МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

ЕЛЕКТРОННИЙ НАВЧАЛЬНО-НАОЧНИЙ ПОСІБНИК

СХЕМОКОНСПЕКТ ЛЕКЦІЙ
з навчальної дисципліни
«ЗАХИСТ РОЗПОДІЛЕНИХ БАЗ ДАНИХ ТА ІНФОРМАЦІЙНИХ СИСТЕМ»
для студентів всіх форм навчання
галузі знань 12 Інформаційні технології

Луцьк, 2023

УДК 37:004:316.772.5

Схемоконспект лекцій з навчальної дисципліни «Захист розподілених баз даних та інформаційних систем» для студентів всіх форм навчання галузі знань 12 Інформаційні технології : електронний навчально-наочний посібник / уклад. Я.Ю. Дорогий. – Луцьк : ДонНТУ, 2023. – 501 с.

Навчальна дисципліна «Захист розподілених баз даних та інформаційних систем» є актуальною і затребуваною в контексті підготовки сучасних фахівців у галузі інформаційних технологій та кібербезпеки та з огляду на формування когнітивних, афективних та моторних компетентностей в контексті вивчення та розуміння принципів організації та реалізації віддаленої обробки даних з використанням технології "клієнт-сервер" та систем управління базами даних (СУБД), які підтримують цю технологію.

Пропонований студентам для опанування схемоконспект лекцій з цієї дисципліни має за мету забезпечити їм певну допомогу під час вивчення ними ключових положень базових тем курсу. Схематична форма конспекту акцентує увагу студентів на закріпленні основних фахових знань, формуванні відповідних умінь і в цілому отриманні ними кваліфікаційних компетентностей.

Укладач: _____ Я. Ю. Дорогий, доцент, д. т. н., професор кафедри ПМІ

Рецензент: _____ В. В. Цуркан, доцент, к. т. н., доцент спеціальної кафедри №5 ІСЗЗІ КПІ ім. Ігоря Сікорського,

Відповідальний за випуск: Н. О. Маслова, доцент, к. т. н., зав. каф. ПМІ

Затверджено навчально-методичним відділом ДонНТУ, протокол № 4 від 26.12.2023 р.

Розглянуто на засіданні кафедри прикладної математики і інформатики, протокол № 12 від 15.12.2023 р.

ЗМІСТ

Тема 6.	Моніторинг та аудит безпеки в БД Oracle.	
	Лекція 8. Техніка аудиту в Oracle.	226
	Лекція 9. Техніка аудиту в Oracle (продовження).	277
Тема 7.	Тема 7. Резервне копіювання та відновлення БД Oracle.	
	Лекція 10. Резервне копіювання та відновлення у Oracle.	304
	Лекція 11. Процеси резервного копіювання та відновлення у Oracle.	335
Тема 8.	Робота з SQL.	
	Лекція 12. Робота з SQL. Складні запити.	362
Тема 9.	Мова програмування PL/SQL.	
	Лекція 13. Вступ до PL/SQL.	452
	Лекція 14. PL/SQL. Управляючі структури. Складені типи.	527
	Лекція 15. PL/SQL. Використання явних курсорів.	583
Тема 10.	Розробка програмних застосунків в Oracle APEX.	
	Лекція 16. Вступ до Oracle APEX.	681
	СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ ДЛЯ САМОПІДГОТОВКИ	**713**

ВСТУП

У сучасних умовах, що характеризуються високою мінливістю та невизначеністю, функціонування держави та суб'єктів господарювання пов'язане з підвищеним ризиком. Це зумовлено виникненням та діями різноманітних загроз як у зовнішньому, так і у внутрішньому середовищі.

Негативні наслідки реалізації загроз можуть бути руйнівними, тому їх нейтралізація є актуальним завданням. Для цього необхідно своєчасно ідентифікувати загрози, розробити та реалізувати ефективні заходи захисту.

Одним із важливих елементів забезпечення безпеки є захист розподілених баз даних та інформаційних систем (РБДІС). РБДІС є критичною інфраструктурою, від надійності та безпеки якої залежить ефективність діяльності держави та суб'єктів господарювання.

Курс "Захист розподілених баз даних та інформаційних систем" спрямований на формування у студентів знань та навичок, необхідних для забезпечення безпеки РБДІС.

Для формування практичних навичок студентам пропонується виконання лабораторних робіт та самостійних завдань.

Стисле, але вичерпне подання матеріалу курсу сприятиме формуванню у студентів необхідних знань, умінь та компетентностей у сфері захисту РБДІС.

Донецький національний технічний університет

Тема 1. Мета, задачі, зміст курсу. Основні визначення. Лекція 1. Вступ до дисципліни

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Мета курсу
- Основна термінологія
- Структура курсу

Мета курсу

- формування компетентностей в контексті вивчення та розуміння принципів організації та реалізації розподіленої обробки даних з використанням технології "клієнт-сервер" та систем управління базами даних (СУБД), які підтримують цю технологію
- розвиток теоретичних та практичних навичок студентів у сфері використання відповідного алгоритмічного та програмного забезпечення, а також СУБД (зокрема, на прикладі СУБД Oracle)
- оволодіння принципами створення та ведення баз даних і способами забезпечення інформаційної безпеки засобами систем управління базами даних
- створення необхідного комплексу навичок і знань, необхідних для вирішення задачі побудови захищеної розподіленої системи управління базою даних на базі Oracle

Інформація

Інформація - це відомості про осіб, предмети, події, явища і процеси (незалежно від форми їх подання), які використовуються з метою отримання знань і практичних рішень. Найбільш важливими в практичному плані властивостями інформації є: цінність, достовірність, своєчасність.

Цінність інформації визначається забезпеченням можливості досягнення мети, поставленої перед одержувачем, тобто цінність інформації визначається ступенем її корисності для власника (надає певні переваги).

Властивості Інформації

- ***Конфіденційність***
  - лише уповноважені користувачі можуть ознайомитись з інформацією
- ***Цілісність***
  - лише уповноважені користувачі можуть модифікувати інформацію
- ***Доступність***
  - уповноважені користувачі можуть отримати доступ до інформації, не очікуючи довше за заданий (малий) час

Справжність, своєчасність

Справжньою або ***достовірною інформацією*** є інформація, яка з достатньою для власника точністю відображає об'єкти і процеси навколишнього світу.

Своєчасність відображає відповідність цінності і достовірності інформації певного часового періоду.

Чутлива та конфіденційна інформація

Чутлива інформація (sensitive information) – інформація, яка є предметом власності і підлягає захисту відповідно до вимог правових документів або вимогами, встановленими власником інформації.

Конфіденційна інформація – інформація, доступ до якої обмежується відповідно до законодавства або вимог власника.

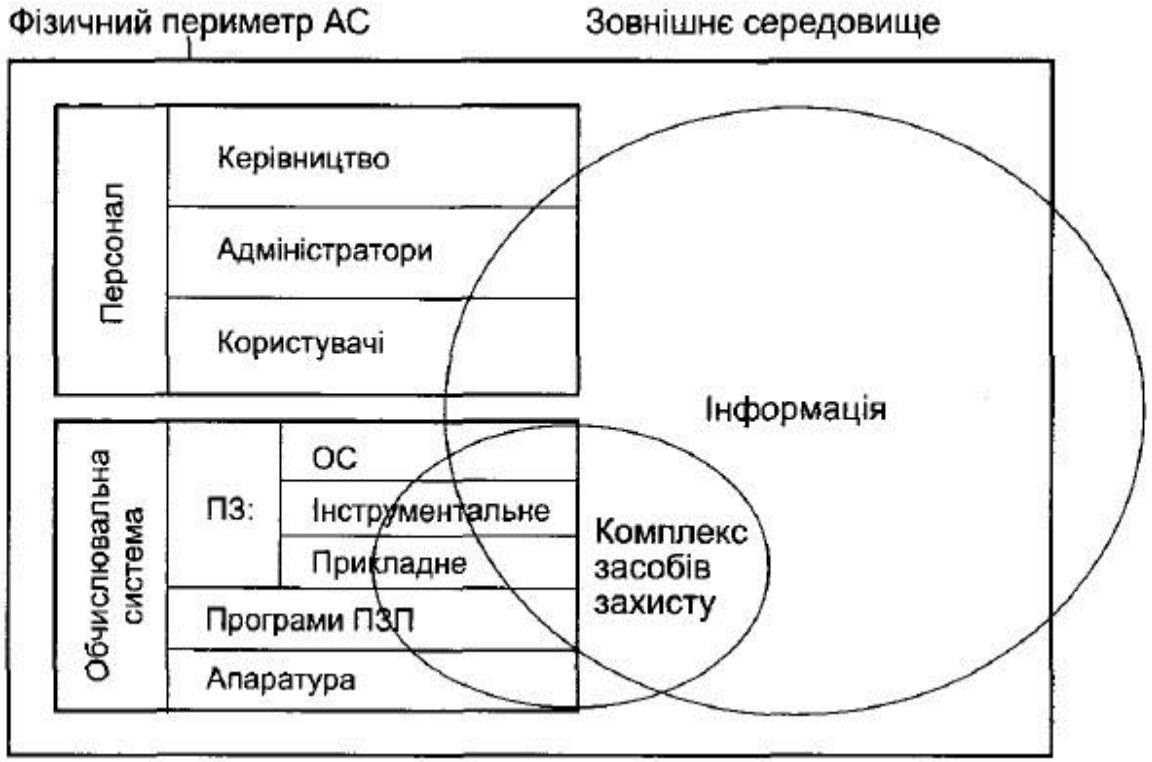
Інформаційний ресурс

Під ***інформаційними ресурсами*** будемо розуміти всю накопичену інформацію про навколишньої дійсності, зафіксовану на матеріальних носіях і в будь-який інший формі, що забезпечує її передачу в часі і просторі між різними споживачами для вирішення наукових, виробничих, управлінських та інших завдань.

Особливо слід виділити положення про те, що ***ресурсом*** є вся ***накопичена інформація***, в тому числі, і інформація недостовірна, представлена сумнівними фактами, помилковими положеннями, неефективними підходами, а також застаріла інформація і явна «дезінформація», що надійшла в інформаційні потоки.

Автоматизована система (АС)

- це організаційно-технічна система, що реалізує інформаційну технологію і поєднує в собі:
  - ✓ Обчислювальну систему
  - ✓ Фізичне середовище
  - ✓ Персонал
  - ✓ Інформацію



Інформаційно-комунікаційна система (ІКС)

До ІКС відносяться:

- Інформаційна система (ІС) - організаційно-технічна система, в якій реалізується технологія обробки інформації з використанням технічних і програмних засобів
- інформаційно-комунікаційна система (ІКС) - сукупність інформаційних та електронних комунікаційних систем, які у процесі обробки інформації діють як єдине ціле

Визначення наведено відповідно до ЗУ «Про захист інформації в інформаційно-комунікаційних системах»

Захищена інформаційна система

Захищена інформаційна система – система, призначена для обробки інформації, що захищається з необхідним рівнем її захищеності.

Інформаційна система називається ***безпечною***, якщо вона, використовуючи відповідні апаратні і програмні засоби, управляє доступом до інформації так, що тільки належним чином авторизовані особи або ж ті, що діють від їхнього імені, процеси отримують право читати, писати, створювати і видаляти інформацію. Очевидно, що абсолютно безпечних систем немає, і тут мова йде про надійну систему в сенсі «система, якій можна довіряти».

Надійна інформаційна система

Інформаційна система вважається ***надійною (інакше довіреною),*** якщо вона з використанням достатніх апаратних і програмних засобів забезпечує одночасну обробку інформації різного ступеня секретності групою користувачів без порушення прав доступу.

Основні критерії оцінки надійності

Політикою безпеки (security policy) називають якісний опис комплексу організаційно-технологічних та програмно-технічних заходів щодо забезпечення захисту даних в ІКС, який включає в себе аналіз можливих загроз і вибір відповідних заходів протидії.

Гарантованість, будучи пасивним елементом захисту, відображає міру довіри, яка може бути надана архітектурі і реалізації системи (іншими словами, показує, наскільки коректно обрані механізми, що забезпечують безпеку системи).

Безпека інформації

- стан інформації, в якому забезпечується збереження властивостей інформації, визначених політикою безпеки

Відповідно до ЗУ «Про засади інформаційної безпеки України» термін «інформаційна безпека» наступний:

- стан захищеності життєво важливих інтересів людини і громадянина, суспільства і держави, при якому запобігається завдання шкоди через неповноту, несвоєчасність та недостовірність поширюваної інформації, порушення цілісності та доступності інформації, несанкціонований обіг інформації з обмеженим доступом, а також через негативний інформаційно-психологічний вплив та умисне спричинення негативних наслідків застосування інформаційних технологій

Загроза, атака, вразливість, порушник

❑***Загроза***

- *будь-які обставини або події, що можуть бути причиною порушення політики безпеки інформації та (або) нанесення збитків ІКС*

❑ ***Атака***

- *спроба реалізації загрози*

❑ ***Вразливість системи***

- *нездатність системи протистояти реалізації певної загрози або сукупності загроз*

❑ ***Порушник, зловмисник***

- *фізична особа, яка порушує політику безпеки (навмисно або ні)*

Моделі

- ***Модель [політики] безпеки***
  - Абстрактний формалізований або неформалізований опис політики безпеки

- ***Модель загроз***
  - Абстрактний формалізований або неформалізований опис методів і засобів здійснення загроз

- ***Модель порушника***
  - Абстрактний формалізований або неформалізований опис порушника

Комплекс засобів захисту (КЗЗ)

Під ***КЗЗ*** розуміють сукупність програмно-апаратних засобів, що забезпечують реалізацію політики безпеки інформації. КЗЗ є складовою обчислювальної системи (див. рис. на сл. 3).

Об'єкти системи (ОС)

ОС – це елемент ресурсів обчислювальної системи, який знаходиться під керуванням КЗЗ і характеризується визначеними атрибутами й поводженням.

Розрізняють наступні види об'єктів:

- ✓Пасивні
- ✓Об'єкти-користувачі
- ✓Об'єкти-процеси

Модель безпеки

Формальний (математичний, алгоритмічний, схемо-технічний) вираз і формулювання політики безпеки називають ***моделлю безпеки (security model)***.

В основі більшості моделей безпеки лежить суб'єктно-об'єктна модель комп'ютерних систем, в тому числі і баз даних як ядра автоматизованих інформаційних систем.

Виділяються **суб'єкти** (subjects) бази даних (активні сутності), **об'єкти** (objects) бази даних (пасивні сутності) і породжувані діями суб'єктів **процеси** над об'єктами.

Доступ

Доступ – взаємодія двох об'єктів обчислювальної системи, коли один із них (той, що здійснює доступ) виконує дії над іншим (тим, до якого здійснюється доступ). Результатом доступу є змінення стану системи або утворення інформаційного потоку.

Правила розмежування доступу – складова політики безпеки, що регламентує правила доступу користувачів і процесів до пасивних об'єктів.

Суб'єкт та об'єкт доступу

Суб'єктом доступу (access subject) називається активний компонент системи, який може стати причиною ініціалізації потоку інформації або зміни стану системи.

Суб'єктом є особа або процес, дії якого регламентуються правилами розмежування доступу.

Об'єкт доступу (access object) – пасивний компонент системи, який зберігає, приймає або передає інформацію, доступ до якої регламентується правилами розмежування доступу.

Об'єктами доступу в БД є практично все, що містить кінцеву інформацію: таблиці (базові або віртуальні), представлення, а також більш дрібні елементи даних: стовпці і рядки таблиць, і навіть окремі поля.

Правила розмежування доступу

Правила розмежування доступу (security policy) – сукупність правил, що регламентують права суб'єктів доступу до об'єктів доступу.

Доступ до інформації поділяють на ***санкціонований*** і ***несанкціонований***:

санкціонований доступ (authorized access to information) – доступ до інформації, яка не порушує встановлених правил розмежування доступу, а ***несанкціонований доступ*** (unauthorized access to information) – доступ до інформації, який порушує встановлені правила розмежування.

