Рис.3.9). У наступному вікні можна вибрати варіант оформлення інтерфейсу **Select a theme** (Рис.3.10). Теми це набори шаблонів, які можна використовувати для завдання розташування елементів і визначення зовнішнього вигляду всього програми.

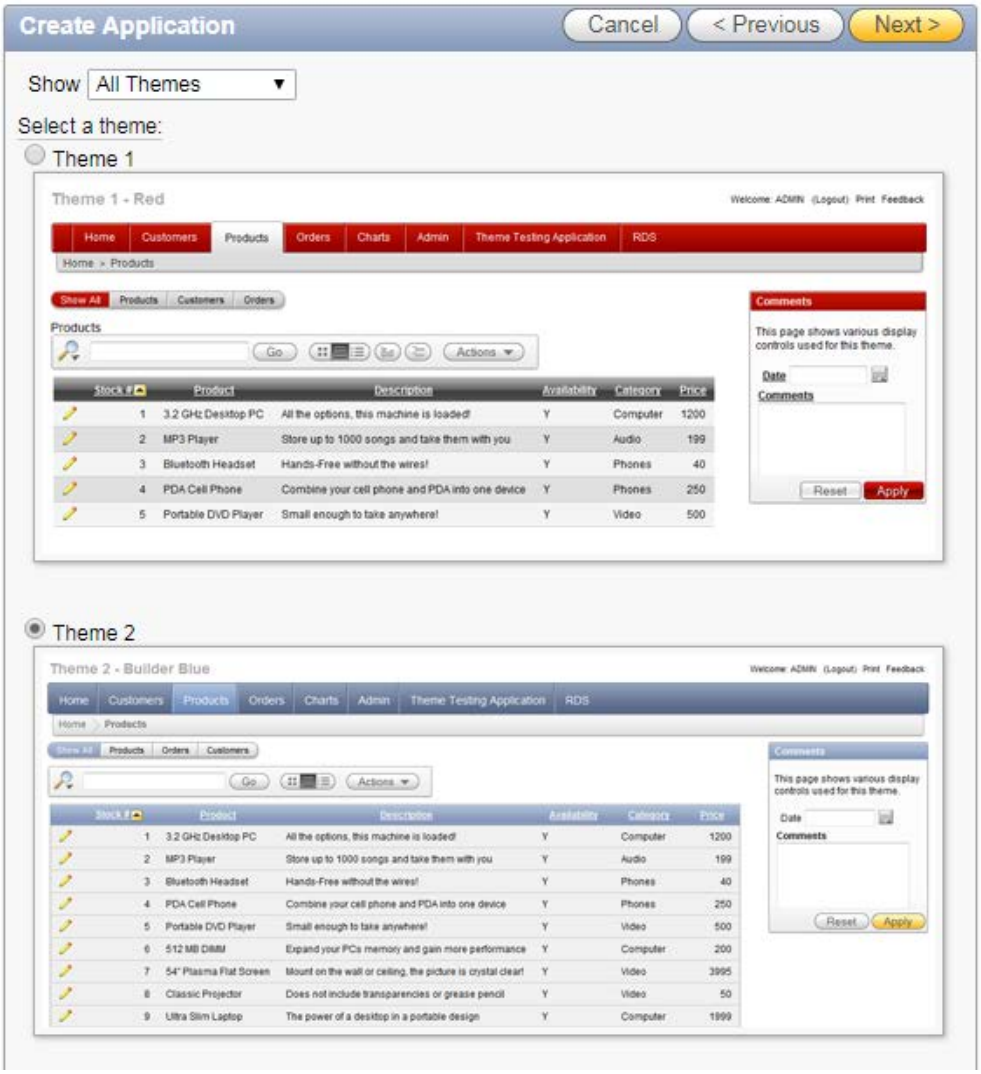


Рисунок 3.10 Вікно вибору інтерфейсу додатку

Підтвердить зроблені зміни. Щоб повернутися на попередню сторінку майстра, натисніть **Previous**. Щоб прийняти зміни, натисніть **Create** (Рис.3.11).

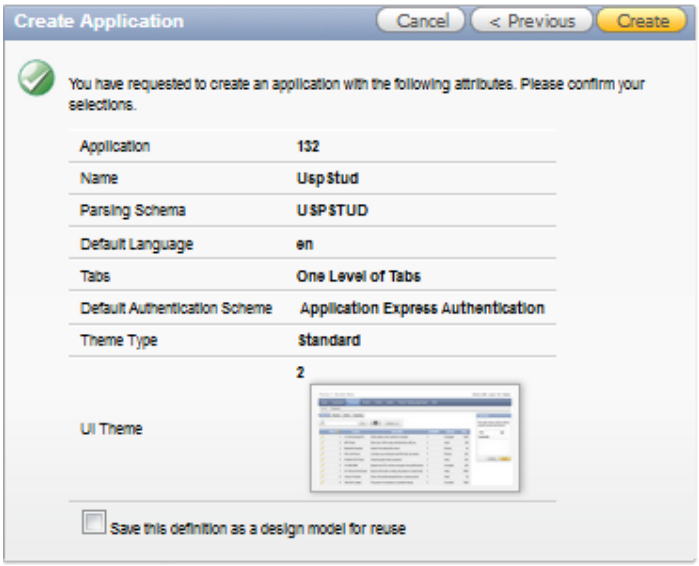


Рисунок 3.11 Вікно для перегляду параметрів додатку

Після того як ви натиснете **Create**, у верхній частині сторінки з'явиться наступне повідомлення: **Application created successfully.**

Спільні компоненти включають аутентифікацію, авторизацію, шаблони користувальницького інтерфейсу та вкладки. У вікні з'явяться три піктограми за допомогою яких можна відкрити сторінки (Рис.3.12).

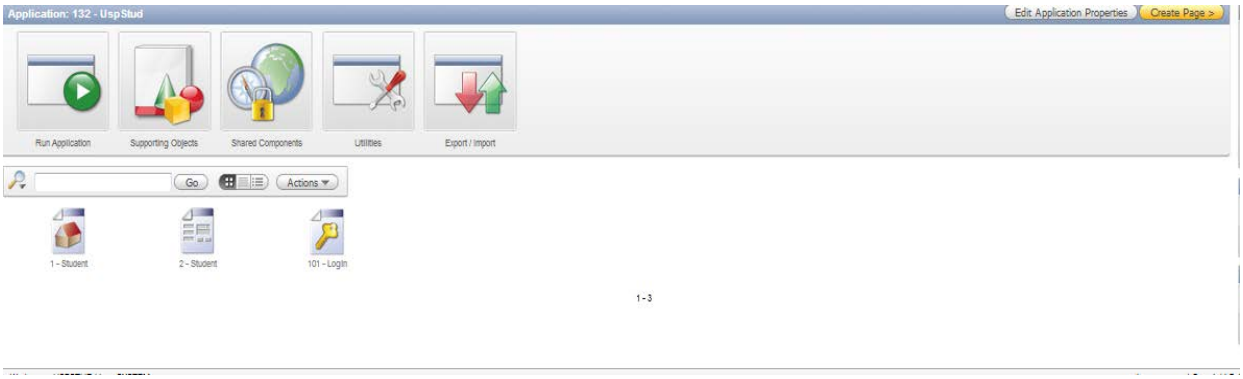


Рисунок 3.12 Вікно для перегляду додатку UspStud

Якщо натиснуть піктограму **«Login»** – відкриється вікно авторизації у якому потрібно ввести ім'я користувача, якій розробляв додаток та пароль (Рис.3.13).

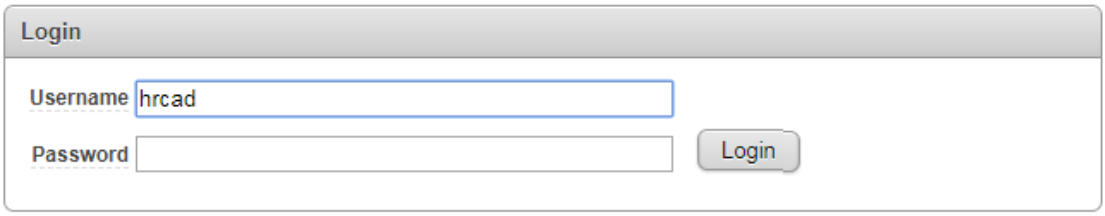


Рисунок 3.13 Вікно авторизації

Якщо натиснуть піктограму **«1-Student»** – відкриється вікно **Report** у вигляді таблиці для перегляду записів, що зберігаються, та їх редагування (Рис.3.14).

Employee Id	First Name	Last Name	Email	Phone Number	Hire Date	Job Id	Salary	Commission Pct	Manager Id	Department Id
100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	-	-	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	-	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	-	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	-	102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	-	103	60
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	-	103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	-	103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	-	103	60
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	-	101	100
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	-	108	100
110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	-	108	100
111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	-	108	100
112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCOUNT	7800	-	108	100
113	Luis	Popp	LPOPP	515.124.4567	07-DEC-07	FI_ACCOUNT	6900	-	108	100
114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-02	PU_MAN	11000	-	100	30

1 - 15

Рисунок 3.14 Вікно **Report**

Якщо натиснуть піктограму «2-Student» – відкриється вікно **Form** у вигляді колонці компонентів системи об’єктно- орієнтованого програмування Label та DBEdit для додавання нової записи у таблицю (Рис.3.15).

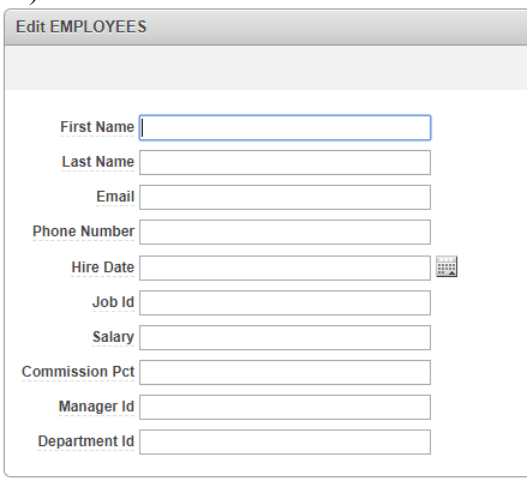


Рисунок 3.15 Вікно **Form**

Програми додатку складаються з сторінок. Для кожної сторінки ви можете декларативно визначити спосіб оформлення та обробки сторінки. Керування, які не є специфічними для сторінки, називаються загальними компонентами.

3.3 Запуск нової програми

Розглянемо порядок запуску створеного додатку на приладі БД HR. Щоб запустити програму додатку клацніть іконку **Run Application** 

На сторінці авторизації, введіть ім’я БД HR як в поле **User Name**, так і в поле Password (Рис.3.16).

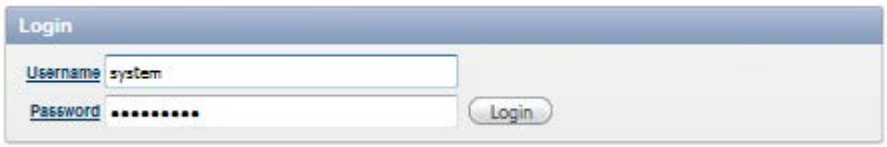


Рисунок 3.16 Введення даних у вікно авторизації

З'явиться додаток, що показує сторінку **Report** для таблиці EMPLOYEES. Можна додати або редагувати записи у таблиці потім потрібно натисніть кнопку **Create** і нова запис буде збережена у БД (Рис.3.17).

Рисунок 3.17 Перегляд даних у вікні **Report**

Для редагування запису потрібно натиснуть праворуч піктограму **Edit**, яка відкриває сторінку **Form** де можливо оновити або ввести нові дані (Рис.3.18).

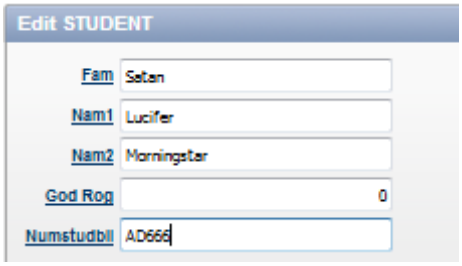


Рисунок 3.18 Редагування даних у вікні **Form**

Для збереження дані, що були відредаговані, натисніть праворуч форми кнопку **Create.**

3.4 Розробка фільтра до таблиці

Для пошуку даних у додатку, використовуйте інструментальне меню розробника **Actions** (Рис.3.19).