Контроль доступу

Контроль доступу обов'язково включає ***ідентифікацію*** (персоналізацію) і ***автентифікацію*** всіх суб'єктів і їх процесів і розмежування повноважень суб'єктів по відношенню до об'єктів з подальшою обов'язковою перевіркою введених повноважень.

Розпізнавання об'єктів

Ідентифікація – процес розпізнавання об'єктів за їх мітками або ідентифікаторами.

Ідентифікатор – унікальний атрибут об'єкта, який дозволяє вирізнити його з поміж інших об'єктів.

Ідентифікація – процес присвоєння ідентифікаторів об'єктам.

Автентифікація, авторизація

Автентифікація – перевірка запропонованого ідентифікатора на відповідність об'єкту, пересвідчення в його справжності.

Авторизація – процедура надання користувачу визначених повноважень у системі.

Відмови

Відмова від авторства – заперечення причетності до створення або передавання якого-небудь документа чи повідомлення.

Відмова від одержання – заперечення причетності до отримання якого-небудь документа чи повідомлення.

Адекватність

Адекватність – показник реального гарантованого рівня безпеки, що відображає ступінь ефективності та надійності реалізованих засобів захисту та їхніх відповідностей поставленим задачам.

Три аспекти

Для баз даних і систем, заснованих на зберіганні даних, важливі ***три основних аспекти інформаційної безпеки*** (інакше базові властивості інформації, яка захищається):

- ***конфіденційність*** (потрібен захист від несанкціонованого доступу),
- ***цілісність*** (потрібен захист від втрати узгодженості та достовірності даних)
- ***доступність*** (потрібен захист від несанкціонованого утримання інформації і ресурсів, підтримка працездатності та захист від руйнування), т. е. під захистом даних розуміють систему заходів, що оберігають дані від несанкціонованого використання, руйнування або спотворення.

Захист даних

Під ***захистом даних*** розуміють систему заходів, що оберігають дані від несанкціонованого використання, руйнування або спотворення.

Необхідно вважати, що ***абсолютний захист даних*** практично не можна реалізувати, тому зазвичай задовольняються відносним захистом інформації - гарантовано захищають її на той період часу, поки несанкціонований доступ до неї, втрата її цілісності або доступності тягне будь-які наслідки.

Типові ситуації

Типові ситуації, в яких дані, які зберігаються можуть бути викрадені, спотворені або загублені:

- ***Несанкціонований доступ*** (НСД)

Якщо будь-хто може отримати доступ до будь-яких збережених даних і внести в них будь-які зміни, то дані можуть бути зруйновані некомпетентним або зловмисним користувачем або використані з утиском прав власника чи на шкоду йому.

- ***Некерований паралелізм***

Як правило, в багатокористувальницькій системі з базою даних одночасно працює кілька користувачів. Деякі з них можуть намагатися одночасно змінювати стан БД. Якщо вони будуть робити це незалежно, то результуючий стан БД може виявитися неузгодженим.

Типові ситуації (продовження)

❑***Локальний збій***

В процесі виконання прикладної програми може виникнути аварійна ситуація (наприклад, ділення на нуль), в результаті якої виконання програми буде припинено. Якщо прикладна програма виконувала оновлення даних, то БД може виявитися в неузгоджену стані.

❑***Втрата оперативної пам'яті (м'який збій)***

Незважаючи на високу надійність ПК, в будь-який момент може статися системний збій (наприклад, відключення живлення), в результаті якого буде втрачено вміст системних буферів і буферів застосунків, розміщених в оперативній пам'яті. Подібні ситуації називаються м'якими збоями системи. Оскільки стан БД в момент м'якого збою непередбачуваний, він може виявитися неузгодженим після перезавантаження системи. На відміну від локального збою при м'якому збої можуть постраждати дані всіх користувачів.

Типові ситуації (продовження)

❑***Фізичне руйнування даних (жорсткий збій)***

В процесі експлуатації пристрою зовнішньої пам'яті може бути зіпсована поверхню диска, блок головок дисковода і т.п. Подібні ситуації називаються жорсткими збоями системи. Імовірність жорсткого збою дуже мала, проте нею не можна нехтувати.

Вимоги захисту даних від руйнування в узагальненому вигляді зводяться до наступного: ніякі випадкові або навмисні руйнування даних не можуть бути незворотними.

Причини типових ситуацій

Перераховані вище ситуації можуть бути, в свою чергу, наслідком:

1. Некомпетентних, безвідповідальних або злочинних дій користувача

2. Неузгоджених оновлень, вироблених одночасно і незалежно користувачами

3. Виникнення необробленої належним чином помилки в програмному забезпеченні

4. Впровадження шкідливого коду в програмне забезпечення

5. Аварійного відключення енергії, збою або відмови різних пристроїв системи, порушення ліній зв'язку

6. Фізичного знищення комп'ютерної системи або її частин

Вимоги до системи

Отже, для захисту даних від втрати цілісності, руйнування і несанкціонованого використання система повинна забезпечувати:

1. Підтримку інформаційної політики підприємства, обмежень прав і повноважень доступу користувачів до даних

2. Захист від шкідливих дій користувачів, спрямованих на пошкодження даних або отримання конфіденційних даних на основі аналізу доступної інформації

3. Дисципліну паралельної роботи багатьох користувачів, що виключає можливість внесення неузгоджених змін до стан БД

4. Відновлення цілісного стану БД після локальних збоїв і м'яких збоїв системи

5. Відновлення БД після жорстких збоїв

Захист інформації в ІКС

Під захистом інформації в ІКС розуміють діяльність, спрямовану на забезпечення безпеки оброблюваної в ІКС інформації та ІКС в цілому, що дає змогу запобігти реалізації загроз або унеможливити її, та зменшити ймовірність завдання збитків від реалізації загроз.

Захист інформації в ІКС полягає у створенні системи технічних (інженерних, програмно-апаратних) та нетехнічних (правових, організаційних) заходів та в підтримці її працездатності.

Система таких заходів – комплексна система захисту інформації (КСЗІ)!

Засоби захисту

Для забезпечення безпеки даних застосовуються комп'ютерні і некомп'ютерні засоби захисту.

❑***Авторизація користувачів і розмежування доступу***.

Авторизація користувачів, що означає надання певних прав користувачеві, що дозволяють їх власнику мати законний доступ до всієї системи або до її окремих об'єктах може бути здійснена як засобами операційної системи, так і засобами самої СУБД.

Комп'ютерні засоби захисту

- ***Доступ до даних через представлення***

Представлення – це результат однієї або декількох реляційних операцій з базовими відносинами з метою створення деякого іншого вигляду даних.

Представлення є віртуальним знімком даних, якого реально в БД не існує, але яке створюється на вимогу окремого користувача в момент надходження цієї вимоги.

Комп'ютерні засоби захисту (продовження)

- ***Оновлення даних та реалізація правил «бізнес логіки» за допомогою збережених процедур, функцій і тригерів.***

Збережені процедури, функції і тригери є фрагментами коду на високорівневій мові програмування, являють собою розширення мови SQL, що зберігаються і виконуються на сервері.

Реалізація всіх правил «бізнес логіки» за допомогою збережених процедур, функцій і тригерів надає значно вищий рівень захисту даних, ніж їх реалізація в клієнтських програмах.

Комп'ютерні засоби захисту (продовження)

- ***Створення резервних копій та відновлення***

Резервне копіювання – періодично виконувана процедура отримання копії БД і її файлу журналу (а також, можливо програм СУБД) у зовнішній постійній пам'яті, найчастіше на окремому носії.

У разі відмови, в результаті якого БД стає непридатною для подальшої експлуатації, резервна копія і зафіксована в файлі журналу оперативна інформація використовуються для відновлення БД до останнього узгодженого стану.

Комп'ютерні засоби захисту (продовження)

❑***Ведення системного журналу***

- це процедура створення і обслуговування файлу журналу, що містить відомості про всі зміни, внесені в БД з моменту створення останньої резервної копії, і призначеного для забезпечення ефективного відновлення системи в разі її відмови.

Якщо в системі, яка відмовила, функція ведення системного журналу не використовувалася, базу даних можна відновити лише до того стану, який було зафіксовано в останній створеній резервній копії. Всі зміни, які були внесені в БД після створення останньої резервної копії, будуть втраченими.

Комп'ютерні засоби захисту (продовження)

- ***Забезпечення цілісності***

Забезпечення цілісності включає перевірку цілісності даних і їх відновлення в разі виявлення суперечностей в БД.

Цілісний стан БД описується за допомогою обмежувачів цілісності у вигляді умов, яким повинні задовольняти збережені в базі дані.

- ***Управління паралелізмом виконання транзакцій***

Управління транзакціями є одним з найбільш важливих завдань СУБД. Розвинені СУБД розраховані на роботу багатьох користувачів, мають достатньо досконалі засоби управління паралелізмом виконання транзакцій, що дозволяє безлічі користувачів мати одночасний доступ до даних.

Комп'ютерні засоби захисту (продовження)

- ***Аудит (реєстрація всіх звернень до інформації, що захищається)***

Для визначення активності використання БД аналізуються її файли журналу. Ці ж джерела можуть бути використані для виявлення будь-яких незвичайних дій в системі.

Регулярне проведення аудиторських перевірок, доповнене постійним контролем вмісту файлів журналу з метою виявлення аномальної активності в системі, дуже часто дозволяє своєчасно виявити і припинити будь-які спроби порушення захисту.

Комп'ютерні засоби захисту (продовження)

- ***Шифрування***

Надійний (на певний час) захист конфіденційних даних можливий лише за допомогою шифрування даних.

Шифрування – це кодування даних за допомогою спеціального алгоритму, в результаті чого дані стають недоступними для читання.

Для успішного дешифрування даних потрібне знання ключа дешифрування.

Деякі СУБД пропонують вбудовані засоби шифрування даних, що зберігаються, які виконують дешифрування «на льоту» в момент звернення користувача до даних.

Комп'ютерні засоби захисту (продовження)

- ***Допоміжні процедури***

Захист від SQL ін'єкцій, агрегування і інференцій.

Некомп'ютерні засоби захисту (продовження)

- ***Заходи забезпечення безпеки і планування захисту від непередбачених обставин***
- ***Контроль над персоналом, і в цілому, контроль за фізичним доступом***
- ***Захист приміщень та сховищ за допомогою різного роду охоронних систем***
- ***Укладання гарантійних угод і договорів про супровід апаратних засобів і програмного забезпечення систем безпеки, вироблених третьою стороною, що дозволяє на певний термін продовжити впевненість в тому, що ці кошти не стануть причиною реалізації загроз безпеки***

Некомп'ютерні засоби захисту (продовження)

- ***Заходи забезпечення безпеки і планування захисту від непередбачених обставин***
- ***Контроль над персоналом, і в цілому, контроль за фізичним доступом***
- ***Захист приміщень та сховищ за допомогою різного роду охоронних систем***
- ***Укладання гарантійних угод і договорів про супровід апаратних засобів і програмного забезпечення систем безпеки, вироблених третьою стороною, що дозволяє на певний термін продовжити впевненість в тому, що ці кошти не стануть причиною реалізації загроз безпеки***

Структура курсу

1) Архітектура розподілених баз даних та інформаційних систем
2) Архітектура БД Oracle
3) Автентифікація та авторизація в БД Oracle
4) Захист та шифрування даних в БД Oracle
5) Моніторинг та аудит безпеки в БД Oracle
6) Резервне копіювання та відновлення БД Oracle
7) Робота з SQL
8) Мова програмування PL/SQL
9) Розробка програмних застосунків в Oracle APEX

Цикл лабораторних робіт

Курсом дисципліни передбачено 13 лабораторних робіт:

- 4 – встановлення та налаштування Oracle
- 4 – робота з SQL в Oracle
- 3 – робота з PL/SQL в Oracle
- 2 – розробка програмних застосунків на Oracle APEX

Донецький національний технічний університет

Тема 2. Архітектура розподілених баз даних та інформаційних систем. Лекція 2. Архітектура розподілених БД

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Мета лекції
- Основна термінологія
- Архітектура розподілених БД

Мета лекції

- Розглянути основну термінологію БД
- Ознайомитися з архітектурою розподілених БД

Предметна область

Область діяльності, в рамках якої вирішується певна проблема, називається ***предметною областю.***

Сутності, що характеризують предметну область, називаються ***об'єктами.***

Атрибут, запис

- Кожен об'єкт характеризується деяким набором даних ***(атрибутів, властивостей).***
- Унікальність об'єкта визначається значеннями атрибутів.
- Опис об'єкта – набір значень атрибутів (запис).

База даних, СУБД

База даних – це сукупність даних про об'єкти предметної області та зв'язки між ними, призначена для спільного використання і керована централізовано.

Система управління базою даних (СУБД) – сукупність мовних і програмних засобів, призначених для створення, ведення і спільного використання БД багатьма користувачами.

Функції СУБД

Функції СУБД:

- ***інтеграція, накопичення, зберігання даних,***
- ***пошук інформації,***
- ***включення нових даних,***
- ***видалення і зміна застарілих записів.***

Принципи побудови $БД_1$

- ***інтегрованість*** – в БД зберігаються дані для декількох користувачів, хоча жоден користувач не розглядає всі дані, що містяться в БД
- ***централізоване управління*** – всі операції, пов'язані з доступом до БД, виконуються ядром (адміністратором) БД
- ***цілісність*** – адекватність інформації, що зберігається в БД, стану предметної області: у будь-який момент часу дані повинні відповідати властивостям і характеристикам об'єктів
- ***відсутність надмірності*** – стан даних, в якому кожен елемент присутній в БД в єдиному екземплярі

Принципи побудови $БД_2$

- ***несуперечливість*** – змістова відповідність між даними, коли вони не суперечать один одному. Розрізняють два аспекти несуперечливості: змістову відповідність різнотипних даних і ідентичність дублюючих даних
- ***безпека*** – система повинна забезпечувати захист даних та програм, що зберігаються в ній, як від випадкових спотворень і знищення, так і від навмисних несанкціонованих дій користувачів
- ***масовість використання*** – сучасна автоматизована інформаційна система повинна забезпечувати колективний доступ користувачів до БД, при якому вони можуть одночасно і незалежно звертатися до БД

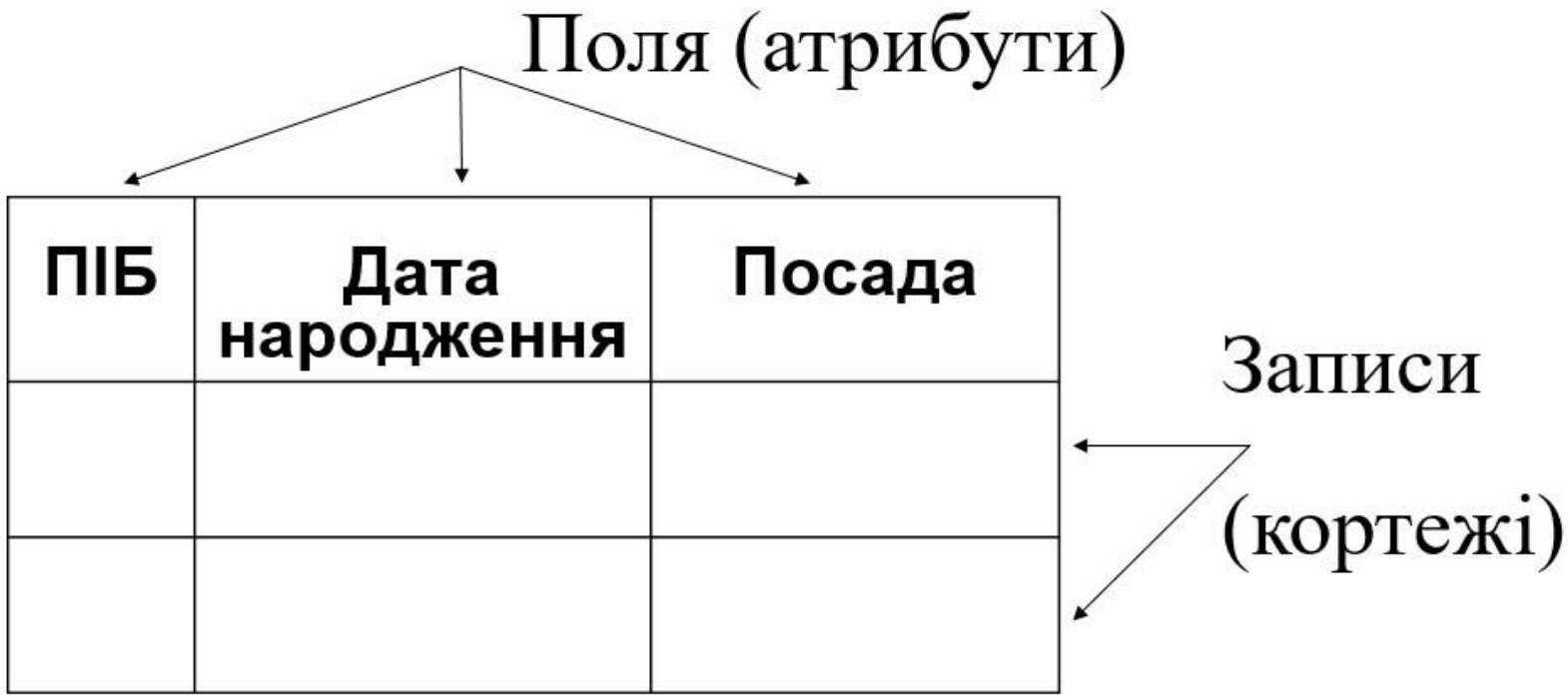
Реляційна модель даних, РМД

- ***Реляційна модель даних (РМД) –*** це модель даних, в якій дані про об'єкти предметної області та інформація про зв'язки між ними представляються однорівневими двовимірними таблицями.
- ***Математична основа РМД –*** теорія множин і відносин (Relations)

Таблиця

У РМД

ВІДНОШЕННЯ = ТАБЛИЦЯ:



Локальні системи

- ***Локальні системи (Ера ПК) (Host-Base Processing)***
  - Всі частини (програми і дані) розміщені в єдиному програмному середовищі – єдина програма на комп'ютері користувача БД
  - Багатокористувацькі системи (Mainframe Multi User Systems)
  - Один користувач (однокористувацькі БД)

Розподілена бази даних, РБД

- ***Розподілена база даних*** – це набір відносин, що зберігаються в різних вузлах комп'ютерної мережі і логічно пов'язаних таким чином, щоб складати єдину сукупність даних
- ***Розподілена база даних*** – це набір логічне пов'язаних між собою даних, які поділяються, і їх описів, які фізично розподілені в деякій комп'ютерній мереж.

РСУБД

- ***Розподілена система управління базою даних (РСУБД) –*** це програмна система, призначена для управління розподіленими базами даних і дозволяє зробити розподіленість інформації прозорою для кінцевого користувача

Структура РБД

- Розподілена система управління базами даних складається з єдиної логічної бази даних, розділеної на кілька фрагментів. Кожен фрагмент бази даних зберігається на одному або декількох комп'ютерах (вузлах, sites), які з'єднані між собою комунікаційною мережею і кожен з яких працює під управлінням окремої СУБД.
- Будь-який користувач може виконати операції над даними на своєму локальному вузлу точно так же, як якщо б цей вузол зовсім не входив в розподілену систему (що створює певну ступінь локальної автономії). З іншого боку, будь-який вузол здатний обробляти дані, що зберігаються на інших комп'ютерах мережі.

Взаємодія з РБД

- Користувачі взаємодіють з розподіленою базою даних через ***застосунки***

не вимагають доступу до даних на інших вузлах

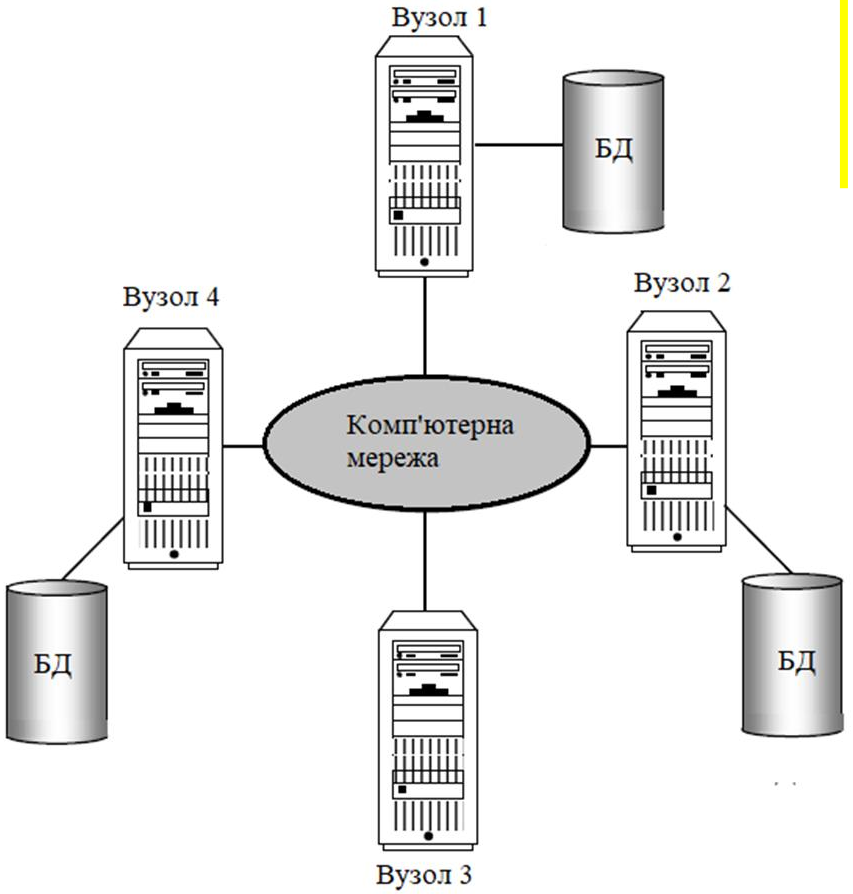
вимагають доступу до даних на інших вузлах

Властивості РБД

- ***У розподіленої СУБД має існувати хоча б один глобальний застосунок, тому будь-яка РСУБД повинна мати такі властивості:***
  - набір логічно пов'язаних даних, які розділяються
  - дані, які зберігаються, розбиті на декілька фрагментів
  - між фрагментами може бути організована реплікація даних
  - фрагменти і їхні репліки розподілені за різними вузлами
  - вузли зв'язані між собою мережевими з'єднаннями
  - робота з даними на кожному вузлу управляється СУБД
  - СУБД на кожному вузлу здатні підтримувати автономну роботу локальних застосунків

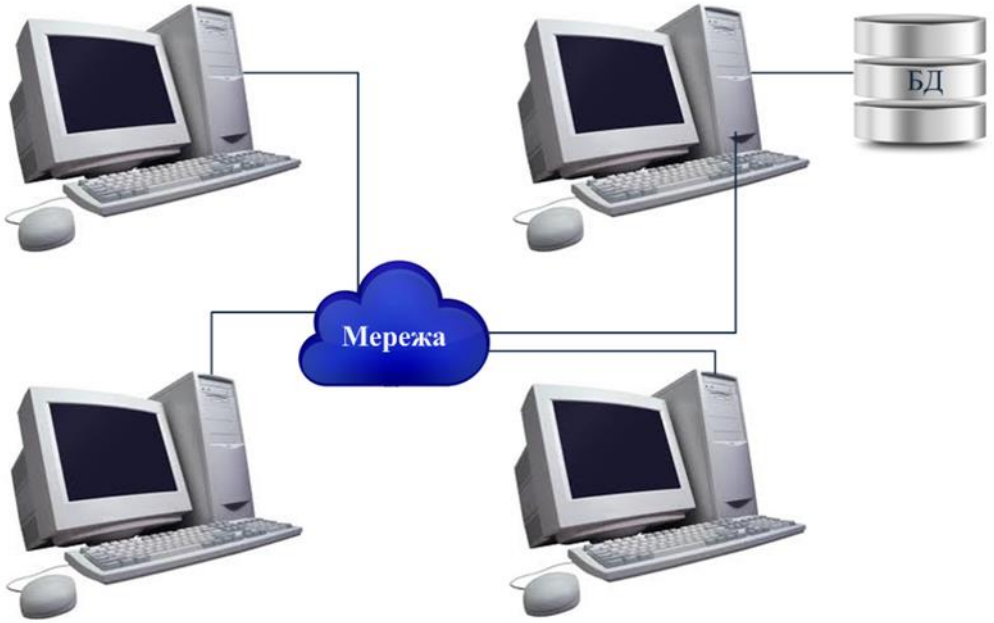
Реплікація

- ***Реплікація*** полягає в підтримці актуальною копії (репліки) деякого фрагмента бази даних на декількох різних вузлах
- Немає необхідності в тому, щоб на кожному з вузлів системи існувала своя власна база даних

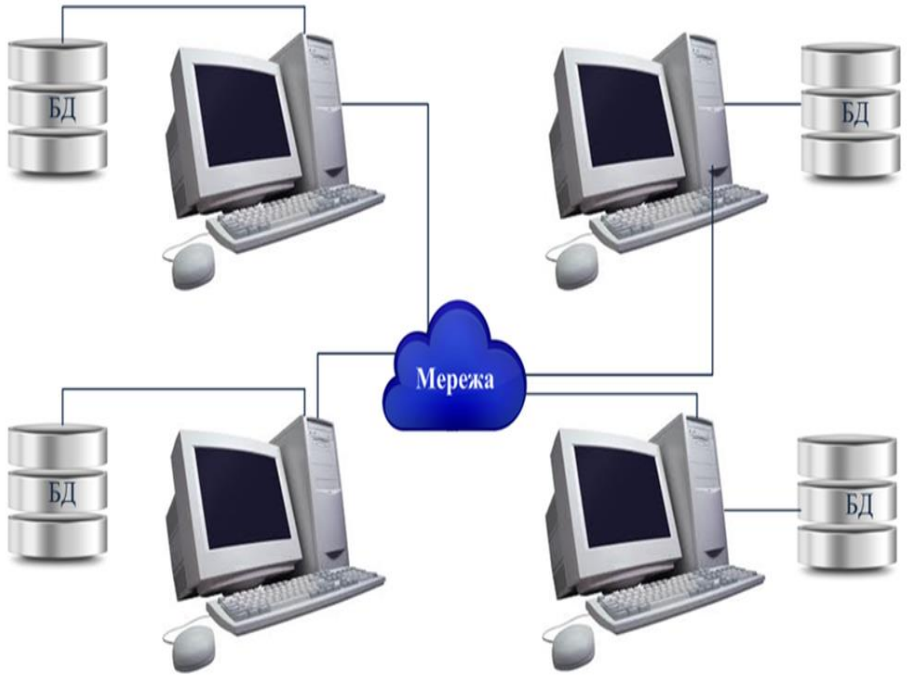


Топологія розподіленої системи управління базою даних

Розподілена СКБД



Централізована БД



Розподілена БД

Кожний(!?) з вузлів мережі містить свою базу даних, однак вони розглядаються як логічна єдина база, а не як сукупність розкиданих у мережі файлів. Усі дані є логічно взаємозалежними.

Розподілена СКБД – це повноцінна СКБД, що виконує всі необхідні функції з керування даними.

Фундаментальний принцип РСУБД

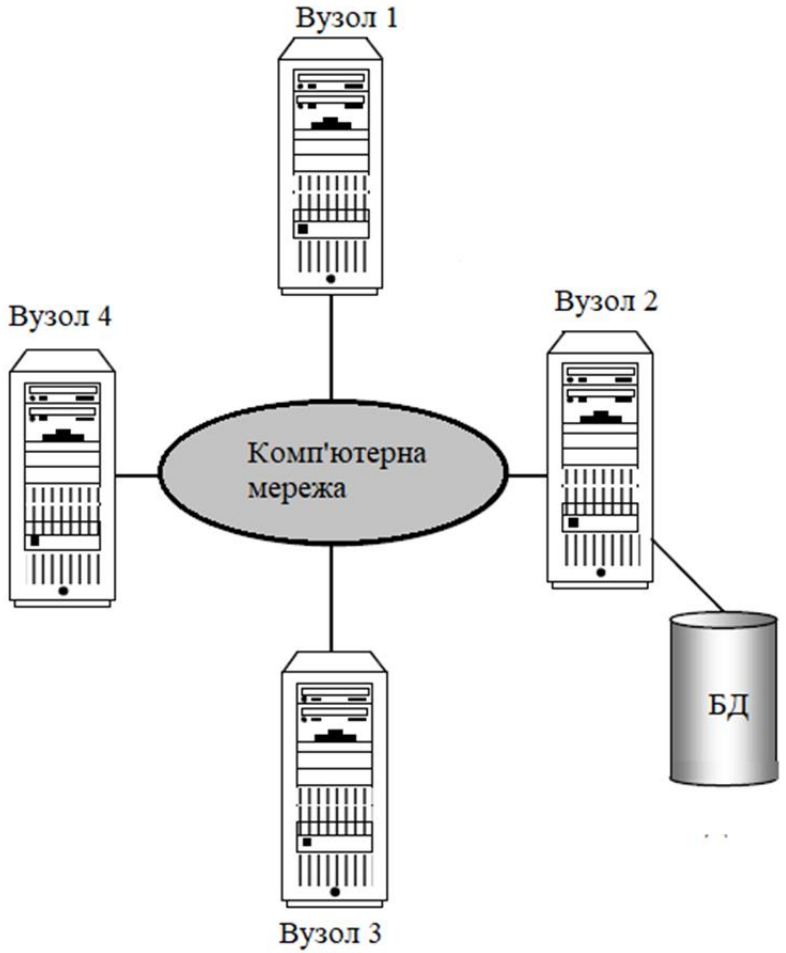
- ***Для кінцевого користувача розподіленість системи повинна бути абсолютно прозора (невидима). Іншими словами, для користувача розподілена система повинна виглядати так само, як нерозподілена система***

Розподілена обробка БД

- ***Розподілена обробка –*** це обробка з використанням централізованої бази даних, доступ до якої може здійснюватися з різних комп'ютерів мережі
- ***Основоположним моментом*** у визначенні розподіленої бази даних є твердження, що система працює з даними, фізично розподіленими в мережі

Клієнт-сервер

- якщо дані зберігаються ***централізовано***, то навіть в 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***
- ця топологія часто називається системою ***«клієнт / сервер»***