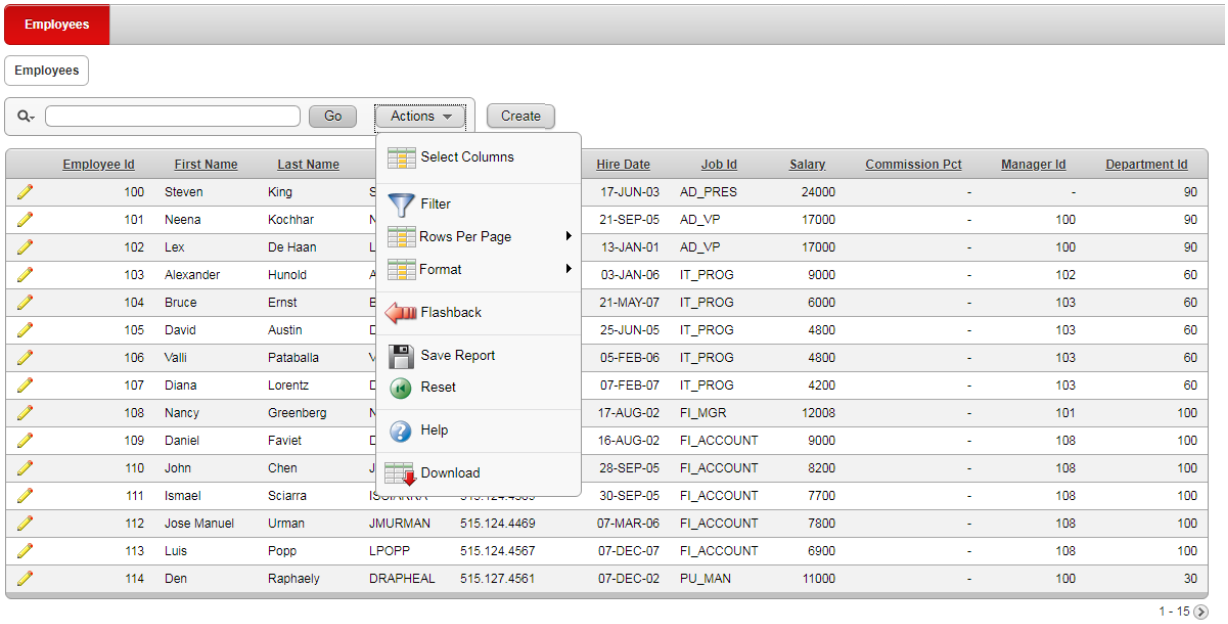


Рисунок 3.19 Вікно для створення фільтрів даних

Натисніть піктограму **Filter,** відкриється шаблон для створення **SQL-**команди за допомогою якої можна виконати пошук потрібної інформації у таблиці. Для цього у колонці Column натисніть групову кнопку та з меню яке складається з назв полів таблиці виберіть потрібне поле.

У колонці **Operator** виберіть арифметичній, логічній або символьний оператор умові пошуку**.** Якщо в умові потрібно вказати символьній тип даних умови, виберіть оператор **Like** (Рис.3.20).

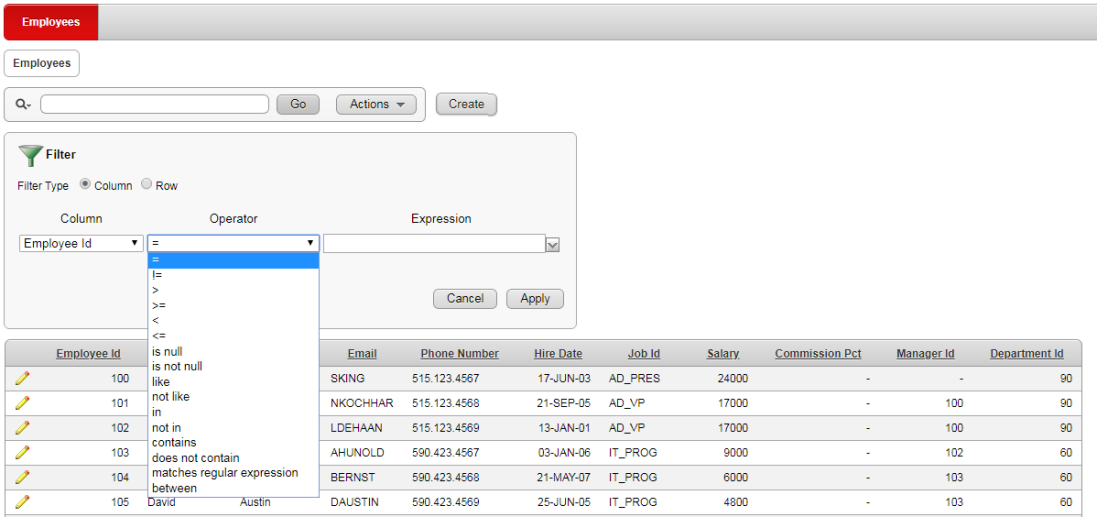


Рисунок 3.20 Вікно для вибору оператору умови пошуку даних фільтру

У колонці **Expression** виберіть вираз для вибраної колонці таблиці (Рис. 3.21).

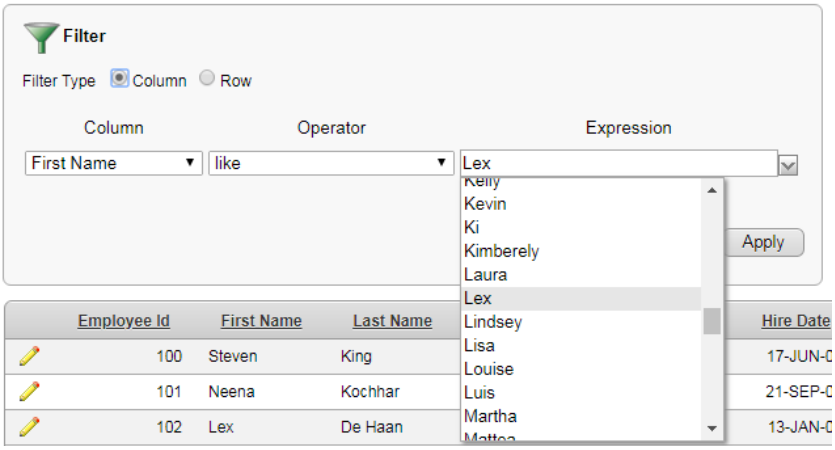


Рисунок 3.21 Вікно для вибору значення виразу умови пошуку даних фільтру

Натисніть **Apply**. У разі успішного виконання, з’явиться таблиця з даними, що відповідають умовам фільтру. На рис.3.22 наведено приклад шаблону та результату виконання фільтру.

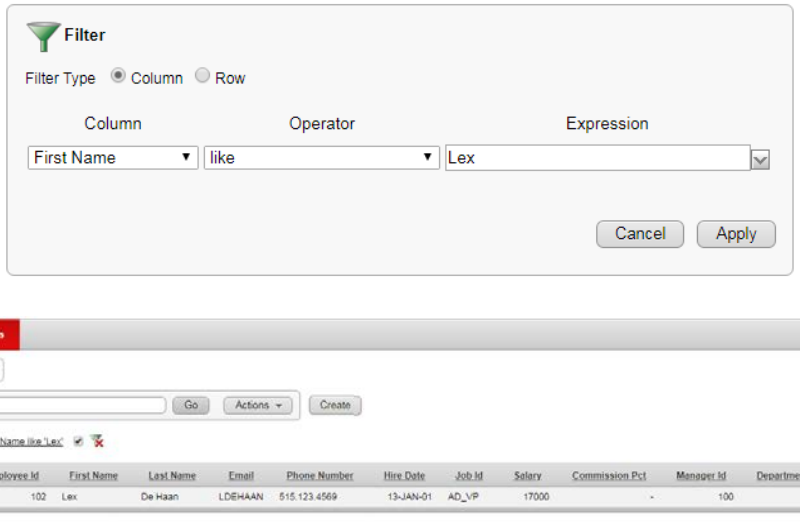


Рисунок 3.22 Приклад шаблону та результату виконання фільтру

3.5 Створення запиту

Фільтр дозволяє виконати пошук тільки з відкритого додатку і вразі якщо додаток закрито то він не виконується.

Запит дозволяє зберегти шаблон для пошуку та виконати його багато разів, при цьому непотрібно відкривати додаток.

Наприклад якщо потрібно у запиті зберегти тільки поля *FirstName*, *LastName*, *PhoneNumber* відкрийте меню **Actions,** виберіть строчку **Select Column,** додайте у вікно що ліворуч потрібні поля та натисніть **Apply.** У результаті буде створено запит якій складається з полів що були вибрані (Рис.3.23)**.**

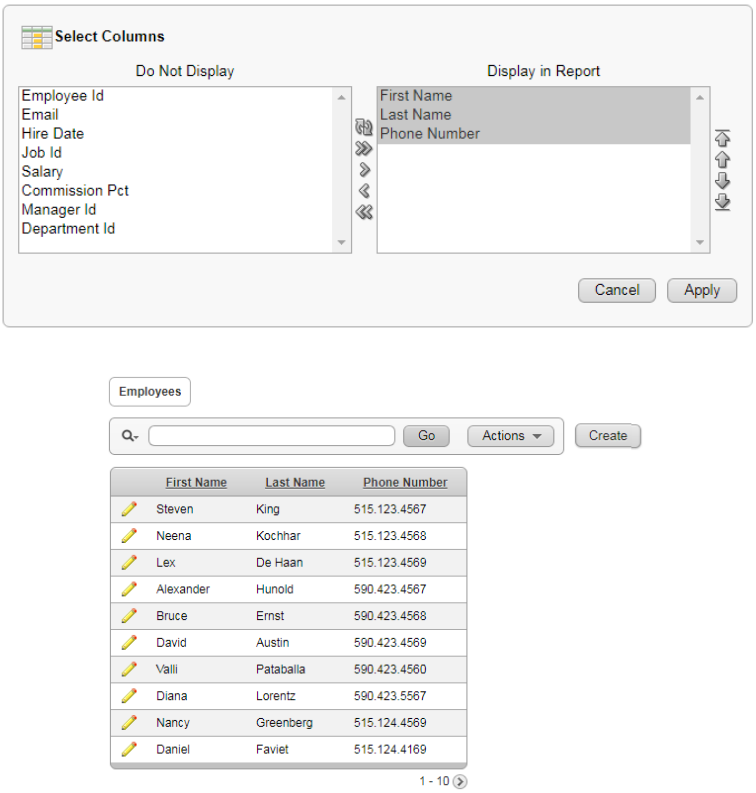


Рисунок 3.23 Створення запиту за допомогою команди **Select Column**

У строчки **Rom Per Page** меню **Actions** можна задати кількість строк таблиці що будуть відображені у результаті виконання запиту.

Повернуться до попереднього виду запиту потрібно вибрати строчку **Flashback** .Запит на **Flashback** дозволяє переглядати дані, як вони існували в попередній момент часу (Рис.3.24).

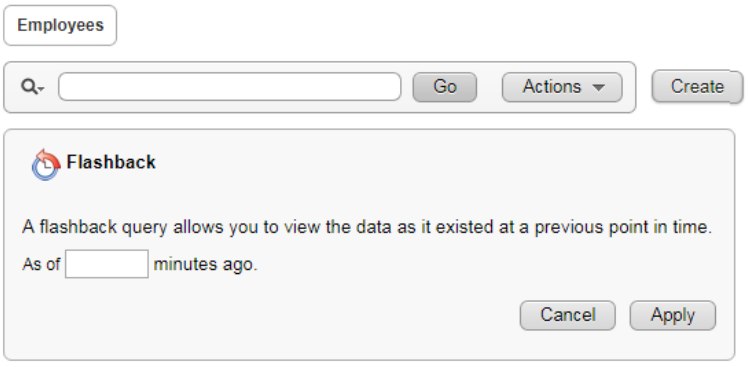


Рисунок 3.24 Перегляд складу запиту в попередній момент часу

У строчки **Raset** меню **Actions** можна відновити звіт за замовчуванням.

Для обробки та візуалізації даних що зберігаються у додатку, використовуйте інструментальне меню **Format** (Рис.3.25).