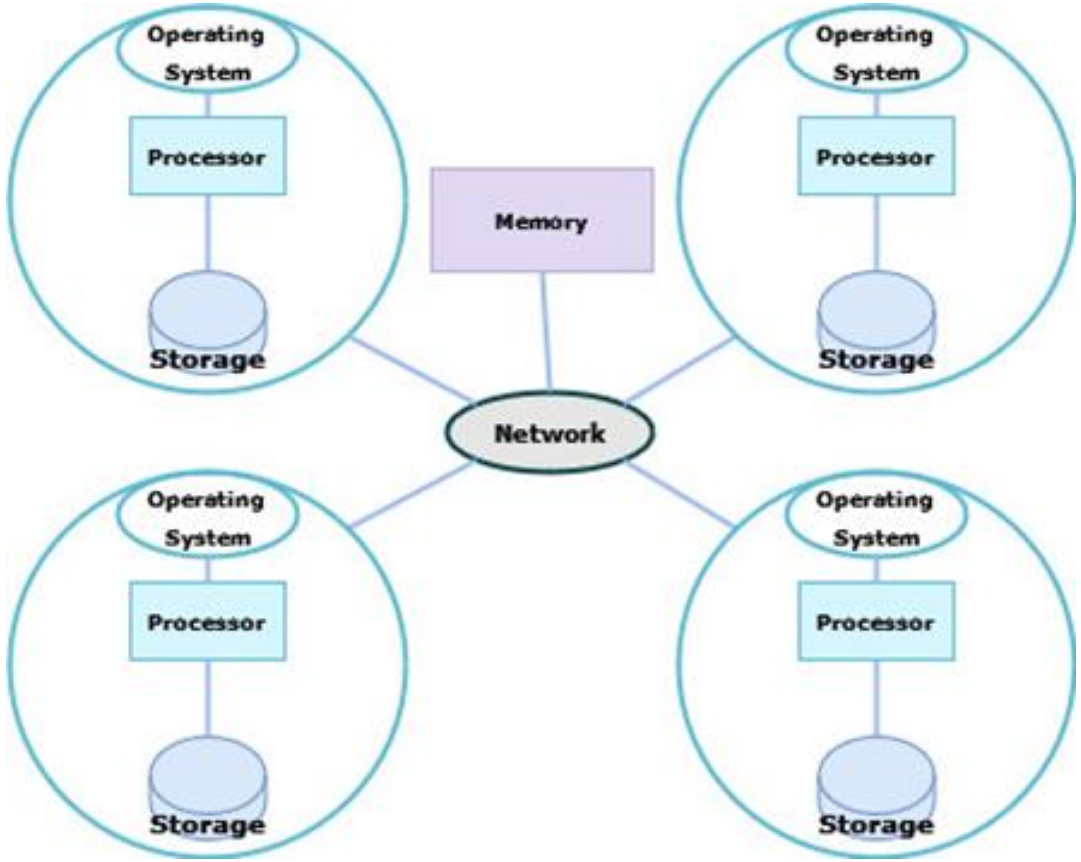


Топологія системи з розподіленою обробкою БД

Паралельна СУБД

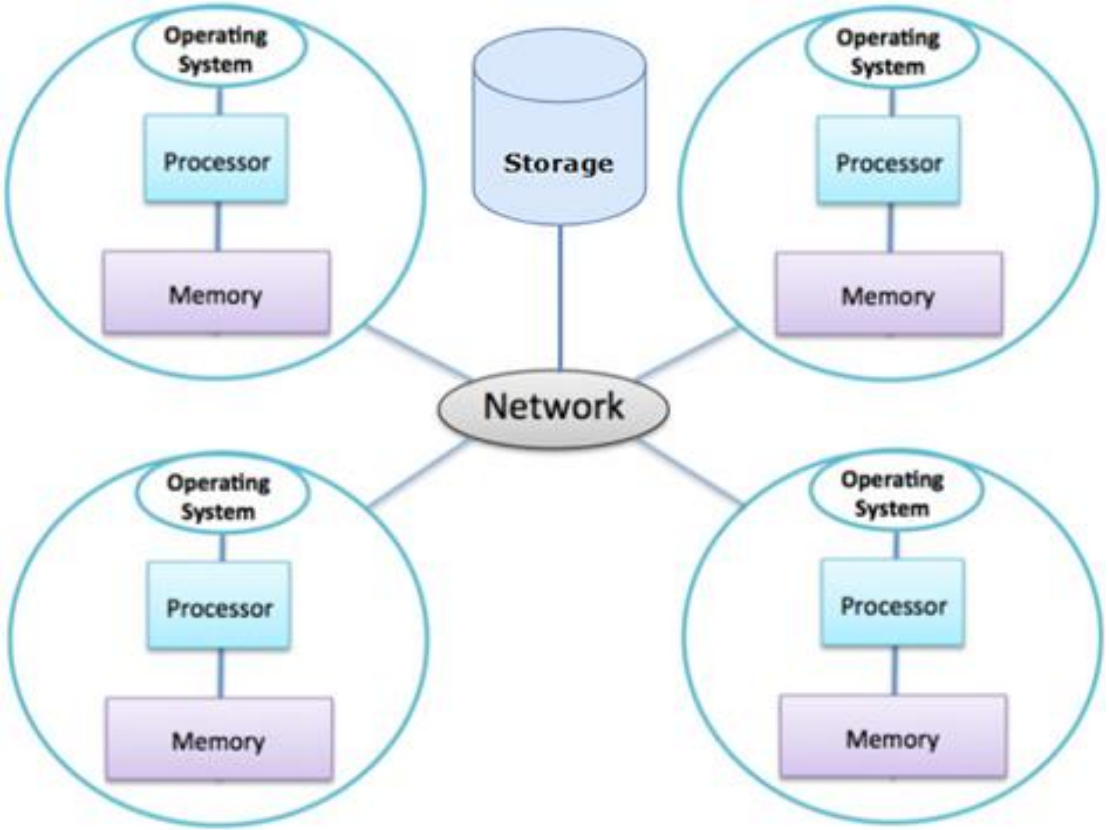
- ***Паралельна СУБД –*** це система управління базою даних, яка функціонує з використанням декількох процесорів і пристроїв жорстких дисків, що дозволяє їй розпаралелювати виконання деяких операцій з метою підвищення загальної продуктивності обробки

Типи архітектури паралельних СУБД$_1$



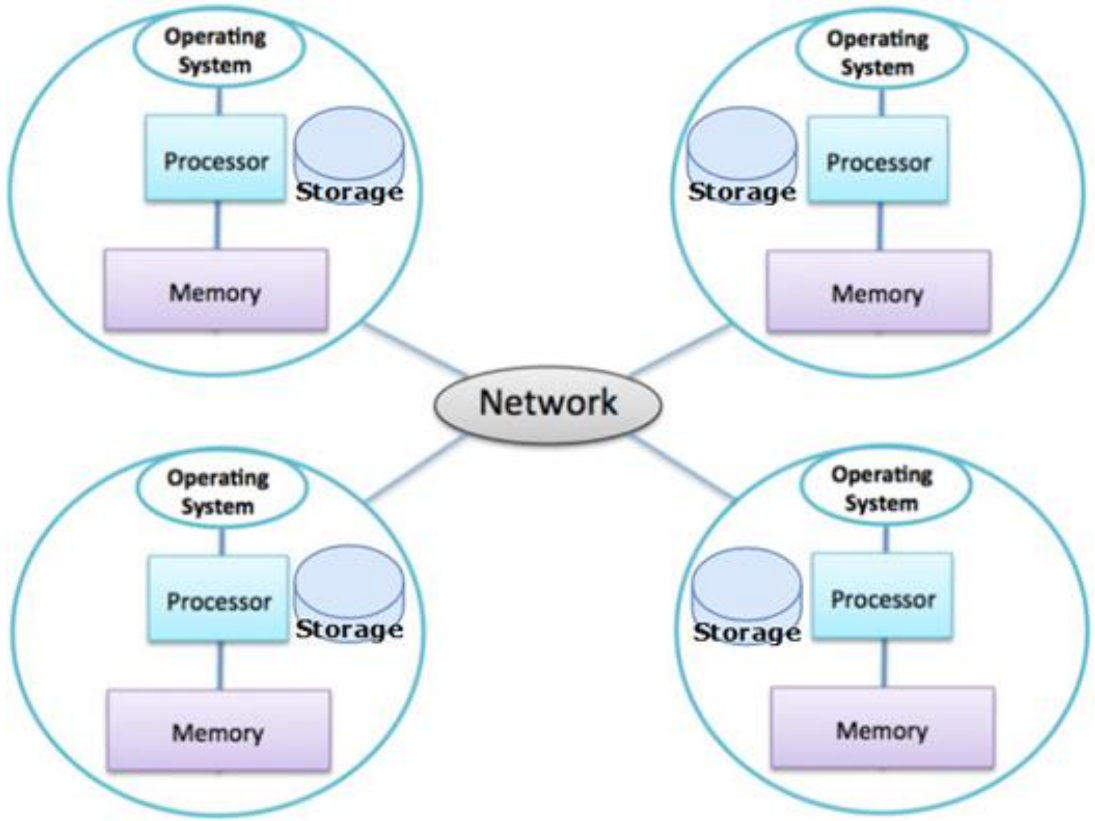
Системи з розподілом пам'яті

Типи архітектури паралельних СУБД$_2$



Системи з розподілом дисків

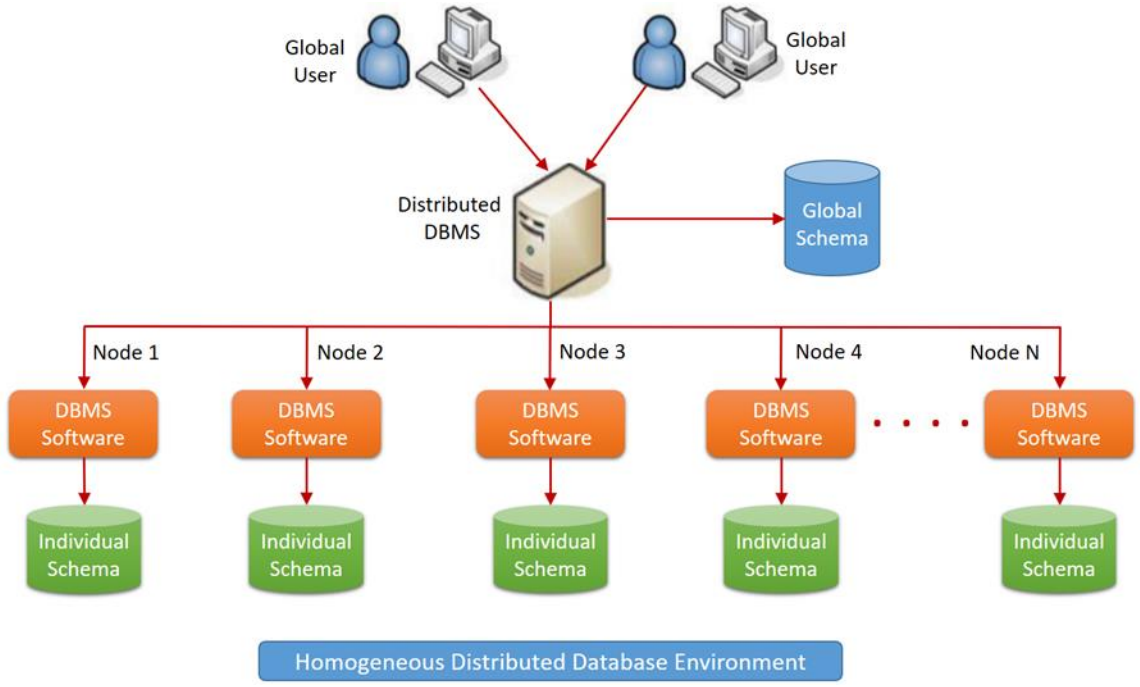
Типи архітектури паралельних СУБД$_3$



Системи без поділу

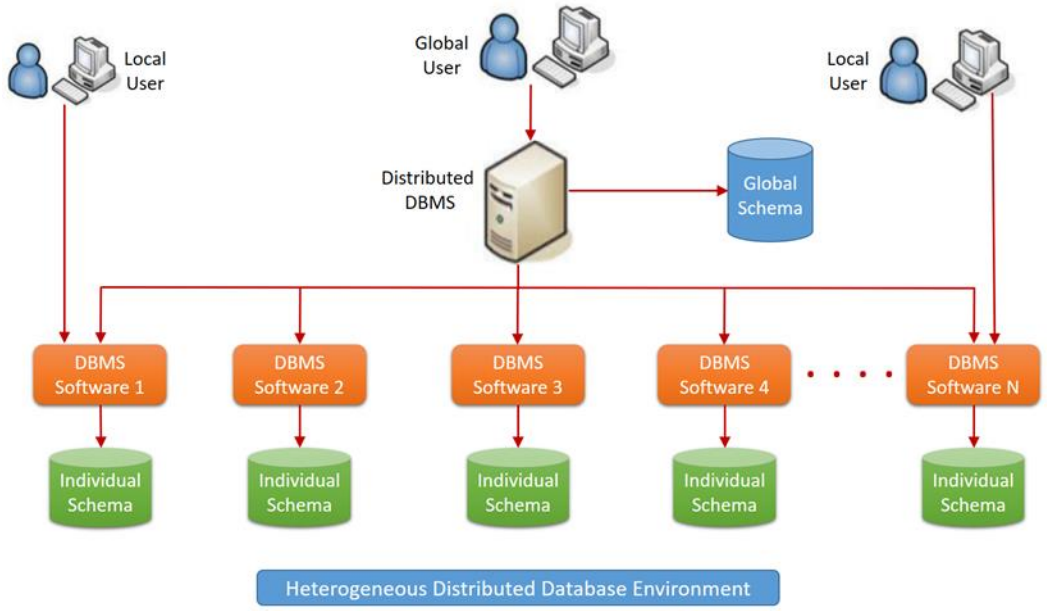
Гомогенні РСУБД

- ***Гомогенні (однорідні)*** **–** всі вузли розподіленої системи використовують один і той же тип СУБД



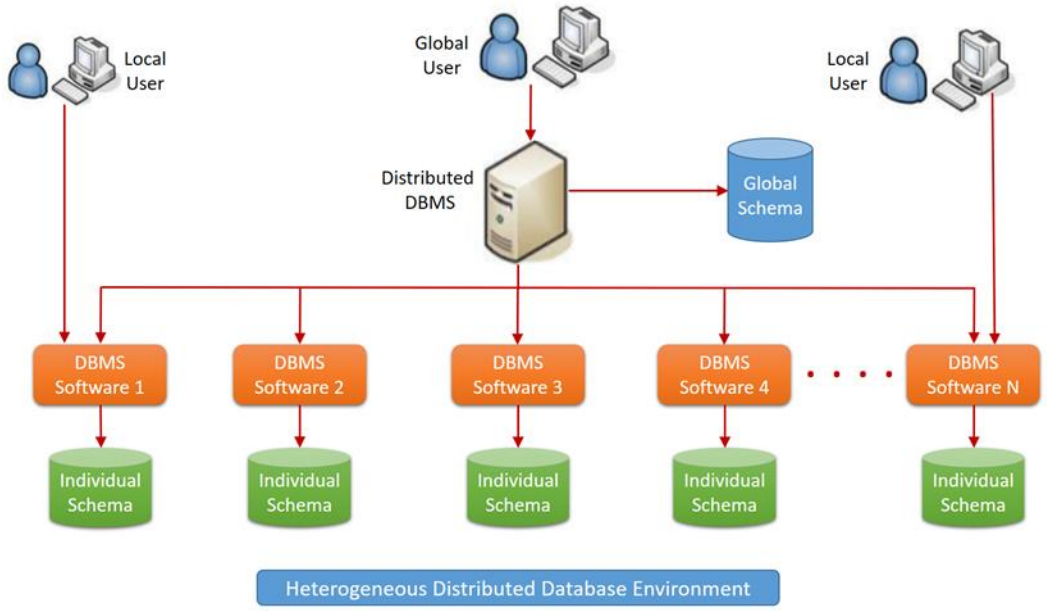
Гетерогенні РСУБД

- ***Гетерогенні (різнорідні)*** **–** на сайтах можуть функціонувати різні типи СУБД, що використовують різні моделі даних (реляційні, мережеві, ієрархічні і ін.)



Мультибазові системи

- ***Мультибазова система*** представляє собою розподілену систему, де керування кожним з вузлів здійснюється автономно.



Переваги та недоліки розподілених СУБД

- Основною причиною використання розподілених баз даних є те, що зазвичай підприємства вже розподілені, принаймні, логічно, тобто на підрозділи, відділи, робочі групи і т.д.
- Великі організації можуть бути розподілені і фізично на відділення, заводи, лабораторії, які можуть перебувати в різних кінцях країни і навіть за її межами.
- Цілком логічне буде припустити, що дані також розподілені, оскільки кожна організаційна одиниця створює і обробляє власні дані, що відносяться до діяльності цієї одиниці.

Переваги розподілених СУБД$_1$

- Інформація підприємства розбивається на частини, які можна назвати островами інформації. Розподілена база даних забезпечує мости для їх 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 отримання доступу до даних, що зберігаються у всіх відділеннях компанії.