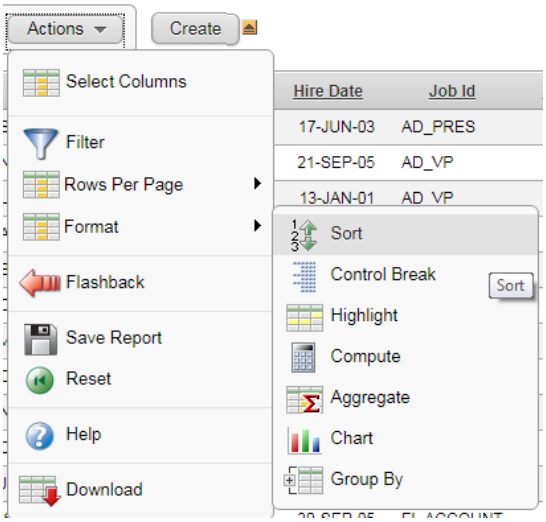


Рисунок 3.25 Редагування запиту за допомогою команди **Format**

Наприклад щоб відобразити дані у вигляді діаграм виберіть строчку **Chart.** У цьому вікні потрібно вказати тип діаграми та поля, інформація з яких буде відображатися (Рис.3.26).

Рисунок 3.26 Редагування діаграми запиту за допомогою команди **Chart**

Для збереження запиту потрібно вибрати команду **Save report** у меню **Actions** та надати ім’я запиту, що був створено (Рис.3.27).

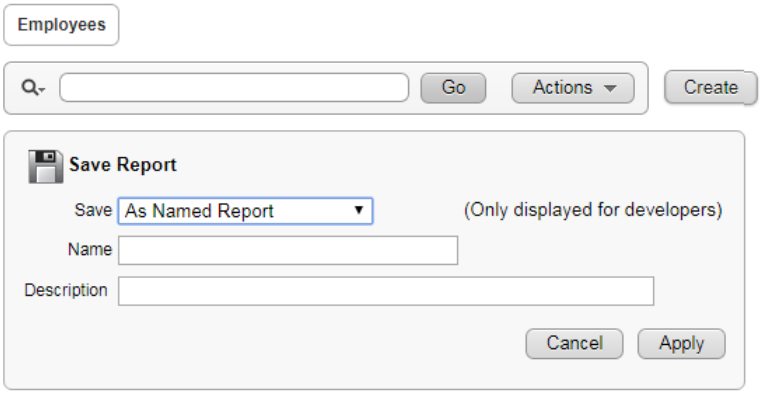


Рисунок 3.27 Збереження запиту за допомогою команди **Save report**

Інструментальне меню розробника дає можливість оперативно відредагувати поточну сторінку, створити нову сторінку, елемент управління або компонент, подивитися стан сесії, а також включити / відключити режим налагодження або посилання редагування.

Для видалення додатка, відкрийте його та праворуч виберіть команду ***Delete this Application*** (Рис.3.28).

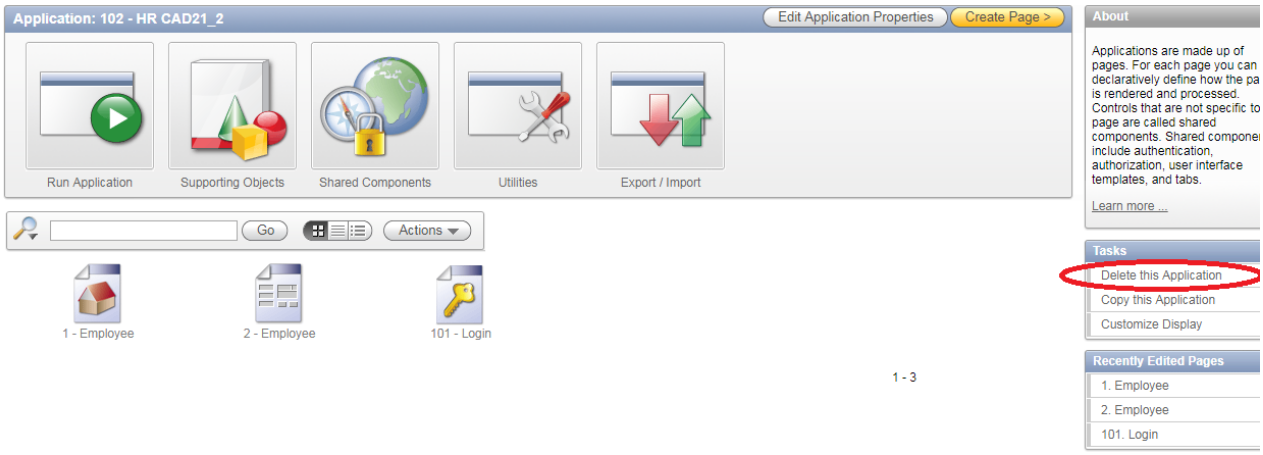


Рисунок 3.28 Видалення додатка

Для виходу з програми і повернення в Application Builder, клацніть Edit Page 1 у меню розробника. Для повернення на домашню сторінку бази даних виберіть пункт **Home.**

4. Створення WEB-додатку

Веб-додатки дозволяють користувачам створювати програми, орієнтовані на дані, без будь-яких знань про програмування SQL. Веб-додатки спрощені та підтримують сторінки, сітки даних і звіти. Ці програми легко створювати та підтримувати внески масивів даних.

Для створення WEB – сторінки у вікні Application builder виберіть **Websheet Applications** (Рис.4.1).

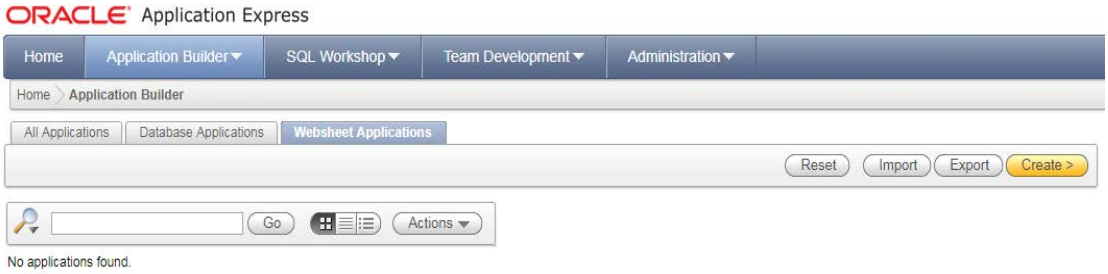


Рисунок 4.1 Вікно Websheet Applications

У вікні Create Application виберіть піктограму Websheet (веб-сторінка) та натисніть кнопку Next (рис.4.2).

Рисунок 4.2 Вибір форми додатку Websheet

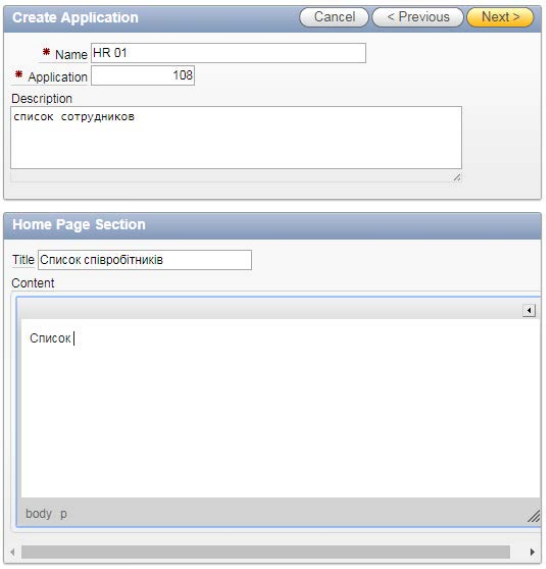


Рисунок 4.3 Вікно Application builder

У полі **Title** введіть назву сторінки. У полі **Content** - опис призначення сторінки.

Натисніть кнопку **Next,** з'явіться повідомлення з параметрами сторінки (рис.4.4).

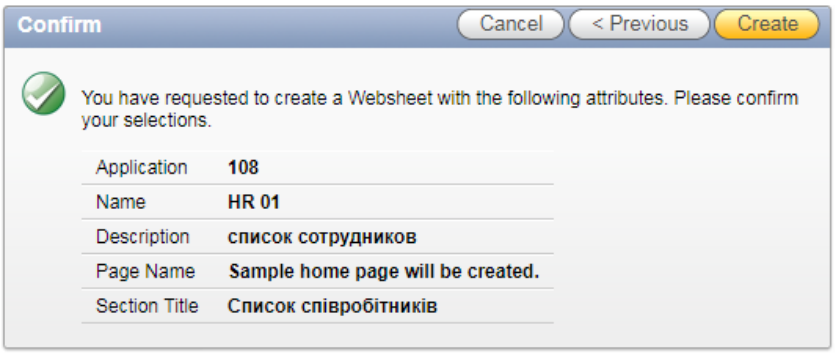


Рисунок 4.4 Повідомлення з параметрами сторінки

Після перегляду опису сторінки натисніть **Create.** Виберіть піктограму **Run** (Рис.4.5).

Рисунок 4.5 Запуск сторінки

Введіть повноваження веб-сайтів: Username –HR; Password -12345 (Рис.4.6).

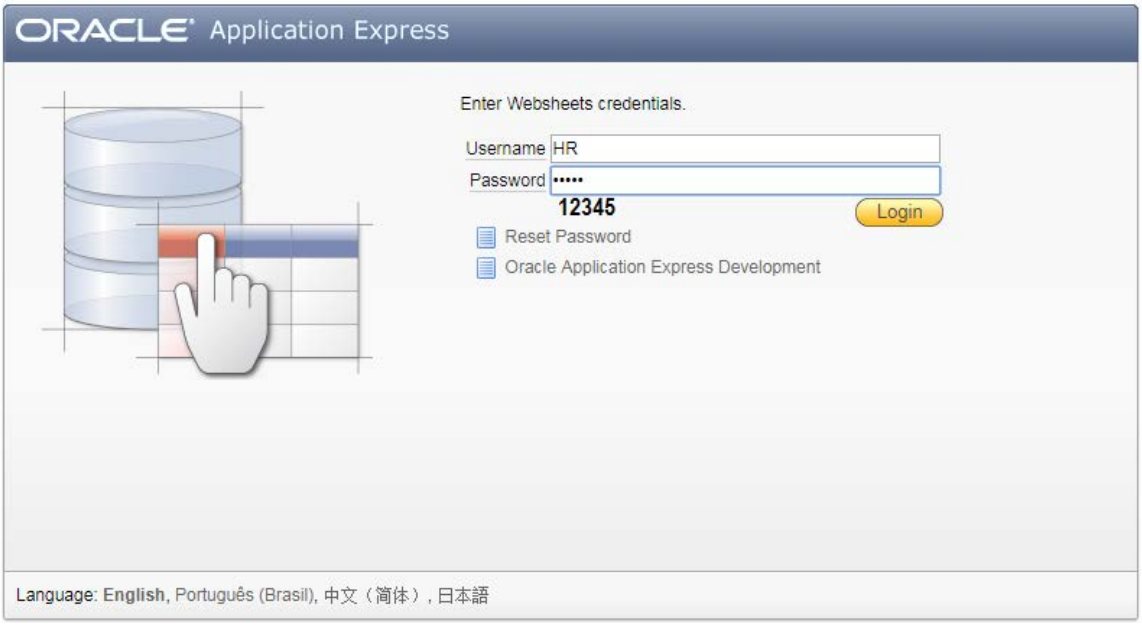


Рисунок 4.6 Авторизація сторінки

Буде створена сторінка на яку можливо додати розділи, що наведено у списку праворуч (Рис.4.7).

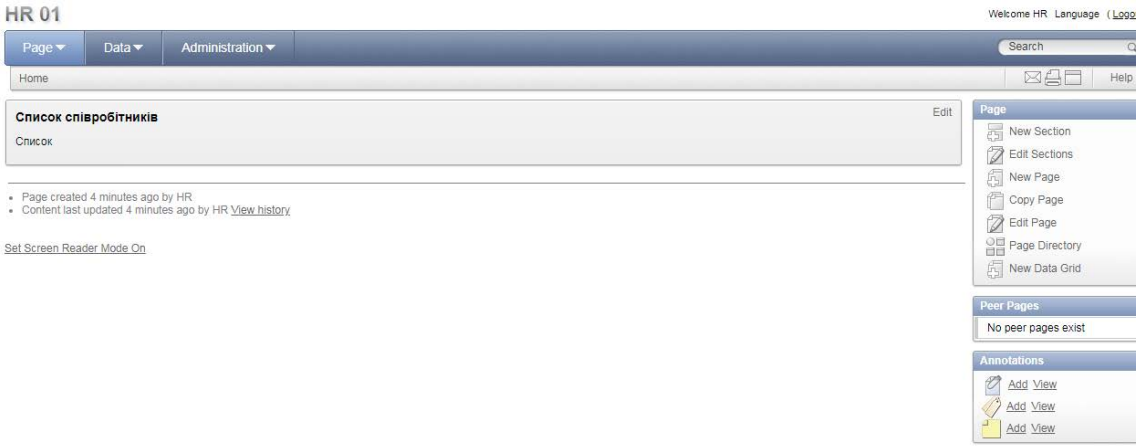


Рисунок 4.7 Вікно сторінки, що відкрито.