Переваги розподілених СУБД$_2$

- Інакше кажучи, розподілена система дозволяє структурі бази даних відображати структуру організації. Це є найбільш важливою перевагою розподілених СУБД.
- У розподілених системах дані розміщуються на тому сайті, на якому зареєстровані користувачі, які їх найчастіше використовують. В результаті користувачі цього вузла отримують локальний контроль над необхідними їм даними і можуть регулювати локальні обмеження на їх використання.
- У цьому полягають ***роздільність і локальна автономність розподілених*** СУБД.

Переваги розподілених СУБД$_3$

- 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 їх копії будуть розміщені на більш ніж одному сайті. Система буде перенаправляти запити до відмовив вузла на адресу іншого сайту. Це призводить до ***підвищення надійності системи і доступності даних***.

Переваги розподілених СУБД$_4$

- В даний час вважається, що набагато дешевше зібрати з невеликих комп'ютерів систему, потужність якої буде еквівалентна потужності одного великого комп'ютера. Виявляється, що набагато вигідніше встановлювати в підрозділах організації власні малопотужні комп'ютери, крім того, набагато дешевше додати в мережу нові робочі станції, ніж модернізувати систему з мейнфреймів. З цього випливають ***економічні переваги*** використання РСУБД.
- Завдяки ***модульності*** розподіленої середовища розширення існуючої системи здійснюється набагато простіше. Додавання в мережу нового вузла не впливає на функціонування вже існуючих. Подібна гнучкість дозволяє організації легко розширюватися. У централізованих СУБД зростання розміру бази даних може вимагати заміни та обчислювальної системи на більш потужну, і використовуваного програмного забезпечення на більш потужну і гнучку СУБД.

Недоліки розподілених СУБД$_1$

- ***Підвищення складності*** (принаймні, з технічної точки зору)

  Розподілені СУБД є більш складними програмними комплексами, ніж централізовані СУБД. Досить вказати на той факт, що дані можуть піддаватися реплікації. Якщо реплікація даних не буде підтримуватися на необхідному рівні, система буде мати більш низький рівень доступності даних, надійності і продуктивності, ніж централізовані системи.

- У розподілених системах можуть виникнути ***проблеми захисту*** не тільки даних, що реплікується на кілька різних сайтів, а й захисту мережевих з'єднань самих по собі. У централізованих системах доступ до даних легко контролюється.

Недоліки розподілених СУБД$_2$

- ***Ускладнення контролю за цілісністю даних***

  Вимоги забезпечення цілісності (правильності та узгодженості 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, що використовуються під час перевірок. У розподілених СУБД підвищена вартість передачі та обробки даних може перешкоджати організації ефективного захисту від порушень цілісності даних

Недоліки розподілених СУБД$_3$

- ***Ускладнення контролю за цілісністю даних***

  Вимоги забезпечення цілісності (правильності та узгодженості 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, що використовуються під час перевірок. У розподілених СУБД підвищена вартість передачі та обробки даних може перешкоджати організації ефективного захисту від порушень цілісності даних

- ***Ускладнення процедури проектування бази даних***

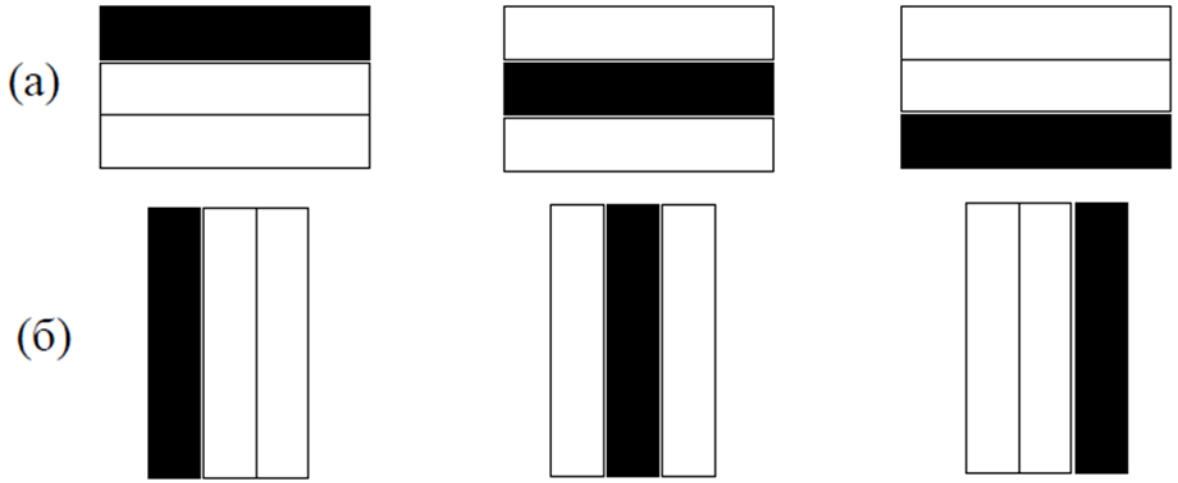
  Крім звичайних проблем, пов'язаних з проектуванням централізованих баз даних, розробка розподілених СУБД вимагає прийняття рішення про фрагментацію даних, розподілу фрагментів по окремих сайтів і організації процедур реплікації даних.

Проектування РБД$_1$

- ***Фрагментація***

  Будь-яке відношення (!) може бути розділене на частини або фрагменти при організації фізичного зберігання цього відношення. Фрагменти розподіляються по різних вузлів.

  Якщо у фрагмент виділяється підмножина кортежів відносин, то він називається ***горизонтальним*** фрагментом (рис. а). Фрагмент називається ***вертикальним*** (рис. б), якщо в ньому використовується підмножина атрибутів відносини.



Проектування РБД$_2$

- ***розподіл***
  - Вузол для зберігання фрагмента вибирається виходячи з деякої оптимальної схеми розміщення фрагментів.
- ***реплікація***
  - Одним із завдань РСУБД – підтримка актуальної копії деякого фрагмента на декількох вузлах.
  - Визначення та розміщення фрагментів проводиться на основі аналізу найбільш важливих застосунків, які визначають особливості використання бази даних.
  - При проектуванні враховуватимуться ***якісні і кількісні показники*** майбутньої системи. Розподіл виконується на основі кількісної інформації, а базою при створенні схеми фрагментації є якісні показники.

Горизонтальна фрагментація$_1$

- ***Горизонтальним*** називається фрагмент, виділений з відношення по горизонталі і що складається з деякої підмножини кортежів цього відношення

Якщо задано відношення $\mathbf{R_1}$, то його горизонтальний фрагмент може бути визначений формулою:

$$\mathbf{R = \sigma_F=(R1)}$$

Тут **F** є предикатом, побудованим з використанням одного або більше атрибутів відносини $\mathbf{R_1}$

Горизонтальна фрагментація$_2$

Приклад

- Нехай інформація про службовців має вигляд відношення **СЛУЖ** (**Табл.**). Воно містить атрибути:
  - табельний номер працівника (**Таб#**),
  - номер відділення компанії (**Відд#**), в якому він працює,
  - прізвище, ім'я, по батькові (**ПІБ**),
  - стать (**Стать**),
  - дата народження (**Нар.**),
  - номер паспорта (**Пасп#**),
  - посаду (**Посада**),
  - оклад (**Оклад**).

Горизонтальна фрагментація$_3$

Приклад

Відношення СЛУЖ (Табл.)

Таб#	Відд#	ПІБ	Стать	Нар.	Пасп#	Посада	Оклад
154	Д1	Іваненко І.І.	Ч	10.11.60	153238	асистент	3200
155	Д1	Петренко П.П.	Ч	13.04.51	473850	менеджер	4000
156	Д2	Стефанюк С.П.	Ж	11.05.69	872134	асистент	2500
157	Д2	Сидорчук С.С.	Ч	15.12.70	876234	менеджер	3000
158	Д3	Медвєдева О.П.	Ж	23.03.68	786349	менеджер	3500

Горизонтальна фрагментація$_4$

Приклад

Нехай відомо, що відділення **Д1** розташоване в Києві, а відділення **Д2** і **Д3** - в Дніпрі. В цьому випадку горизонтальна фрагментація відношення СЛУЖ по атрибуту **Відд#** може бути виконана таким чином:

$$R = \sigma_F(R_1)$$

$$К\_СЛУЖ = \sigma_{Відд\#}\ 'Д1'(СЛУЖ)$$

$$Д\_СЛУЖ = \sigma_{Відд\#}\ 'Д2'(СЛУЖ) \cup \sigma_{Відд\#}\ 'Д3'(СЛУЖ)$$

В результаті будуть створені два фрагмента.

Другий фрагмент з ім'ям **Д_СЛУЖ** буде зберігатися на вузлу в Дніпрі і складатися з кортежів, в яких значення атрибута **Відд#** дорівнює **'Д2'** або **'Д3'**.

Горизонтальна фрагментація$_5$

Приклад

Перший фрагмент, вміст якого представлено нижче, буде складатися з кортежів, в яких значення атрибута **Відд #** дорівнюватиме **'Д1'**. Цей фрагмент має всередині системне ім'я **К_СЛУЖ** і буде зберігатися на вузлу в Києві.

Горизонтальний фрагмент **К_СЛУЖ** відношення **СЛУЖ**.

Таб#	Відд#	ПІБ	Стать	Нар.	Пасп#	Посада	Оклад
154	Д1	Іваненко І.І.	Ч	10.11.60	153238	асистент	3200
155	Д1	Петренко П.П.	Ч	13.04.51	473850	менеджер	4000

Горизонтальна фрагментація$_6$

Приклад

Другий фрагмент з ім'ям **Д_СЛУЖ** буде зберігатися на вузлу в Дніпрі і складатися з кортежів, в яких значення атрибута **Відд#** дорівнює **'Д2'** або **'Д3'**.

Горизонтальний фрагмент **Д_СЛУЖ** відношення **СЛУЖ**.

Таб#	Відд#	ПІБ	Стать	Нар.	Пасп#	Посада	Оклад
156	Д2	Стефанюк С.П.	Ж	11.05.69	872134	асистент	2500
157	Д2	Сидорчук С.С.	Ч	15.12.70	876234	менеджер	3000
158	Д3	Медвєдева О.П.	Ж	23.03.68	786349	менеджер	3500

Горизонтальна фрагментація$_7$

Запропонована схема фрагментації відповідає всім правилам коректності:

- ***Повнота.*** Кожен кортеж вихідного відношення присутній або у фрагменті **К_СЛУЖ**, або у фрагменті **Д_СЛУЖ**.
- ***Відновлюваність***. Відношення **СЛУЖ** може бути відновлено з створених фрагментів за допомогою наступної операції об'єднання:

СЛУЖ = К_СЛУЖ U Д_СЛУЖ

- ***Непересікальність***. Отримані фрагменти не перетинаються, оскільки не існує значення атрибута **Відд#**, яке одночасно дорівнювало б значенням **'Д1'** і **'Д2'** або **'Д3'**.

Горизонтальна фрагментація$_8$

Побудова схеми фрагментації передбачає пошук набору ***мінімальних*** (тобто повних і релевантних) предикатів для розбиття відносини на фрагменти.

Набір предикатів є ***повним*** тоді і тільки тоді, коли ймовірність звернення до будь-яких двох кортежів одного і того ж фрагмента з боку будь-якої програми буде однакова.

Предикат є ***релевантним***, якщо існує, принаймні, один застосунок, який по-різному звертається до виділених за допомогою цього предиката фрагментів.

Вертикальна фрагментація$_1$

- ***Вертикальним*** називається фрагмент, виділений з відносини по вертикалі і складається з підмножини атрибутів цього відносини.
  - При вертикальній фрагментації в різні фрагменти об'єднуються атрибути, використовувані окремими застосунками. Визначення фрагментів в цьому випадку виконується за допомогою операції ***проекції*** реляційної алгебри. Для заданого відношення $\mathbf{R_1}$ вертикальний фрагмент може бути обчислений за допомогою наступної формули:

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

де a_1, ..., a_n є атрибути відношення R_1.

Вертикальна фрагментація$_2$

❑***Будемо використовувати той самий приклад:***

Нехай інформація про службовців має вигляд відношення **СЛУЖ** (**Табл.**). Воно містить атрибути:

- табельний номер працівника (**Таб#**),
- номер відділення компанії (**Відд#**), в якому він працює,
- прізвище, ім'я, по батькові (**ПІБ**),
- стать (**Стать**),
- дата народження (**Нар.**),
- номер паспорта (**Пасп#**),
- посаду (**Посада**),
- оклад (**Оклад**).

Вертикальна фрагментація$_2$

Приклад

Застосунок, який друкує платіжні відомості, для кожного з працівників компанії використовує атрибути:

- табельний номери працівника (**Таб#**),
- посаду (**Посада**),
- стать (**Стать**),
- дата народження (**Нар.**),
- номер паспорта (**Пасп#**),
- оклад (**Оклад**).

Відомість, яка видається для відділу кадрів, містить атрибути:

- табельний номер (**Таб#**),
- прізвище, ім'я, по батькові (**ПІБ**) і
- номер відділення компанії (**Відд#**).

Виходячи з цих відомостей, вертикальна фрагментація відношення **СЛУЖ** може бути виконана за допомогою наступних визначень:

$$C_1 = \pi_{Таб\#,Посада,Стать,Нар,Пасп\#,Оклад}(СЛУЖ),$$

$$C_2 = \pi_{Таб\#,ПІБ,Відд\#}(СЛУЖ).$$

Вертикальна фрагментація$_3$

Приклад

- За допомогою вказаних формул будуть створені два фрагмента C_1 та C_2, вміст яких представлено в наступних таблицях. При цьому обидва фрагмента містять первинний ключ - атрибут ***Таб#*** - що дозволяє при необхідності реконструювати вихідне відношення.
- Перевага вертикальної фрагментації полягає в тому, що окремі фрагменти можуть розміщуватися на тих сайтах, на яких вони використовуються. Це додатково надає позитивний вплив на продуктивність системи, оскільки розміри кожного з фрагментів менше розмірів вихідної таблиці.

Вертикальна фрагментація$_4$

Приклад

Вертикальний фрагмент **C_1** відношення **СЛУЖ**

Таб#	Стать	Нар.	Пасп#	Посада	Оклад
154	Ч	10.11.60	153238	асистент	3200
155	Ч	13.04.51	473850	менеджер	4000
156	Ж	11.05.69	872134	асистент	2500
157	Ч	15.12.70	876234	менеджер	3000
158	Ж	23.03.68	786349	менеджер	3500

Вертикальна фрагментація$_5$

Приклад

Вертикальний фрагмент **C_2** відношення **СЛУЖ**

Таб#	Відд#	ПІБ
154	Д1	Іваненко І.І.
155	Д1	Петренко П.П.
156	Д2	Стефанюк С.П.
157	Д2	Сидорчук С.С.
158	Д3	Медвєдева О.П.

Реплікація. Показники

- До ***кількісних показників*** відносяться наступні:
  - частота запуску застосунку на виконання;
  - вузол, на якому запускається застосунок;
  - вимоги до продуктивності транзакцій і застосунків.
- ***Якісна інформація*** може включати:
  - перелік транзакцій, які виконуються в застосунку;
  - використовувані в цих транзакціях відносини, атрибути і кортежі;
  - тип доступу до цих об'єктів (читання або запис);
  - предикати, які використовуються в операціях читання.

Цілі поділу на фрагменти$_1$

- ***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 буде існувати 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 перевантажений запитами з боку інших вузлів, що призведе до зниження продуктивності всієї системи. З іншого боку, деякі вузли будуть «простоювати», тобто ресурси системи при неправильному розподілі будуть використовуватися неефективно

Цілі поділу на фрагменти$_2$

- ***Мінімізація витрат на передачу даних***
  - Вартість виконання в системі віддалених запитів – одна з найважливіших характеристик системи. Витрати на вибірку будуть мінімальні при забезпеченні максимальної локальності посилань, тобто коли кожен вузол буде мати свою власну копію даних. Однак при оновленні реплікованих даних внесені зміни доведеться поширювати на всі вузли з репліками, що викличе збільшення витрат на передачу даних
- ***Прийнятне співвідношення між вартістю і ємкістю зовнішньої пам'яті***
  - На всіх вузлах зазвичай рекомендується використовувати більш дешеві накопичувачі. Ця вимога має бути погоджено з вимогою підтримки локальності посилань

Роздільне (фрагментоване) розміщення

- локальність посилань – висока
- надійність і доступність – низька для окремих елементів, висока для системи в цілому
- продуктивність – задовільна
- вартість пристроїв зберігання – найнижча
- витрати на передачу даних – низькі

Розміщення з повною реплікацією

- 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 вибіркової реплікацією

- 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 – низька

Донецький національний технічний університет

Тема 3. Архітектура БД Oracle
Лекція 3. Архітектура БД Oracle. Фонові процеси

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Мета лекції
- Архітектура БД Oracle
- Основні фонові процеси Oracle

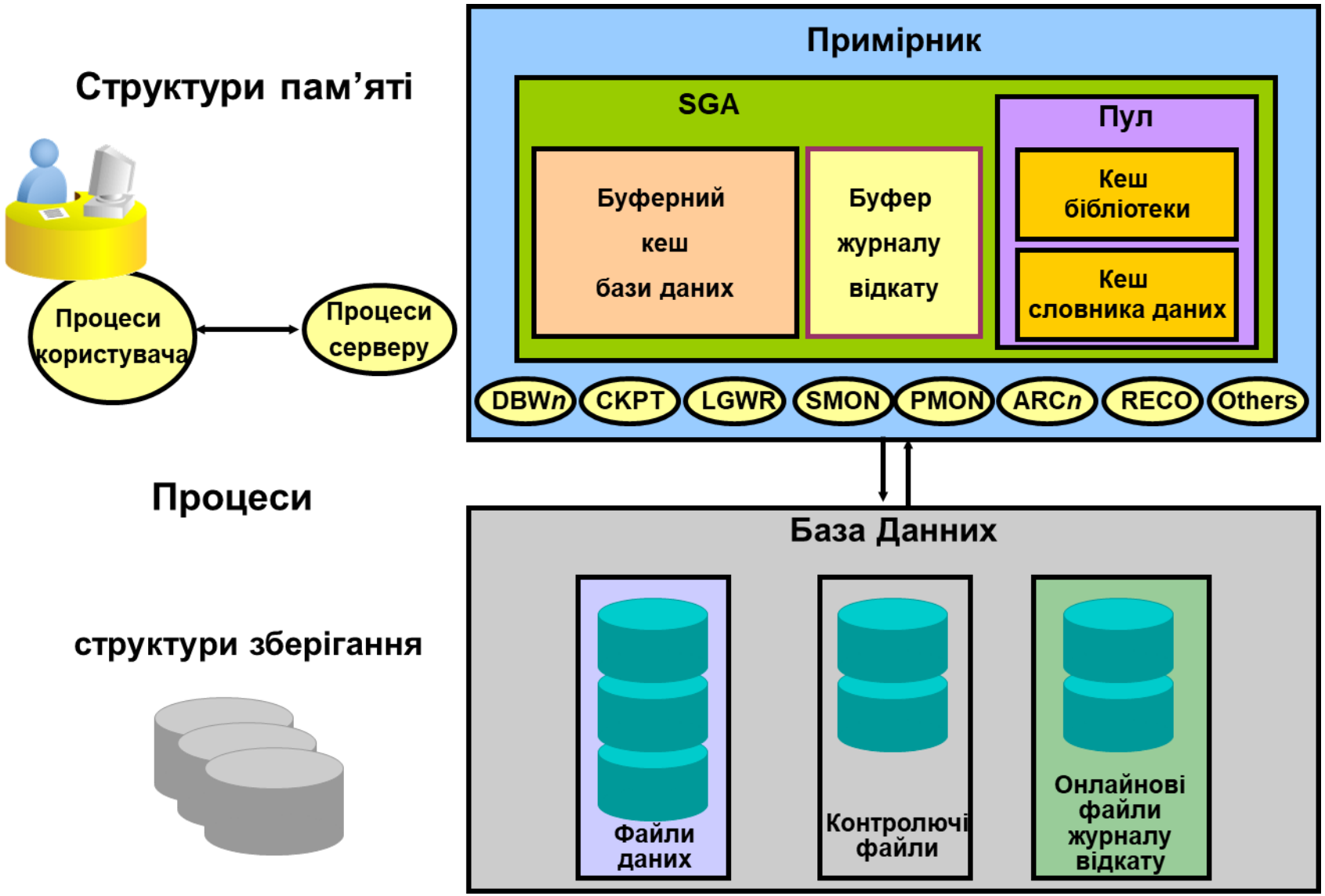
Мета лекції

- Ознайомитися з архітектурою БД Oracle
- Розглянути основні фонові процеси

Архітектурні компоненти бази даних Oracle(огляд)

Система управління реляційними базами даних Oracle (RDBMS) є системою управління базами даних, яка забезпечує відкритий, всебічний, комплексний підхід для управління інформацією.

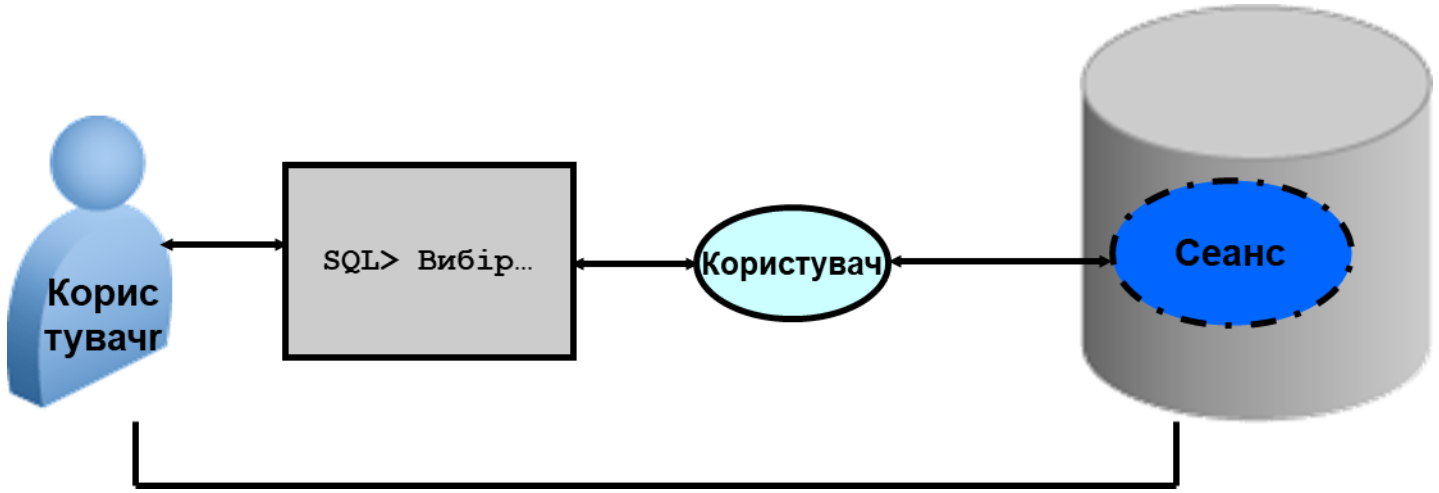
Структури сервера бази даних Oracle



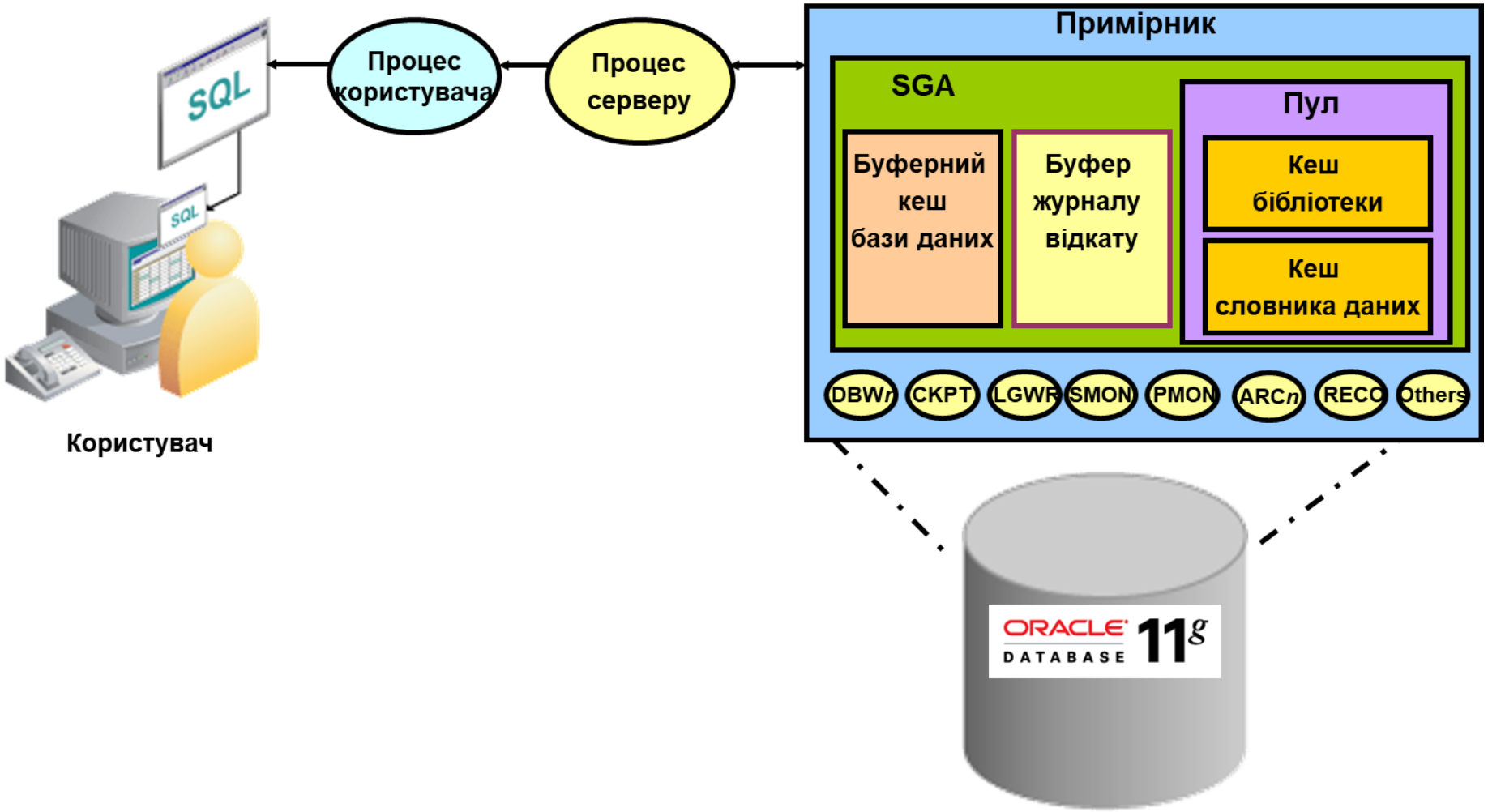
З’єднання з базою даних

З'єднання – комунікаційна траса між процесом користувача і інстансом бази даних

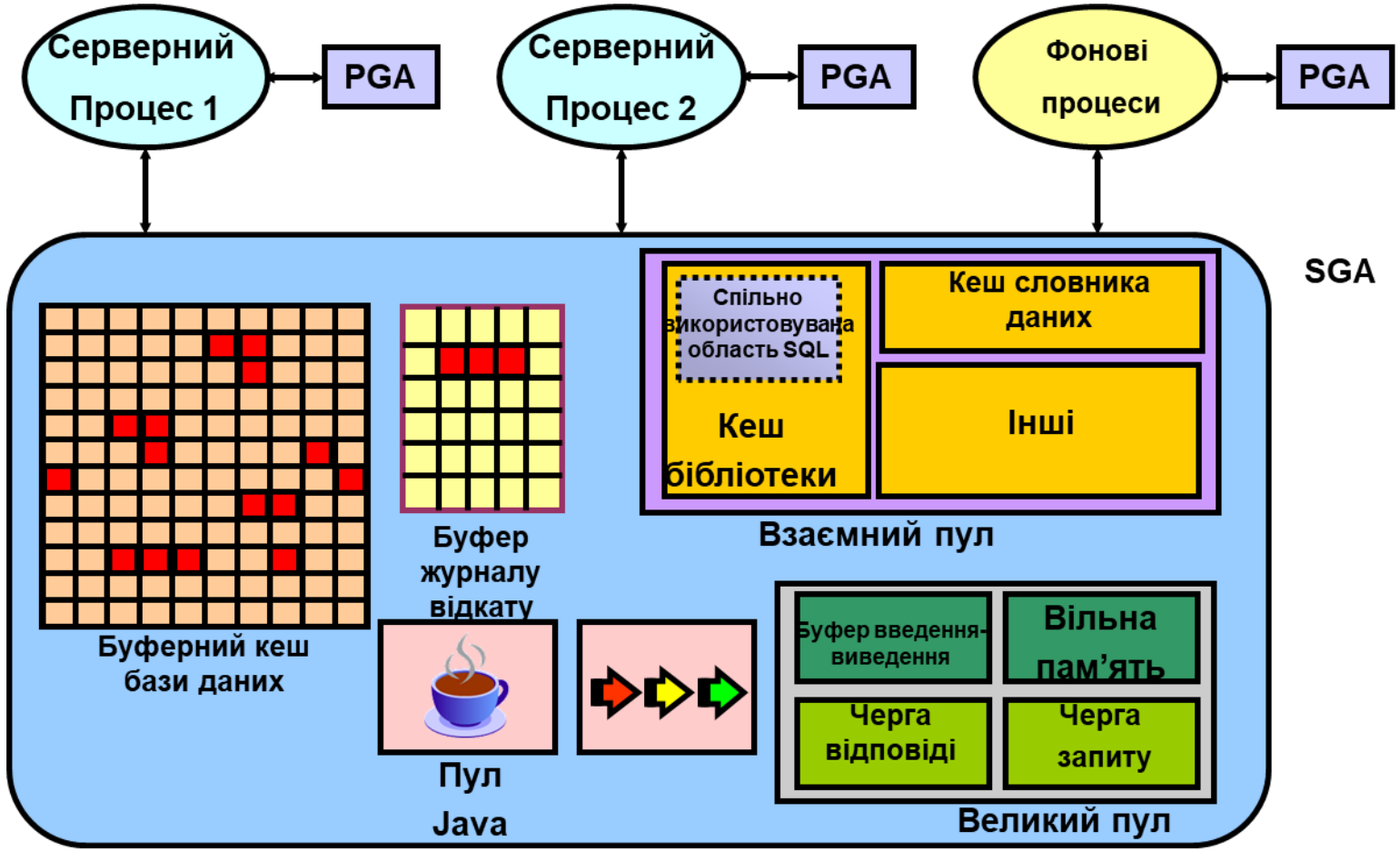
Сеанс – певне з'єднання користувача до інстансу бази даних за допомогою процесу користувача



Взаємодія з базою даних Oracle



Архітектура пам'яті Oracle



System Global Area (SGA)