Для створення таблиці БД додаємо **Data Grid,** для цього натисним закладку (Рис.4.8). Сітки даних декларованим чином редагують та підтримують збірки даних. Сітка даних не базується на існуючих таблицях, дані для мереж даних підтримуються Oracle Application Express.

Рисунок 4.8 Закладка для створення Data Grid

На наступному етапі визначим вид оформлення сторінки, наприклад **Form Scratch** (Рис.4.9).

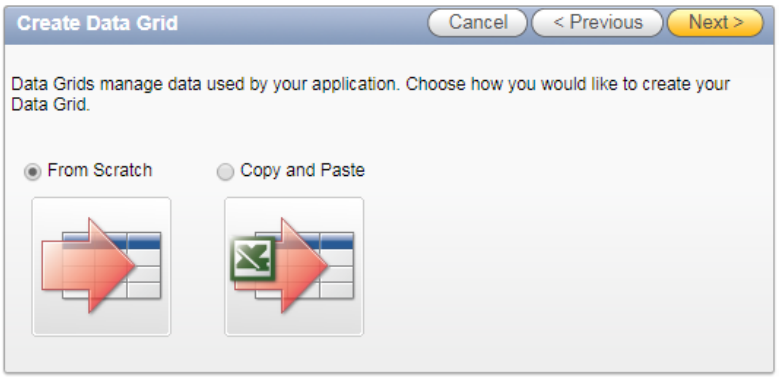


Рисунок 4.9 Вікно для визначення виду сторінки

Призначимо ім’я, аліас, назву колонок, тип даних та натиснім кнопку **Create** (Рис.4.10)**.**

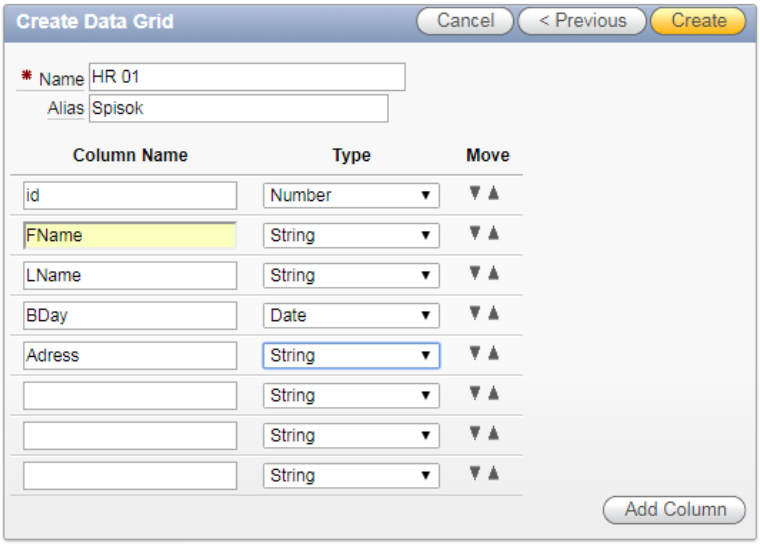


Рисунок 4.10 Вікно для визначення атрибутів

Буде створено закладку Data у якої відображається створеній **Data Grid** з ім'ям **HR01** (Рис.4.11)**.**

Рисунок 4.11 Вікно у якому відображається створеній Data Grid

Для додавання даних потрібно натиснуть на піктограму **HR01** (Рис.4.12). Відкриється вікно, у якому натисніть кнопку Add Row (Додати строку).

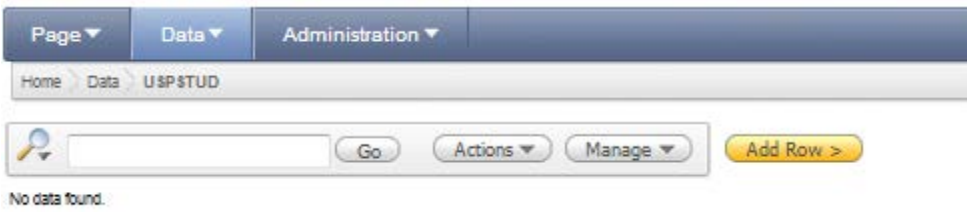


Рисунок 4.12 Вікно для відкриття додавання записів

У наступному вікні (Рис.4.13) можна додати дані у шаблон таблиці та зберегти, натисніть **Save.**

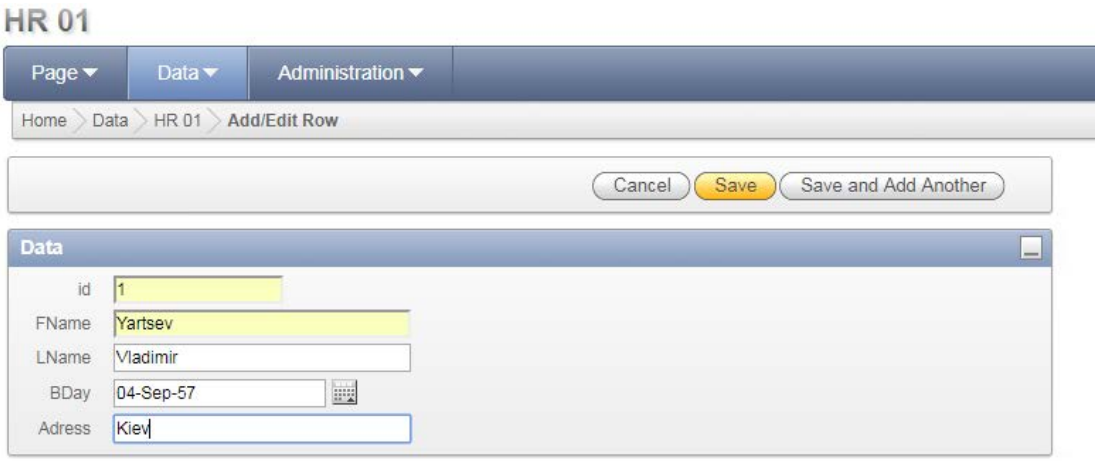


Рисунок 4.13 Введення даних у таблицю БД

Перевірите запис що була додано, для чого подвійно клацніть по піктограмі **HR01** (Рис.4.14).

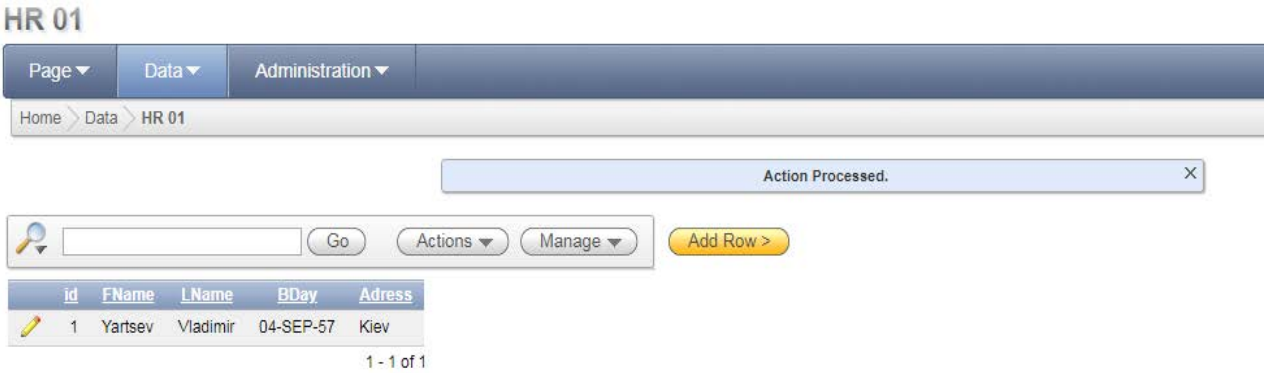


Рисунок 4.14 Перевірка запису

Перевірте склад записів у таблицях БД **HR.** Натисніть закладку **Home** (Рис.4.15).

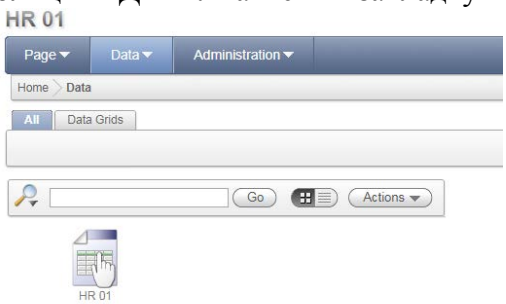


Рисунок 4.15 Вікно реєстрації сторінок веб-додатку

5. Утиліта SQL*Plus

Підключатися і працювати з базами даних Oracle можна багатьма способами. Однак найчастіше для цього застосовується пропонований в Oracle інтерфейс і набір команд SQL*Plus. Інтерфейс SQL*Plus, по суті, відкриває вікно в базу даних Oracle і тому дуже широко використається розроблювачами Oracle для створення програмних одиниць SQL і PL/SQL. Для адміністраторів баз даних Oracle цей інтерфейс теж є дуже коштовним інструментом по наступних причинах.

SQL*Plus дозволяє виконувати запити мовою SQL і блоки коду мовою PL/SQL (який являє собою пропоновану в Oracle розширену процедурну версію мови SQL) і одержувати результати. Утиліта дозволяє виконувати команди, пов'язані з адмініструванням баз даних, і автоматизувати їх. Також SQL*Plus дозволяє запускати й завершувати роботу бази даних. Він надає зручний спосіб для створення звітів по адмініструванню баз даних.

Інтерфейс SQL*Plus являє собою утиліту, що найчастіше застосовується для підключення й роботи з базами даних Oracle. Він поставляється в складі як серверного програмного забезпечення Oracle Database 11*g*, так і клієнтського програмного забезпечення Oracle Client, а також нового програмного забезпечення Oracle Instant Client.

Послу установки SQL*Plus на сервері або клієнтській машині процес підключення

до сервера або клієнта й запуску сеансу SQL*Plus виглядає дуже просто. Через те, що каждый сеанс SQL*Plus має на увазі установку з'єднання з базою даних усе, що потрібно для запуску SQL*Plus і підключення до бази даних - це дійсне ім'я користувача й пароль.

5.1 Настроювання середовища

Перед викликом SQL*Plus буде потрібно спочатку правильно настроїти середовище Oracle.

Зокрема, необхідно встановити значення для таких змінні середовища, як ORACLE_SID, ORACLE_HOME і LD_LIBRARY_PATH. Крім цього іноді потрібно встановити значення й для таких змінні середовища, як NLS_LANG і ORA_NLS11.

У випадку не установки правильних значень для необхідних змінні середовища буде виникати помилка. Наприклад, не установка належного значення для змінний ORACLE_HOME перед запуском SQL*Plus буде приводити до появи наступної помилки:

$ sqlplus
*Error 6 initializing SQL*Plus*
Message file sp1<lang>.msb not found
SP2-0750: You may need to set ORACLE_HOME to your Oracle software directory

*Помилка 6 при ініціалізації SQL*Plus*
Не вдалося виявити файл sp1<lang>.msb
SP2-0750: Можливо, потрібно вказати в ORACLE_HOME каталог, у якому установлено програмне забезпечення Oracle

У випадку одержання показної вище помилки досить установити значення для змінного середовища ORACLE_HOME:

$ export ORACLE_HOME= /u01/app/oracle/product/11.1.0/db_1

5.2 Програмне забезпечення SQL*Plus Instant Client

Для використання SQL*Plus инсталлировать повністю все серверне програмне забезпечення Oracle Database зовсім не обов'язково. Якщо потрібно взаємодіяти через інтерфейс SQL*Plus з базою даних, що перебуває на іншому сервері, цілком вистачить і програмного забезпечення SQL*Plus Instant Client. За допомогою цього програмного

забезпечення до будь-якої бази даних Oracle, що функціонує під керуванням будь-якої операційної системи, можна підключатися вилученим образом за рахунок вказівки її імені й застосування ідентифікатора мережного підключення Oracle.

Єдиною вимогою для підключення до вилученої бази даних подібним чином є вказівка імені вилученої бази даних у файлі tnsnames.ora. Саме тому для SQL*Plus Instant Client потрібно задавати змінну середовища ORACLE_HOME.