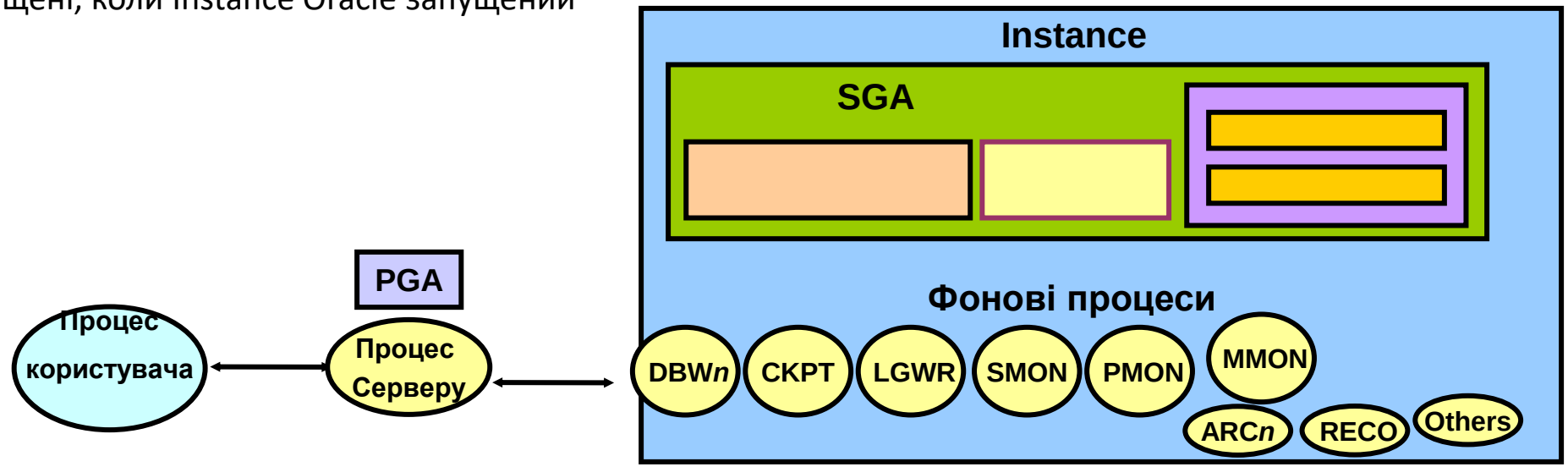
System Global Area (SGA) *є важливою областю пам'яті в архітектурі бази даних Oracle, яка містить дані та керуючу інформацію для конкретного екземпляра бази даних.*

Структура SGA включає наступні компоненти даних:

- буферний кеш бази даних (Database Buffer Cache)
- буфер журналу відкату (Redo Log Buffer)
- спільно використовуваний пул (Shared Pool)
- великий пул (Large Pool)
- пул Java (Java Pool)
- потоковий пул (Streams Pool)

Архітектура процесів

- ***Користувальницький процес***
  - Запущений, коли користувач бази даних або процес пакетної обробки з'єднуються з БД Oracle
- ***Процеси бази даних***
  - серверний процес
    - Підключення до примірника Oracle і запущений, коли користувач встановлює сеанс
  - фонові процеси
    - Запущені, коли Instance Oracle запущений

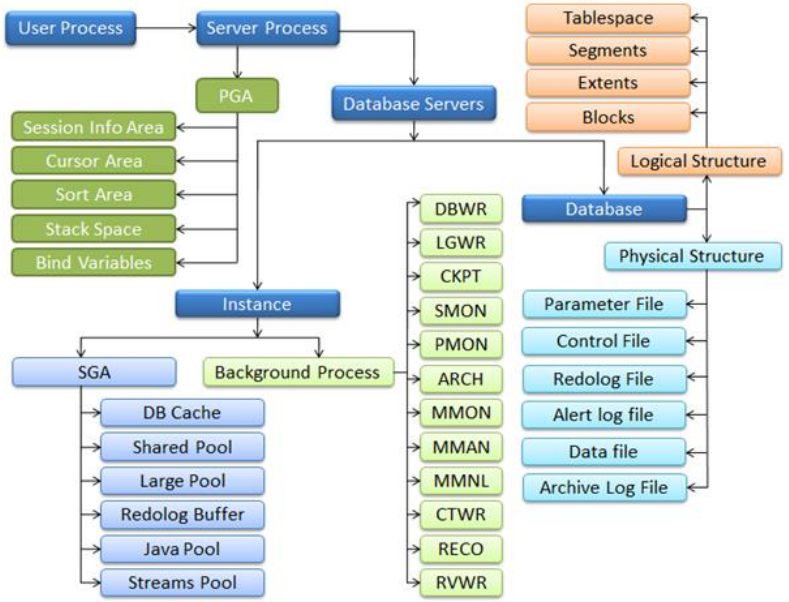


Серверні процеси

- ***Серверні процеси*** в контексті Бази даних Oracle виконують важливу функцію обробки запитів, що надходять від користувацьких процесів, які підключені до конкретного екземпляра бази даних.

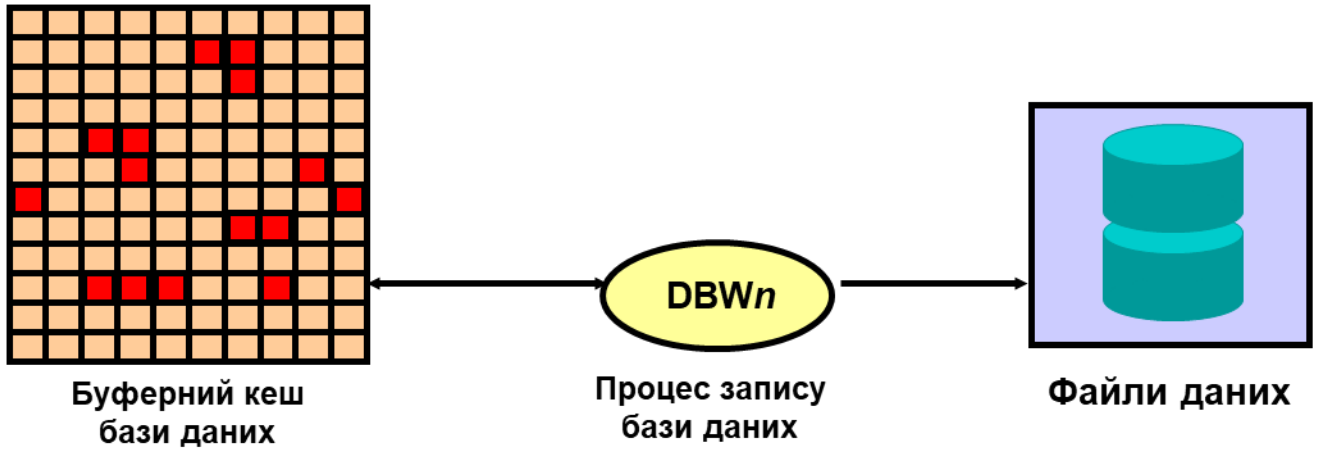
Фонові процеси

- ***Для успішного запуску екземпляра*** бази даних необхідні наступні фонові процеси:
  - процес запису бази даних (Database Writer - DBWn)
  - процес журналу (Log Writer - LGWR)
  - процес контрольної точки (Checkpoint Process - CKPT)
  - системний керуючий процес (System Monitor Process - SMON)
  - процес моніторингу процесу (Process Monitor Process - PMON)
  - процес архівації (Archiver – ARCn)
  - процес відновлення (Recoverer – RECO)
  - процеси черги завдань (QMNC, Qnnn)



Процес запису бази даних (DBWn)

- ***Запис змінених (брудних) буферів в буферному кеші бази даних на диск:***
  - ***Асинхронно при виконанні іншої обробки***
  - ***Періодично – для удосконалення контрольної точки***

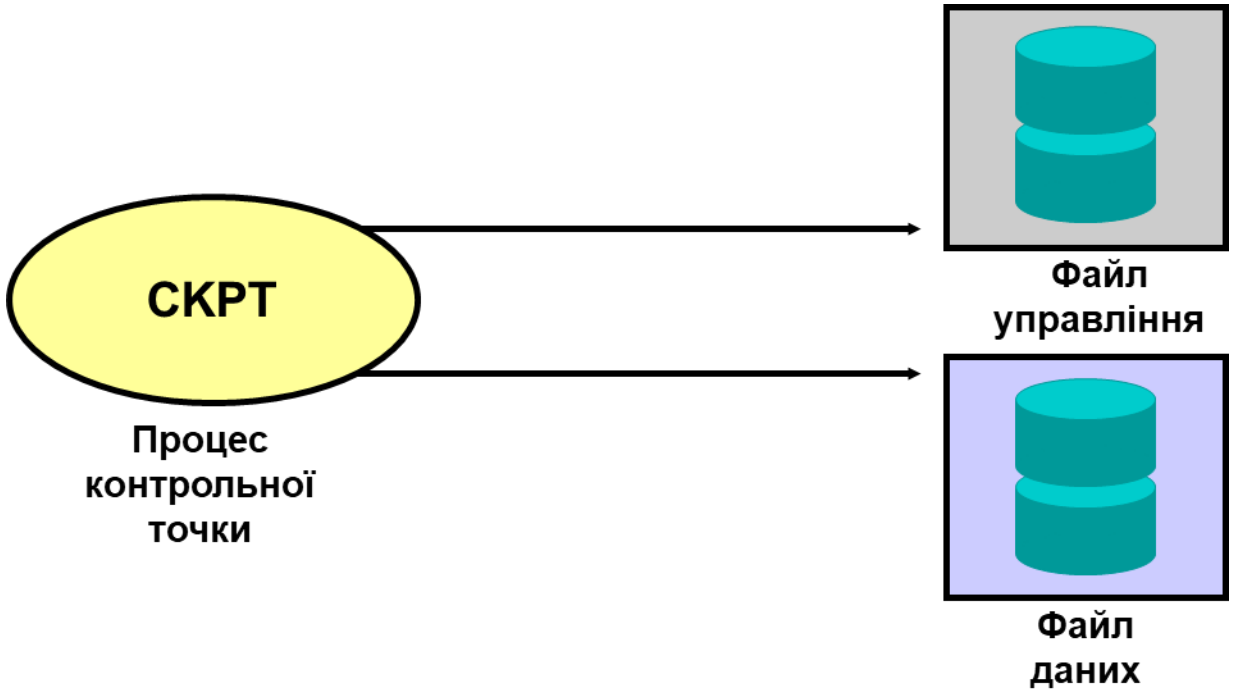


Процес запису журналу (LGWR)

- ***LGWR записує, коли:***
  - Користувацький процес фіксує транзакцію
  - Коли буфер відкату заповнюється до третини
  - Перед процесом DBW при необхідності

Процес контрольної точки (CKPT)

- Записи відзначають інформацію контрольною точкою в:
  - файлі управління
  - в кожному заголовку файлу даних

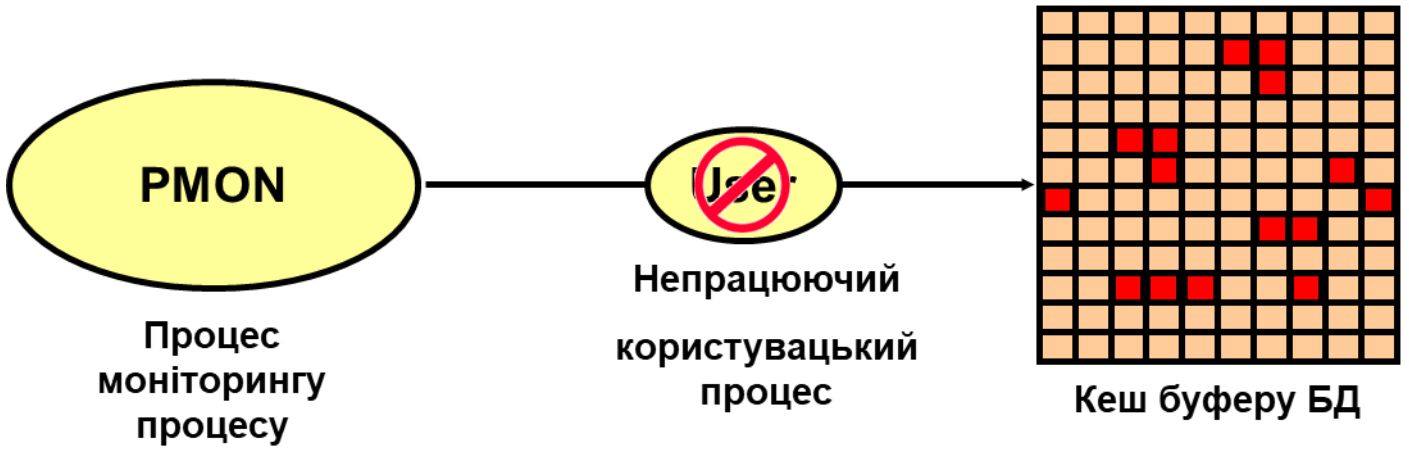


Системний керуючий процес (SMON)

- Виконує відновлення при запуску **Instance**
- Очищає невикористані тимчасові сегменти

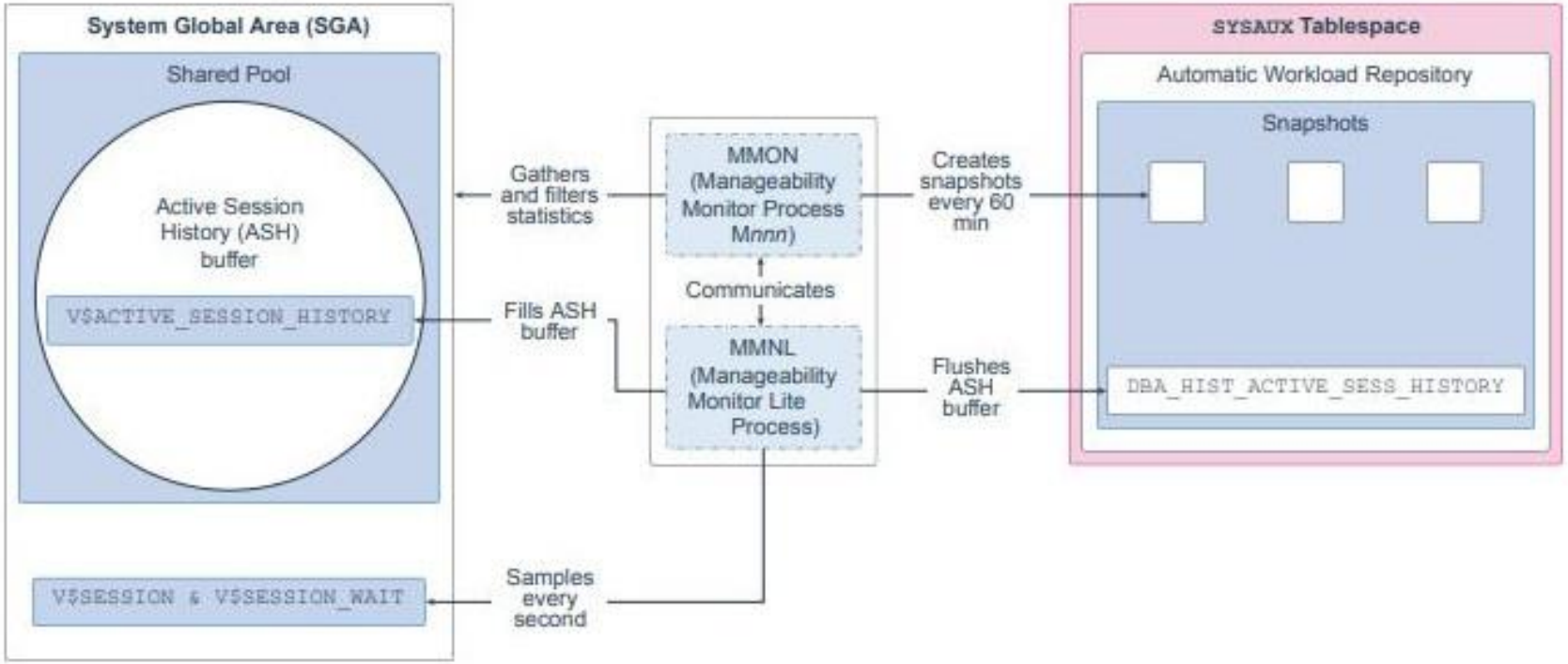
Процес моніторингу процесу (PMON)

- ***Виконує відновлення процесу, коли користувальницький процес перестав працювати***
  - очищає буферний кеш бази даних
  - звільняє ресурси
- ***Контроль стану диспетчера та серверних процесів***
- ***Динамічна служба реєстрації інформації про екземпляри та процеси диспетчера в мережевому слухачі***