Існує також метод, що не вимагає застосування на клієнтському сервері файлу tnsnames.ora. Називається він методом *простого підключення* (easy connect). Нижче наведений приклад, що показує, як за допомогою ідентифікатора простого підключення встановити від імені користувача OE підключення до бази даних testdb, розташованої на сервері myserver:

$ sqlplus oe/oe@//myserver.mydomain:1521/testdb

У цьому прикладі 1521 - це порт, використовуваний слухачем для одержання запитів на установку підключення.

5.3 Запуск сеансу SQL*Plus з командного рядка.

Перш ніж підключатися до сеансу SQL*Plus, необхідно спочатку правильно набудувати середовище й указати, з якою базою даних на сервері повинне встановлюватися з'єднання за замовчуванням. Робиться це за допомогою змінного середовища ORACLE_SID.

Наприклад:

$ ORACLE_SID=orcl

$ export ORACLE_SID

Після вказівки бази даних, що повинна використатися за замовчуванням (у наведеному прикладі це orcl) у змінного середовища ORACLE_SID, можна одержувати доступ до SQL*Plus із запрошення командного рядка, просто вводячи команду **sqlplus** без імені користувача й пароля. SQL*Plus запропонує ввести ім'я користувача й пароль. У випадку надання імені користувача разом з командою (наприклад: sqlplus salapati), SQL*Plus буде запрошувати ввести тільки пароль. Адміністратор баз даних повинен входити від імені однієї з адміністративних облікових записів.

Зрозуміло, уводити ім'я користувача й пароль можна й безпосередньо при виклику SQL*Plus, але тоді пароль буде видний іншим при його уведенні. Нижче наведений приклад:

$ sqlplus salapati/sammyy1

SQL>

Запрошення SQL (SQL>) означає, що з'єднання з SQL*Plus ініційоване, і можна починати вводити команди і оператори SQL, PL/SQL і SQL*Plus.

Для того щоб підключитися до іншої бази даних, а не тієї, що встановлено по умовчанню, потрібно використати наступну команду:

$ sqlplus ім'я_користувача@ідентифікатор_підключення

Певні операції, наприклад запуск і завершення роботи, дозволене виполнять тільки у випадку підключення до SQL*Plus із привілеями SYSDBA або SYSOPER. При наявності привілеїв SYSDBA (або SYSOPER) підключатися до SQL*Plus можна наступним образом:

$ sqlplus sys/sammyy1 AS SYSDBA

SQL> SHO USER

USER is "SYS"

SQL>

Конструкція AS дозволяє встановлювати привілейовані підключення пользователям, яким були видані системні привілеї SYSDBA або SYSOPER.

Якщо в базі даних був створений обліковий запис аутентифицированного пользователя операційної системи (раніше називалося OPS$*ім'я*), установлювати підключення можна й просто вказівкою символу косої риси (/), як показано нижче:

$ sqlplus /
SQL> SHO USER
USER is "OPS$ORACLE"
SQL>

Можна також підключатися через метод аутентификации операційної системи, за рахунок включення власника програмного забезпечення Oracle у групу адміністраторів баз даних (DBA):

$ sqlplus / AS SYSDBA
SQL> SHO USER
USER is "SYS"
SQL>

У всіх попередніх прикладах ім'я бази даних при підключенні через SLQ*Plus не вказувалося. Пояснюється це тим, що підключення встановлювалося до прийнятого за замовчуванням екземпляру, тобто до бази даних, на яку вказує значення змінного середовища ORACLE_SID. Указувати ім'я бази даних при використанні SQL*Plus для підключення до прийнятого за замовчуванням базі даних не обов'язково. Для підключення до іншої бази даних, доступної по мережі, потрібно обов'язково використати ідентифікатор підключення (ім'я мережної служби).

Ім'я екземпляра, ім'я бази даних і ім'я служби можуть як збігатися, так і відрізнятися.
З теоретичної точки зору, підключатися до бази даних можна з використанням повного синтаксису ідентифікатора підключення, як показано в наступному прикладі, де для підключення до бази даних orcl застосовується вся адреса цілком:

$ sqlplus salapati/sammyy1@(DESCRIPTION =
(ADDRESS=(PROTOCOL=tcp)(HOST=sales-server)(PORT=1521)
(CONNECT_DATA= (SERVICE_NAME=orcl.mycompany.com)))

Однак за рахунок використання імені мережної служби, певного в мережному файле tnsnames.ora, можна підключатися до бази даних більше простим образом:

$ sqlplus salapati/sammyy1@orcl

Крім того, для підключення до бази даних можна застосовувати простий метод підключения. Синтаксис простого методу підключення виглядає так:

$ [//]хост[:порт][/[ім'я_служби]]

Наприклад, от як підключитися за допомогою цього методу до бази даних orcl:

$ sqlplus hr/hr_passwd@sales-server:1521/orcl.mycompany.com

У випадку застосування простого методу підключення мережний файл (tnsnames.ora) не потрібний. Який би з перерахованих методів не використався, в остаточному підсумку буде обов'язково успішно встановлюватися сеанс SQL*Plus або з базою даних за замовчуванням, або з тієї, що була зазначена в ідентифікаторі підключення.

Установка підключення за допомогою команди CONNECT. В SQL*Plus підтримується команда CONNECT, що дозволяє після входу в SQL*Plus виконувати підключення від імені іншого користувача. Крім того, вона позволяет після підключення до однієї бази даних підключатися до іншої бази даних.

Нижче наведений приклад використання команди CONNECT для виконання підключення від імені іншого користувача:

SQL> CONNECT новий_користувач/пароль
Connected.
SQL>

Наступний приклад демонструє, як в SQL*Plus підключатися до іншої бази
даних за рахунок надання ідентифікатора підключення у вигляді частини команди

CONNECT:
SQL> CONNECT salapati/sammyy1@orcl
Connected.
SQL>

Перед підключенням до іншої бази даних необхідно перевіряти, що у файлі tnsnames.ora є присутнім необхідна інформація про підключення до вилученої бази даних.

Команду CONNECT можна використати в SQL*Plus разом із синтаксисом / AS SYSDBA і / AS SYSOPER, як показано нижче:

CONNECT sys/sammy1@prod1 as sysdba
CONNECT / AS SYSDBA
CONNECT користувач/пароль AS SYSDBA
CONNECT / AS SYSOPER
CONNECT користувач/пароль AS SYSOPER

Запуск сеансу SQL*Plus без установки підключення до бази даних за допомогою параметра /NOLOG

Сеанс SQL*Plus можна також запускати й без установки підключення до бази даних, рахунок указавши разом з командою sqlplus параметр /NOLOG. У подібному може виникати необхідність, наприклад, при запуску базі даних або просто для використання доступних в SQL*Plus команд для запису або редагування сценаріїв. Після запуску сеансу SQL*Plus для підключення до бази даних завжди можна застосувати команду CONNECT.

Нижче наведений приклад використання параметра /NOLOG:

$ sqlplus /NOLOG
*SQL*Plus: Release 11.1.0.6.0 - Production on Wed Jan 2 18:35:25 2008*
Copyright (c) 1982, 2007, Oracle. All rights reserved.
SQL> SHO USER
USER is "
"
SQL> SHO SGA
SP2-0640: Not connected
SQL> CONNECT salapati/sammyy1
Connected.
SQL>

5.4 Підключення до SQL*Plus через графічний інтерфейс Windows

У випадку використання графічного інтерфейсу SQL*Plus на машині Windows для запуску сеансу SQL*Plus досить клацнути на піктограмі SQL*Plus і на екрані з'явиться запрошення ввести ім'я користувача. За умови, що з'єднання з базою даних установлюється через відповідні сутності у файлі tnsnames.ora після уведення імені користувача можна приступати до роботи з інтерфейсом SQL*Plus.

Працювати з утилітою SQL*Plus можна як у ручному, так і в сценарному не інтерактивному режимі. Саме собою зрозуміло, що уразливі адміністративні завдання, начебто відновлення бази даних, краще виконувати в інтерактивному режимі. Що ж стосується рутинних операцій по обробці SQL, те їхнє виконання краще автоматизи ровать за допомогою сценаріїв. І в тім і в інший випадку самі команди будуть виглядати однаково - відрізнятися буде лише режим, у якому вони будуть виконуватися.

Синтаксис команди підключення до SQL*Plus:

CONN[ECT] [{ реєстраційне_ім'я | / } [AS {SYSOPER | SYSDBA | SYSASM}]]

Підключатися від імені користувача із привілеями SYSOPER, SYSDBA або SYSASM необхідно для виконання привілейованих операцій, начебто завершення роботи й запуску бази даних або резервного копіювання або відновлення бази даних.

Привілей SYSAM є новою в Oracle Database 11*g* і призначена для поділу звичайних операцій по адмініструванню баз даних і операцій автоматичного керування пам'яттю (Automatic Storage Management — ASM).

5.5 Команди SQL та SQL*Plus

Після підключення до інтерфейсу SQL*Plus можна починати вводити в ньому будь-які команди SQL*Plus, SQL або PL/SQL. Як буде пояснюватися пізніше в цій главі, оператори SQL кінчаються або символом крапки з коми (;), або символом косої риси (/), а блоки коду PL/SQL - тільки символом косої риси (/). Висновок можна як переглядати на екрані, так і при бажанні записувати у файл. Команди SQL*Plus завжди кінчаються символів нового рядка. При уведенні команди SQL*Plus клієнтська програма SQL*Plus аналізує неї, і якщо та являє собою оператор SQL або PL/SQL, відправляє її серверу баз даних для обробки.

Як символ продовження можна використати дефіс (-), хоча при закінченні першого рядка застосовувати символ продовження зовсім не обов'язково. У кожної строке SQL можна вводити будь-яка кількість символів або слів і потім просто натискати клавішу <Enter> для продовження на наступному рядку. SQL*Plus буде автоматично додавати перед кожним рядком її номер.

У деяких випадках, однак, символ продовження (-) виявляється корисним, як у наступному прикладі, де потрібно ввести SQL-оператор SELECT 200 - 100 FROM dual:

SQL> SELECT 200 -
> 100 from dual;
select 200 100 from dual
***
ERROR at line 1:
ORA-00923: FROM keyword not found where expected
Не вдалося виявити ключове слово FROM там, де воно очікувалося
SQL>

У цьому прикладі через перехід на другий рядок після дефіса (-), що ще также є й знаком мінус, утиліта SQL*Plus автоматично інтерпретувала його як символ продовження й видала помилку, тому що оператор вийшов синтаксически некоректним (select 200 100 from dual). Уникнути цієї проблеми можна за рахунок використання наприкінці першого рядка другого дефіса (знака мінус) для выполнения ролі символу продовження:

SQL> SELECT 200 - -
> 100 FROM dual;
200-100
-i-i-i-i-i
100
SQL>

В Oracle для виконання певних запитів необхідно використати таблицю DUAL, оскільки в підтримуваному Oracle синтаксисі SQL наявність конструкції FROM в операторі SELECT є обов'язковим (наприклад, SELECT sysdate FROM dual;).

У базах даних Microsoft SQL Server, з іншого боку, використати таблицю DUAL не потрібно, тому що в синтаксисі SQL Server допускається застосування операторів SELECT без конструкції FROM.

Завершується сеанс SQL*Plus уведенням команди **EXIT**, причому як у нижньому, так і у верхньому регістрі. За допомогою команди QUIT здійснюється вихід у операційну систему (регістр символів теж ролі не грає).

У випадку виконання акуратного виходу з SQL*Plus по команді EXIT (або QUIT) буде негайно відбуватися фіксація всіх транзакцій. Якщо не потрібно, щоб відбувалася фіксація транзакцій, перед виходом буде потрібно виконати команду rollback.

5.6 Команди SQL*Plus і SQL

Інтерфейс SQL*Plus дозволяє взаємодіяти з базами даних Oracle. Зокрема, для цього в ньому основному застосовуються команди двох наступних видів.

***Локальні команди*.** Ці команди виконуються в SQL*Plus локальним образом і звичайно не відправляються на сервер. До їхнього числа ставляться такі команди, як COPY, COMPUTE, REM і SET LINESIZE. Всі вони кінчаються новим рядком і не бідують ні в якому спеціальному символі завершення.

***Команди, виконувані сервером*.** Ці команди не виконуються в SQL*Plus локально, а замість цього обробляються сервером. До їхнього числа ставляться всі інші команди, такі як SQL-команди CREATE TABLE і INSERT і команди PL/SQL, що містять в оператори BEGIN і END. Всі SQL-команди кінчаються або символом крапки з коми (;), або символом косої риси (/), а всі команди PL/SQL - тільки символом косої риси (/).

Безпека SQL*Plus. Крім обов'язкової вимоги вказувати ім'я користувача й пароль для роботи з інтерфейсом SQL*Plus, в Oracle також передбачений додатковий механізм захисту, дія якого полягає в застосуванні спеціальної таблиці product_ user_profile. Володіє цією таблицею користувач System, що є одним із двох суперкористувачів бази даних Oracle. За допомогою цієї таблиці такий користувач може обмежувати доступ до команд SQL*Plus, SQL і PL/SQL.

Коли той або інший користувач входить у сеанс SQL*Plus, SQL*Plus перевіряє таблицю product_user_profile для з'ясування того, які обмеження повинні застосовуватися до цього користувача протягом його сеансу. Те, як Oracle адмініструє цей рівень захисту, виглядає небагато заплутано. Користувач може володіти привиле гией на виконання в базі даних операцій вставки або видалення, але через те, що привілею SQL*Plus перевизначають такий привілей, Oracle може відмовити користувачеві в праві користуватися даним привілеєм.

Після створення бази даних для забезпечення підтримки механізму безпеки SQL*Plus потрібно виконати спеціальний сценарій pupbld.sql. Цей сценарій перебуває в каталозі $ORACLE_HOME/sql/admin і повинен виконуватися від імені користувача System. Саме він і буде приводити до побудови таблиці product_user_profile, що являє собою синонім для sqlplus_product_user_profile.

6. Призначення, основні оператори та синтаксис мови PL/SQL

PL/SQL - це процедурна блочно-структурована мова.

Являє собою розширення мови SQL та призначена для того, щоб позбутися від деяких обмежень, що існують у SQL, та мати можливість для повного програмного рішення розробникам життєво важливих додатків роботи з БД Oracle.

PL/SQL — це:

– високоструктурована, легка для читання та доступна мова;

- стандартна мова для розробки додатків на Oracle;

- вбудована мова (SQL*Plus, Oracle Developer);

– високопродуктивна та високоінтегрована мова для роботи з базами даних.

Можливості PL/SQL:

- реалізація підпрограм як окремих блоків, використання вкладених блоків;
- створення пакетів, процедур і функцій, що зберігаються в БД;
- надання інтерфейсу для виклику зовнішніх процедур;
- підтримка типів даних SQL та типів, що вводяться в PL / SQL;

- застосування явного і неявного курсора, а також оператора циклу FOR для курсора;
- введення у змінних PL / SQL та курсорів атрибутів, які дозволяють посилатися на тип даних або структуру елемента;
- введення типів **колекцій** і **об'єктних** типів;
- підтримка набору операторів управління і операторів циклу;
- реалізація механізму обробки винятків (исключений).

6.1 Структура блока PL/SQL

Основною програмною одиницею PL/SQL є **блок**,
він може містити вкладені блоки - **підблоки.**

Блок – це конструкція, що забезпечує виконання фрагмента коду і визначає межі видимості змінних і область дії обробників винятків.

Блок дозволяє об'єднувати оголошення і оператори, пов'язані загальною логікою.

PL/SQL дозволяє створювати

- *анонімні* блоки (блоки коду, що не мають назви)
- *іменовані* блоки (процедури, функції або тригери).

Розділи блоку

Блок PL / SQL може включати в себе до чотирьох розділів, лише один з яких є обов'язковим.

Заголовок

Використовується тільки для іменованих блоків. Тема визначає, яким чином буде викликатися іменований блок або програма. Необов'язковий розділ.

Розділ оголошень

Визначає змінні, курсори і подблоки, які згадуються в розділах виконання і винятків. Необов'язковий розділ.

Розділ виконання - Обов'язковий розділ

Містить оператори, які буде виконувати ядро PL / SQL при виконанні блоку ..

Розділ винятків

Обробляє виняткові (по відношенню до нормальної роботи) ситуації (попередження і помилки).

Необов'язковий розділ.

```
<<label_name>> ]          - Мітка блока
[DECLARE]                 - Секція оголошень
BEGIN                     - Тіло блока
.
.
[EXCEPTION]               - Обробники винятків
END [label_name];
```

Анонімні блоки. У ньому ***відсутній розділ заголовку***, починається з DECLARE або BEGIN.

Його не можна буде викликати з іншого блоку, оскільки немає на що встановити посилання.

Служать контейнерами для операторів PL / SQL і зазвичай включають в себе виклики процедур та функцій.

Синтаксис анонімного блока PL/SQL:

[DECLARE
... оголошення ...]
BEGIN
... один або декілька виконуваних операторів ...

[EXCEPTION
... оператори обробки винятків ...]
END;
Короткий анонімний блок:
BEGIN
DBMS_OUTPUT.PUT_LINE(SYSDATE);
END;
Блок з такою самою функціональністю, в який було додано розділ оголошень:
DECLARE
l_right_now VARCHAR2(9);
BEGIN
l_right_now := SYSDATE;
DBMS_OUTPUT.PUT_LINE(l_right_now);
END;
Блок с обработчиком исключений:
DECLARE
l_right_now VARCHAR2(9);
BEGIN
l_right_now := SYSDATE;
DBMS_OUTPUT.PUT_LINE(l_right_now);
EXCEPTION
WHEN VALUE_ERROR
THEN
DBMS_OUTPUT.PUT_LINE('I bet l_right_now is too small for the default date format!')
END;

6.2 Набір символів PL/SQL

Програма на PL/SQL складається з послідовності операторів, кожен з яких утворений одним або кількома рядками тексту.

Набір символів, з яких можна складати ці рядки тексту, залежить від використовуваного в базі даних набору символів.

Тип	**Символи**
Літери	A-Z, a-z
Цифри	0–9
Символи	~ ! @ # $ % * ( ) _ - + = | : ; " ' < > , . ? / ^
Символи пробілу	Знак табуляції, знак пробілу, символ нового рядка, кінець рядка

Групи символів утворюють **лексеми**, які також називають атомарними одиницями мови, адже вони є її найменшими неподільними складовими.

Лексемами в PL / SQL є ***ідентифікатори, літерали, розділювачі та коментарі.***

PL/SQL **не чутливий до регістру**,

за виключенням **строкових змінних та констант**.

Кожна конструкція PL / SQL **має закінчуватися** символом **;**
Одна конструкція може бути розташована на декількох рядках.
Прості та складові символи PL/SQL

Символ	**Опис**
;	Крапка з комою завершує оголошення та оператори.
%	Знак процента вказує на атрибути (атрибути курсора, такі як % ISOPEN і атрибути непрямого оголошення, наприклад, % ROWTYPE); також використовується як багатобайтний символ підстановки за умови LIKE.
_	Одинарне підкреслення: одинарний груповий символ одноколісного символу за умови LIKE.
@	Знак @ вказує на віддалене місцезнаходження.
:	Двокрапка є індикатором хост-змінної, наприклад, :block.item та Oracle Forms.
**	Подвійна зірочка — це оператор зведення у ступінь.
< > або != або ^= або ~=	Способи позначення оператора відношення «не дорівнює».
||	Подвійна вертикальна риска — це знак операції **конкатенації (**склеивание).
<< і >>	Розділювачі міток.
<= і >=	Оператори відношень «менше або дорівнює» та «більше або дорівнює».
:=	Оператор присвоювання.
=>	Оператор зв'язування для зв'язування за ім'ям.
..	Дві крапки — оператор діапазона.
--	Подвійний дефіс вказує на однорядковий коментар.
/* і */	Початковий та кінцевий обмежувачі багаторядкового коментаря.

6.3 Ідентифікатори

Ідентифікатор - **це ім'я об'єкта** PL / SQL (ім'я змінної або програми, зарезервоване слово).

Ідентифікатори мають такі властивості:

• Довжина до **30** символів

• Повинні починатися **з літери**

• Можуть включати в себе знаки $, (_) і (#)

• Не можуть містити символи пробілу

Якщо два ідентифікатора відрізняються тільки регістром однієї або декількох літер, то PL / SQL сприймає їх як один і той самий ідентифікатор.

Приклад, PL / SQL сприймає наступні ідентифікатори як ідентичні:

lots_of_$MONEY$

LOTS_of_$MONEY$

Lots_of_$Money$

Літерали

Літерал — це значення, яке не представлено ідентифікатором,

тобто просто значення, яке існує саме по собі.

Рядкові літерали

Рядковий літерал — це текст, наведений в одинарних лапках, наприклад:

'What a great language!‘

На відміну від ідентифікаторів, рядкові літерали в PL / SQL **чутливі до регістру,** через це наступні два литерали сприймаються як різні:

'Steven'

'steven‘

Тому така умова буде обчислена як FALSE:

IF 'Steven' = 'steven'

Числові літерали

Числові літерали можуть бути цілими або дійсними числами (з дробом).

PL / SQL вважає число 154.00 ***дійсним числом типу*** NUMBER, хоча його дробова частина дорівнює нулю і насправді число є цілим.

Для запису числового літералу можна використовувати експонентний формат.

Буква «E» (у верхньому або нижньому регістрі) у записі числа означає його множення на 10 у відповідному ступені, наприклад: 3.05E19, 12e5.

Літерали з плаваючою точкою можуть мати бінарне подання зі звичайною (32 біта, в кінці ставиться літера «F») або з подвійною точністю (64 біта, в кінці ставиться «D»).

У деяких виразах можна використовувати іменовані константи, зазначені в стандарті IEEE (Institute of Electrical and Electronics Engineers – Інститут інженерів з електротехніки та електроніки).

Логічні літерали

PL/SQL пропонує два літерали для репрезентації логічних (булевих) значень:

TRUE та FALSE.

Ці значення не є рядковими**, їх не слід брати в лапки.**

Використовуйте логічні літерали для присвоювання значень логічним змінним, наприклад:

DECLARE

enough_money BOOLEAN; Оголошення логічної змінної

BEGIN

enough_money := FALSE; Надання їй значення

END;

При перевірці значення логічного виразу не обов'язково посилатися на літерал.

Вираз буде «говорити сам за себе», як в умові наступного оператора IF:

DECLARE
enough_money BOOLEAN;
BEGIN
IF enough_money
THEN
...

Логічний вираз, змінна або константа також **можуть приймати значення NULL,** що не є ані TRUE, ані FALSE.

Програмні дані

Будь-який написаний блок PL / SQL буде визначати програмні дані та їх обробляти.

Програмні дані представляють собою структури даних, які існують тільки в рамках вашого сеансу (фізично вони знаходяться в програмній глобальній області (Program Global Area - PGA) вашого сеансу) та не зберігаються в базі даних.

Перш ніж ви зможете приступити до роботи зі змінною або константою, її **необхідно оголосити,** а після оголошення **призначити їм ім'я й тип даних**.

При виборі імен для змінних, констант і типів даних необхідно дотримуватися двох основних рекомендацій:

Обов'язково переконайтеся в тому, що кожна назва точно відображає призначення об'єкта та її можна легко зрозуміти;

Виробіть продумані й послідовні принципи іменування.

Подібні принципи зазвичай стосуються застосування префіксів і / або суфіксів для відображення типу та призначення.

Наприклад, імена всіх локальних змінних повинні починатися з префікса **«l_»,** а імена глобальних змінних, що визначаються в пакеті, повинні мати префікс **«g_».**

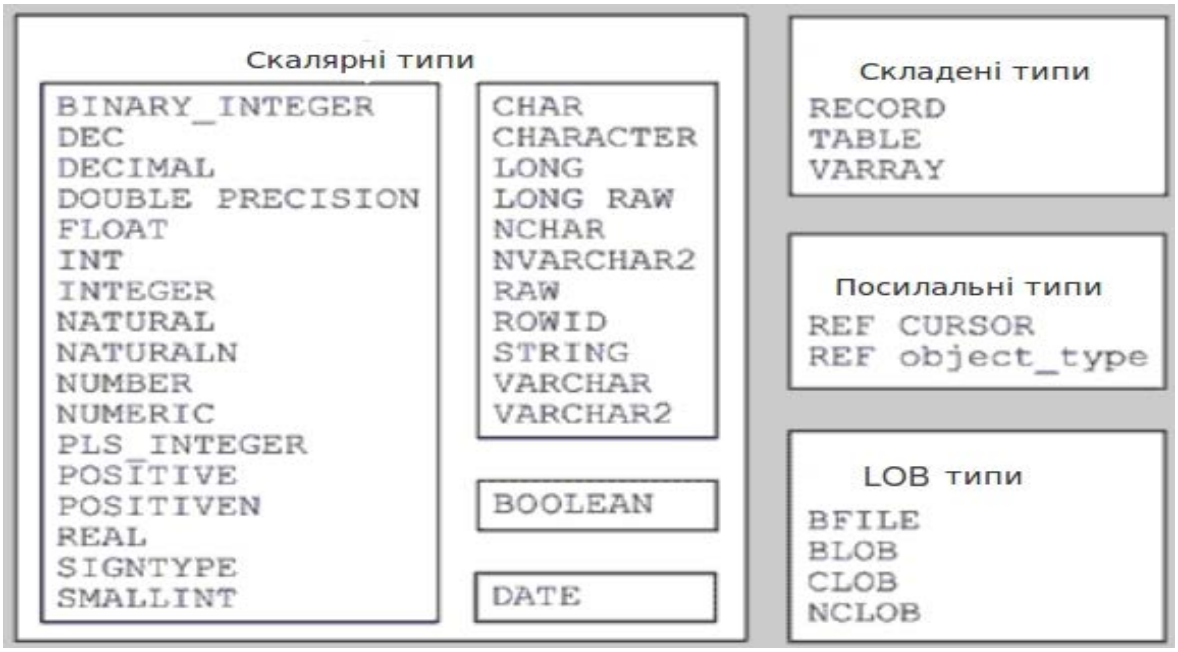
Імена типів записів повинні включати в себе суфікс «_rt» тощо.

Типи даних

Мова PL / SQL підтримує наступні категорії типів:

- вбудовані типи даних, зокрема, колекції і записи;
- об'єктні типи даних.

Вбудовані типи даних



Мова PL / SQL дозволяє визначати нові підтипи як підмножину значень деякого базового типу з тим самим набором операцій.

Підтип не вводить ніяких додаткових операцій над даними і не визначає ніякого нового типу.

Визначення підтипу може мати наступний формальний опис:

SUBTYPE subtype_name IS base_type;

Користувач може визначити свій тип як деякий підтип в секції оголошень блоку, підпрограми або пакеті PL / SQL.

Наприклад:

```
DECLARE
  SUBTYPE MyDate IS DATE;              /    - Заснований на типі DATE
  TYPE MyRec IS RECORD
        (time1 INTEGER,
         time2 INTEGER);
  SUBTYPE MyInterval IS MyRec;         /    - Заснований на типі RECORD
  SUBTYPE ID_N IS tbl1.f1%TYPE;        /    - Заснований на типі стовпцю
```

Типи, що використовуються як базові, не можуть містити обмежень довжини.

Для створення типу з обмеженням довжини, створеного самим користувачем, попередньо оголошується змінна такого типу і вже на її основі визначається призначений для користувача тип.

Наприклад:

```
DECLARE
  var1  VARCHAR2 (6);   -- Оголошення змінної з обмеженням довжини

  SUBTYPE string_6 IS var1%TYPE;
```

LOB-типи

LOB-типи використовуються для зберігання великих об'єктів (Large Object).

Стандарт SQL-99 ввів підтримку LOB-типів для розширеного рівня відповідності.

Повний набір LOB-типів в Oracle дозволяє зберігати дані об’ємом **до 4 Гбайт.**

Типи LOB відрізняються від типу **LONG** тим, що при виборі значення будь-якого LOB-типу за допомогою оператора SELECT **повертається покажчик**, а не саме значення; окрім того, типи LOB можуть бути й зовнішніми.

Oracle підтримує наступні **чотири типи для великих об'єктів:**

BFILE - для зовнішнього двійкового файлу;

BLOB - для внутрішнього двійкового об'єкта;

CLOB - для внутрішнього символьного об'єкта;

NCLOB - для внутрішнього символьного об'єкта, знаціональний набір символів.

Будь-який об'єкт LOB складається з двох частин:

- даних;
- покажчика на ці дані, що називається локатором.

Типи BLOB, CLOB або NCLOB можуть використовуватися як для стовпця бази даних, так і для змінної PL / SQL.

Для завантаження об'єкта LOB передбачений пакет ***PL/SQL DBMS_LOB.***

Пакет DBMS_LOB для роботи з LOB-типами містить процедури та функції.

Синтаксис	**Опис**
APPEND (d1,d2)	Додає d2 до d1

COMPARE(d1,d2,n,pos1,pos2)	Порівнює n байт значень d1 та d2
COPY (d,s,n,dp,sp)	Копіює n байт з d до s.
FILEOPEN (bdata,m)	Відкриває об’єкт типу BFILE в режимі, вказаному параметром m
LOADFROMFILE (bdata1,data2,n,pos1,pos 2)	Копіює n байт об’єкту типу BFILE bdata1 в будь-який об’єкт LOB data2
GETLENGTH (data)	Повертає довжину вказаного об’єкту LOB
READ (data,n,pos,buf)	Читає з об’єкту data n байт
WRITE (data,n,pos,buf)	Копіює з буферу buf n байт
EMPTY_CLOB (), EMPTY_BLOB ()	Створює "пустий" об’єкт вказаного типу

Наприклад:

```
DECLARE
 pf1 CLOB;
pf2 BLOB;
buf varchar2;
BEGIN
CREATE TABLE tbl1
 ( f1 CLOB, f2 BLOB);
INSERT INTO tbl1 VALUES (empty_clob(),empty_blob() );
 SELECT f1
INTO pf1
FROM tbl1
 FOR UPDATE; buf := 'Текст, який буде вставлено в объект LOB';
 DBMS_LOB.write (pf1, length(buf), 0, buf);
END;
```

6.4 Структури програми PL/SQL

6.4.1 Структури керування обчисленнями

Для реалізації алгоритмів процедурною мовою програмування потрібне наявність у ньому трьох наступних структур керування обчисленнями, причому повинна бути можливість вкладати всі структури друг у друга довільним образом:

- послідовність команд (виконання команд відповідно до їх упорядкованості);
- вибір (перевірка умови й виконання тієї або інший послідовністю команд залежно від істинності або хибності умови);
- повторення (виконання послідовності команд доти, поки умова повторення приймає щире значення).

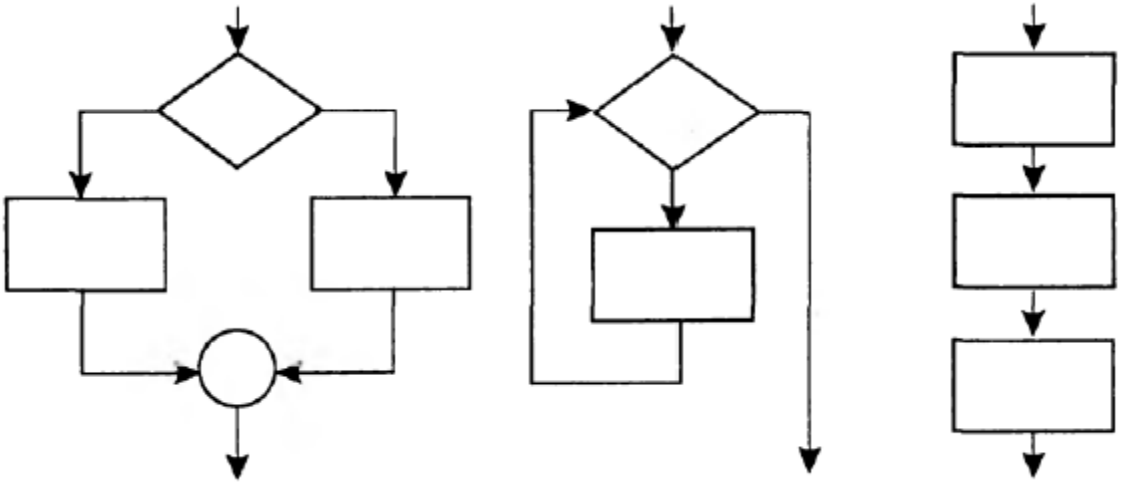


Рис. 1. Структури керування обчисленнями.

Команди, зазначені в коді PL/SQL, виконуються послідовно. Така схема називається потоком команд, тобто перша необхідна структура керування обчисленнями (послідовність) в PL/SQL є. Розглянемо язикові конструкції PL/SQL для вибору й повторення (умовні команди, команди переходу і цикли).

6.4.2 Умовні команди

В PL/SQL до умовних команд ставляться команди IF і CASE, пер-ходи реалізуються командами GOTO і NULL.

Умовна команда IF. Умовна команда IF дозволяє перевірити задану логічну умову й, залежно від результатів перевірки (TRUE, FALSE або UNKNOWN), виконати різні галузі коду.

В PL/SQL є три форми команди IF:

- IF THEN END IF
- IF THEN ELSE END IF
- IF THEN ELSIF ELSE END IF

Границі дії команди IF визначаються закриваючими ключовими словами END IF. Альтернативна послідовність команд треба після ключового слова ELSE, для розширення структури розгалуження додатково можна задати додаткову умову після ключового слова ELSIF.

Команда IF має наступний синтаксис:

```
IF умова 1 THEN команди 1;        -галузь 1
ELSE
IF умова 2 THEN команди 2;  - галузь 2
ELSE команди n;  - альтернативна послідовність команд (ELSE-галузь)
END IF;
```

Команди першої галузі коду виконуються тільки тоді, коли перевірка умови визначає його істинність (TRUE).

Якщо ж перевірка умови визначає FALSE або UNKNOWN (наприклад, при порівнянні з NULL), то виконуються команди ELSE-галузі.

Приклад програми, що виводить повідомлення про клас випромінювання залежно від значення параметра довжини, що вводить, хвилі (довжина хвилі передбачається заданої в мікронах).

```
SQL> DECLARE
2 lamda NUMBER; -і Довжина хвилі
3 text1 VARCHAR2(30) := 'Інфрачервоне випромінювання';
4 text2 VARCHAR2(30) := 'Видиме світло';
5 text3 VARCHAR2(30) := 'Ультрафіолет';
6 -і виконує раздел
7 BEGIN
8 lamda := &Input_Data;
9 DBMS_OUTPUT.PUT_LINE('');
10 IF (lamda > 0.65)
11 THEN DBMS_OUTPUT.PUT_LINE(text1);
12 ELSIF (lamda < 0.41)
13 THEN DBMS_OUTPUT.PUT_LINE(text3);
14 ELSE
15 DBMS_OUTPUT.PUT_LINE(text2);
16 END IF;
17 END;

Enter value for input_data: 0.33
old 8: lamda := &Input_Data;
new 8: lamda := 0.33;
Ультрафіолет
PL/SQL procedure successfully completed.
```

Умовна команда CASE. При складній логіці розгалуження й перевірці численних умов рекомендується замість вкладених команд IF використати умовну команду CASE, тому що з нею виходить більше зрозумілий і компактний код.

Команда CASE має два різновиди:

- проста команда CASE, що зв'язує одну або кілька послідовностей команд PL/SQL з деякими значеннями (виконувана послідовність команд вибирається при збігу результату обчислення заданого вираження зі значенням, пов'язаним із цією послідовністю команд);

- пошукова команда CASE, що вибирає для виконання послідовність команд залежно від результатів перевірки списку логічних умов (виконується послідовність команд, пов'язана з першою логічною умовою в списку, результат перевірки якого виявився рівним TRUE).

Незважаючи на громіздкий опис, працювати з командою CASE обох різновидів просто й зручно.

Проста команда CASE має наступний синтаксис:

```
CASE вираження
```

```
  WHEN результат 1 THEN
        послідовність команд 1;
  WHEN результат 2 THEN
        послідовність команд 2;
…
ELSE
      альтернативна послідовність команд;
END CASE;
```

Проста команда CASE звичайно використається для рятування від численних команд IF і конструкцій ELSE у них шляхом формування добре структурованих галузей коду залежно від списку значень, які може приймати деяка керуюча змінна. Приклад пошуку слова російською мовою по англійському аналогу:

```
SQL> DECLARE
2 english_termin VARCHAR2(20);
3 text1 VARCHAR2(30) := 'Інфрачервоне випромінювання';
4 text2 VARCHAR2(30) := 'Видиме світло';
5 text3 VARCHAR2(30) := 'Ультрафіолет';
6 text4 VARCHAR2(30) := 'Невідомий термін';
7 BEGIN
8 english_termin := &Input_Data;
9 CASE english_termin
10 WHEN 'Infrared radiation' THEN DBMS_OUTPUT.PUT_LINE(text1);
11 WHEN 'Visible light' THEN DBMS_OUTPUT.PUT_LINE(text2);
12 WHEN 'Ultraviolet' THEN DBMS_OUTPUT.PUT_LINE(text3);
13 ELSE DBMS_OUTPUT.PUT_LINE(text4);
14 END CASE;
15 END;
16/
Enter value for input_data: 'Ultraviolet'
old 8: english_termin := &Input_Data;
new 8: english_termin := 'Ultraviolet';
Ультрафіолет
PL/SQL procedure successfully completed.
```

Пошукова команда CASE має наступний синтаксис:

```
CASE
WHEN вірно логічна умова 1 THEN
послідовність команд 1;
WHEN вірно логічна умова 2 THEN
послідовність команд 2;
…
ELSE
альтернативна послідовність команд;
END CASE;
```

Перепишемо приклад визначення джерела випромінювання з використанням пошукової команди CASE замість команди IF:

```
SQL> DECLARE
2 lamda NUMBER;
```

```
3 text1 VARCHAR2(30) := 'Інфрачервоне випромінювання';
4 text2 VARCHAR2(30) := 'Видиме світло';
5 text3 VARCHAR2(30) := 'Ультрафіолет';
6 BEGIN
7 lamda := &Input_Data; 8 CASE
9 WHEN (lamda > 0.65)
10 THEN DBMS_OUTPUT.PUT_LINE(text1);
11 WHEN (Lamda < 0.41)
12 THEN DBMS_OUTPUT.PUT_LINE(text3);
13 ELSE
14 DBMS_OUTPUT.PUT_LINE(text2);
15 END CASE;
16 END;
17/
Enter value for input_data: 0.50
old 7: lamda := &Input_Data;
new 7: lamda := 0.50;
Видиме світло
PL/SQL procedure successfully completed.
```

Крім умовної команди CASE, в PL/SQL є вираження CASE, що послідовно обчислює логічні вираження із заданого списку, поки одне з них не стане щирим, а потім повертає результат пов'язаного з ним вираження. У цьому укладається відмінність вираження CASE від команди CASE - команда CASE управляє потоком інших команд, а вираження CASE обчислюється і його значення може бути привласнено змінним, використано в інших вираженнях і т.д.

```
DECLARE
lamda NUMBER := 0.50;
text1 VARCHAR2(30) := 'Інфрачервоне випромінювання';
text2 VARCHAR2(30) := 'Видиме світло';
text3 VARCHAR2(30) := 'Ультрафіолет';
answer VARCHAR2(30);
BEGIN
answer := CASE WHEN (lamda > 0.65) THEN text1
WHEN (Lamda < 0.41) THEN text3
ELSE text2
END;
DBMS_OUTPUT.PUT_LINE(answer);
END;
```

6.4.3 Команда переходу

Команда переходу GOTO дозволяє здійснити перехід по мітці, що є присутнім у коді PL/SQL. За допомогою унікального ідентифікатора, ув'язненого в подвійні кутові дужки (мітки), можна позначити будь-яку частину блоку, що виконує, PL/SQL для переходу в це місце.

```
SQL> DECLARE
2 s NUMBER := 1;
3 BEGIN
4 IF s = 1 THEN
5 GOTO mybranch; -і перехід по мітці mybranch
6 ELSE
```

```
7 s := 1;
8 END IF;
9 <<mybranch>> -i установка мітки mybranch
10 NULL;
11 END;
PL/SQL procedure successfully completed.
```

Команда NULL. Команда NULL («порожня» команда) звичайно використається як «заглушка» у місці, де треба написати яку-небудь команду, тому що нічого не написати там не можна по вимогах синтаксису PL/SQL. Потім, у міру появи визначеності, «заглушка» заміняється на функціональний код:

```
CASE sex
WHEN 'M' THEN
sex_decoded := 'male';
WHEN 'F' THEN
sex_decoded := 'female';
ELSE
NULL; -i toDo: write code for exception sex not in list {F,M} ;))
END CASE;
```

Команда NULL використається при обробці виключень, коли обробка якого-небудь виключення укладається у відсутності яких-небудь дій.

6.4.4 Цикли

У мові PL/SQL є три види циклів:

- простий цикл, що починається із ключового слова LOOP і завершується командою END LOOP;
- цикл WHILE із предусловием, що дозволяє виконати ту саму послідовність команд, поки перевіряє условие, що, істинно;
- цикл FOR з лічильником.

Простий цикл розглянемо на прикладі визначення числа, факторіал якого є найменшим числом, що вперше перевищує задану константу (1 000 000 000).

```
 SQL> DECLARE
2 arg NUMBER; -i Змінна для обчислення факторіала
3 i NUMBER; -i Змінн-лічильник
4 limit NUMBER := 1000000000; -i Границя
5 text1 VARCHAR2(80) := 'n! числа, що вперше перевищує 1000000000';
6
7 BEGIN
8 i := 0;
9 arg := 1;
10 LOOP
11 EXIT WHEN arg > limit;
12 arg := arg*(i+1);
13 i := i + 1;
14 END LOOP;
15 DBMS_OUTPUT.PUT_LINE(text1);
16 DBMS_OUTPUT.PUT_LINE(TO_CHAR(arg));
17 DBMS_OUTPUT.PUT_LINE('Шукане число = '||TO_CHAR(i));
18 END;
```

```
/
n! числа, що вперше перевищує 1000000000
6227020800
Шукане число = 13
PL/SQL procedure successfully completed.
```

З будь-якого циклу в PL/SQL можна вийти командою EXIT із вказівкою логічної умови виходу. В основному команда EXIT використається в простих циклах, тому що в циклах FOR і WHILE і так явно вказуються умови закінчення циклу, а мати в коді більше однієї умови закінчення для циклу є поганим стилем програмування.

Якщо відбувається зациклення (виконання нескінченного циклу без виходу з нього), то програма PL/SQL «іде в себе» («повисає»).

Для припинення виконання такої програми в SQL*Plus варто нажати на клавіатурі комбінацію клавіш Ctrl+C:

```
SQL> BEGIN LOOP NULL; END LOOP; END;
```

Цикл WHILE із предусловием дозволяє виконати ту саму послідовність команд PL/SQL поки перевіря істинно предусловие.

За допомогою циклу WHILE знайдемо число, факторіал якого є найменшим числом, що вперше перевищує 1 000 000 000 000:

```
SQL> DECLARE
2 arg NUMBER; -і Змінна для обчислення факторіала
3 i NUMBER; -і Змінн-лічильник
4 limit NUMBER := 1000000000000;-і Границя
5 text1 VARCHAR2(80):='n! числа, що вперше перевищує 1000000000000;
6
7 BEGIN
8 i := 0;
9 arg := i;
10 WHILE arg < 1000000000000 LOOP
11 arg := arg*(i+1);
12 i := i + 1;
13 END LOOP;
14 DBMS_OUTPUT.PUT_LINE(text1);
15 DBMS_OUTPUT.PUT_LINE(TO_CHAR(arg));
16 DBMS_OUTPUT.PUT_LINE('Шукане число = '||TO_CHAR(i));
17 END;
/
n! числа, щовперше перевищує 1000000000000
1307674368000
Шукане число = 15
PL/SQL procedure successfully completed.
```

Якщо умова циклу WHILE споконвічно ложно (FALSE), то цикл не виконається жодного разу.

Цикл FOR («цикл із лічильником»), використається в тому випадку, коли відомо, скільки разів потрібно виконати ітерацію циклу. Приведемо приклад обчислення факторіала заданого числа.

```
SQL> DECLARE
2 arg NUMBER := 1;
3 n NUMBER := 20;
4 text1 VARCHAR2(30) := 'Факторіал числа '||n||' = ';
5 BEGIN
```

```
6 FOR i IN 1..n LOOP
7 arg := arg*i;
8 END LOOP;
9 DBMS_OUTPUT.PUT_LINE(text1||TO_CHAR(arg));
10 END;
/
Факторіал числа 20 = 2432902008176640000
PL/SQL procedure successfully completed.
```

Лічильник - керуючу змінну циклу (i) повідомляти в розділі оголошень не потрібно, вона оголошується автоматично з областю видимості між ключовими словами LOOP і END LOOP.

При використанні FOR їсти можливість зробити так, щоб значення лічильника циклу не зростали, а зменшувалися, і є можливість нетривіальних, не на одиницю, збільшень кроку лічильника циклу.

Цикл із ключовим словом REVERSE

```
SQL> BEGIN
2 FOR i IN REVERSE 1..5 LOOP
3 DBMS_OUTPUT.PUT_LINE(i);
4 END LOOP;
5 END;
6/
5
4
3
2
1
PL/SQL procedure successfully completed.
```

Цикл із лічильником, кратним 10

```
SQL> BEGIN
2 FOR i IN 1..20 LOOP
3 IF MOD(i,10)=0 THEN
4 -і тіло циклу
5 DBMS_OUTPUT.PUT_LINE(i);
6 END IF;
7 END LOOP;
8 END;
9/
10
20
PL/SQL procedure successfully completed.
```

Для зворотного циклу в конструкції FOR LOOP варто вказати ключове слово REVERSE.

Для нетривіальних збільшень потрібно усередині циклу за допомогою команди IF просто пропускати кроки з непотрібними значеннями лічильника. Лічильник циклу завжди змінюється на одиницю, просто на деяких кроках циклу нічого робити не потрібно.

Команда CONTINUE з'явилася у версії Oracle 11g. При істинності умови, записаного в конструкції WHEN команди CONTINUE, виконання поточної ітерації циклу припиняється й відбувається перехід на початок наступної ітерації.

Завдяки команді CONTINUE можна, наприклад, винести перевірки в початок циклу.

Перепишемо наведений вище приклад з нетривіальним збільшенням лічильника циклу FOR без приміщення коду обробки в умовну команду IF:

```
BEGIN
FOR i IN 1..20 LOOP
CONTINUE WHEN MOD(i,10)!=0;
DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
END;
```

6.5 Побудова запитів за допомогою утиліти SQLPlus

За допомогою утиліти SQLPlus можна виконати усі операцій адміністрування та створення об'єктів БД.

Для запуску серверу натиснув кнопку Пуск\Все програми\ Oracale Database 11g\Start Database. Відкриється вікно для введення DOS команд.

Запустите утиліту SQLPlus, набрав наступну команду та натисніть кнопку «Enter»:

```
C:\Windows\system32> SQLplus
```

,

Як що з'явиться повідомлення про підключення, можна виконати підключення до серверу БД.

```
SQL*Plus: Release 11.2.0.2.0 Production on ≤э |ЊЕ 12 22:39:43 2018

Copyright (c) 1982, 2010, Oracle.  All rights reserved.
```

Підключитися до серверу, введіть логін - **system** та пароль – **masterkey**. Натисніть кнопку «Enter».

```
Enter user-name: system
Enter password:
```

Якщо з'явиться повідомлення про підключення, можна виконати підключення до навчальної БД компанії Oracle з назвою «HR».

```
Connected to:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - Production
```

Підключитися до БД HR, введіть логін – **UspStudCAD21** та пароль –**masterkey**.

```
SQL> connect CAD21HR
Enter password:
Connected.
```

Виконайте запит з агрегатною функцією «Max», знайдіть максимально значення окладу співробітників у таблиці «Employee»:

```
SQL> select max(salary) from employee;

MAX(SALARY)
-----------
       1000
```

Створити уявлення для розрахунку середнього окладу:

```
SQL> create view AvgSalary as
  2  select avg(salary) as sredoklad
  3  from employee;

View created.
```

Перевірте виконання уявлення:

```
SQL> select* from AvgSalary;

 SREDOKLAD
----------
       550
```

Бібліографія

1. Астахов И.Ф., Потапов А.С., Чулюков В.А., Стариков В.Н. Практикум по информационным системам. Oracale/ 2-изд., К., Юниор, 2004. -180 с.
2. Р. Гринвальд, Р. Стаковьяк. Oracle llg. Основы, 4-е издание. СПб.: Символ-Плюс, 2009.
3. Грег Риккарди. Системы баз данных. Теория и практика использования в Internet и среде Java. Вильямс, 2001.
4. Алапати Сэм Р. Oracle Database 11g: руководство администратора баз данных. : Пер. с англ. -М. : ООО “И.Д. Вильямс”, 2010. — 1440 с. : ил.
5. Oracle для профессионалов. Пер. с англ./ТомКайт- СПб.:ООО «ДиаСофтЮП», 2003. - 672 с.
6. Кайт Томас, Кун Дарл. Oracle для профессионалов: архитектура и методики программирования, 3-е изд.: Пер. с англ. - М.: 000 "ИД. Вильямс", 2016. - 960 с.