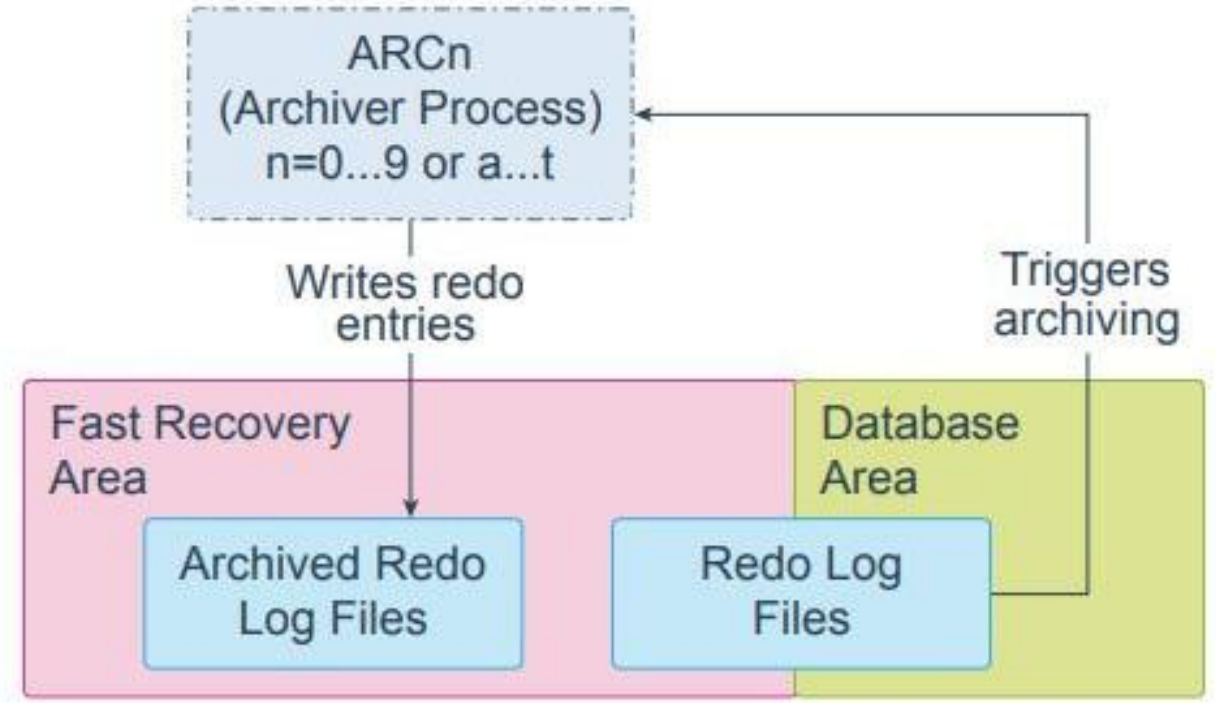
Процеси моніторингу управління (MMON, MMNL))

- ***Підтримка роботи AWR***
- ***Створюють знімки автоматичного репозиторію робочого навантаження (AWR)***



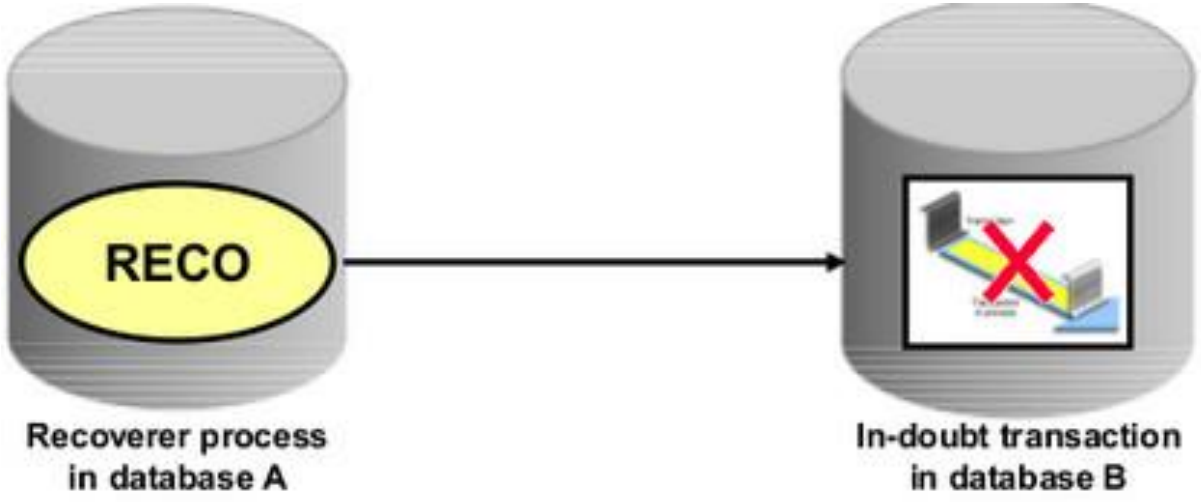
Процес архівації (ARCn)

- ***Архівує онлайн журнали перезавантаження (redologfiles)***



Процес відновлення (RECO)

- ***Відповідає за відновлення всіх сумнівних та очікуючих транзакцій в РБД***
- ***Автоматично вирішує всі проблеми з сумнівними транзакціями***
- ***Видаляє будь-які рядки, що відповідають сумнівним транзакціям***



Процеси керування чергами (QMNC, Qnnn)

- ***QMNC моніторить розширені черги***
- ***QMNC та Qnnn відповідають за поширення черг***

Донецький національний технічний університет

Тема 3. Архітектура БД Oracle
Лекція 4. Архітектура БД Oracle. Структури пам’яті, логічні та фізичні структури зберігання.

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

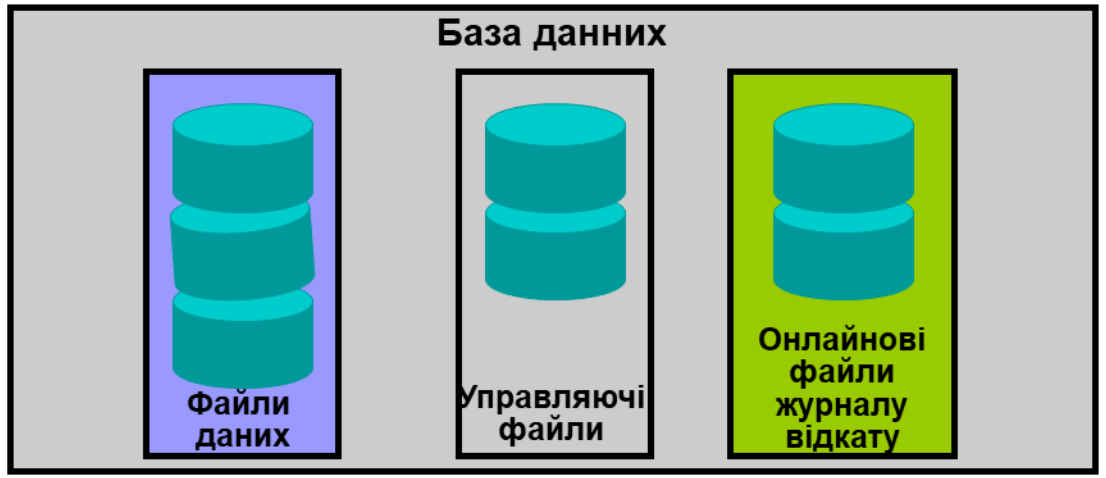
Лекційні питання

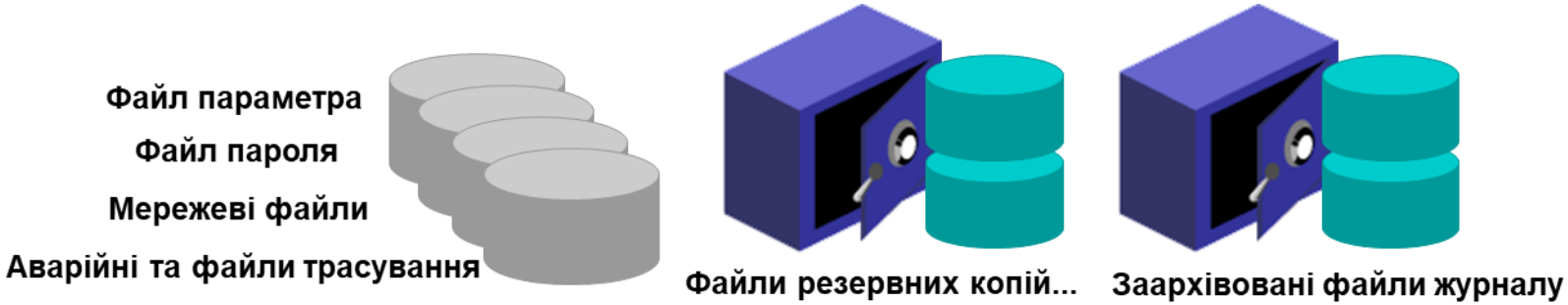
- Мета лекції
- Структури пам'яті БД Oracle
- Логічні та фізичні структури зберігання БД Oracle

Мета лекції

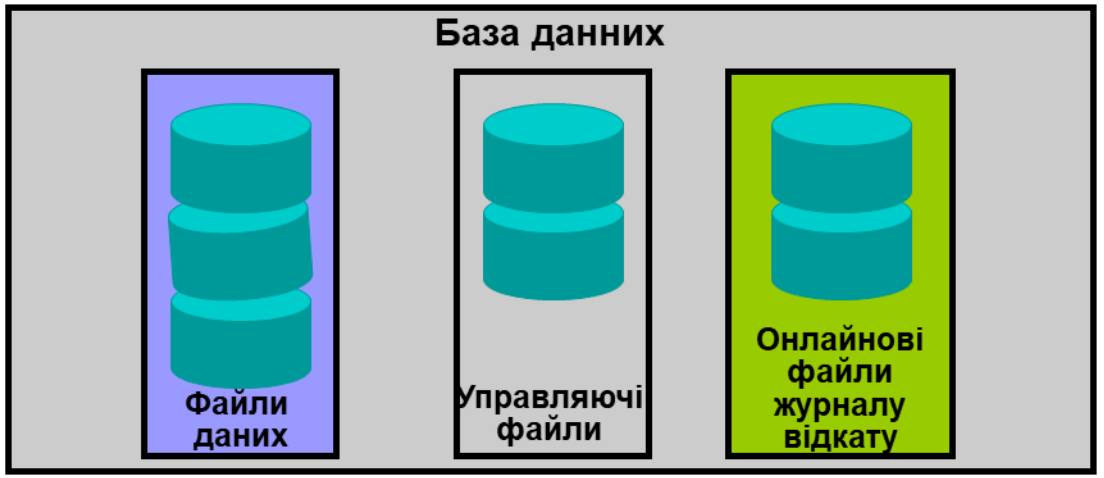
- Розглянути структури пам'яті БД
- Розглянути логічні та фізичні структури зберігання

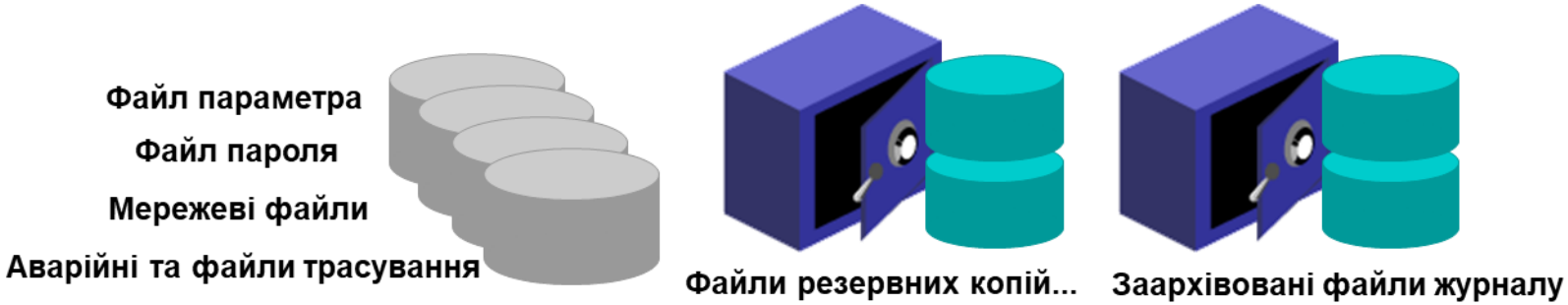
Архітектура зберігання бази даних Oracle



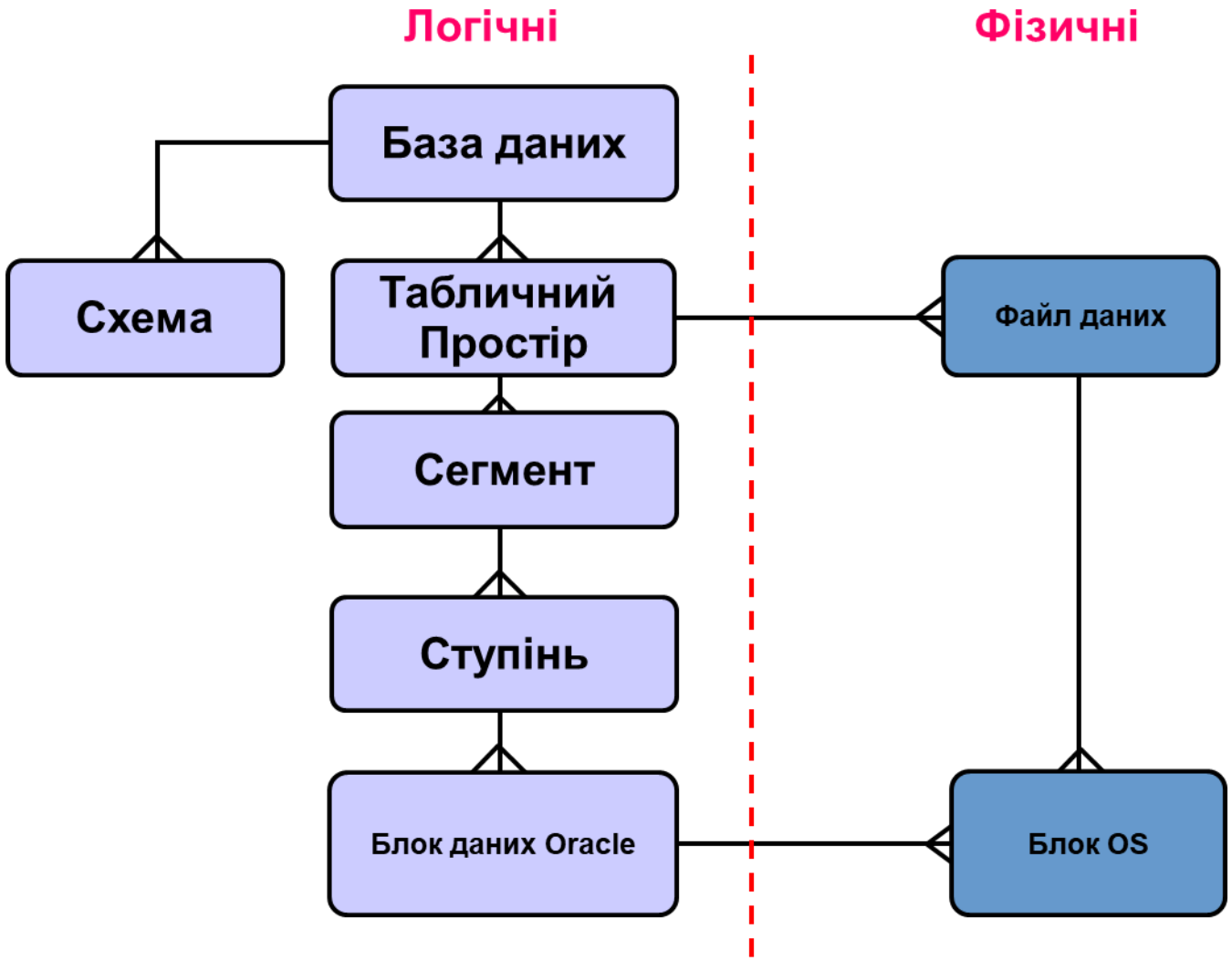


Архітектура зберігання бази даних Oracle. Додаткові файли

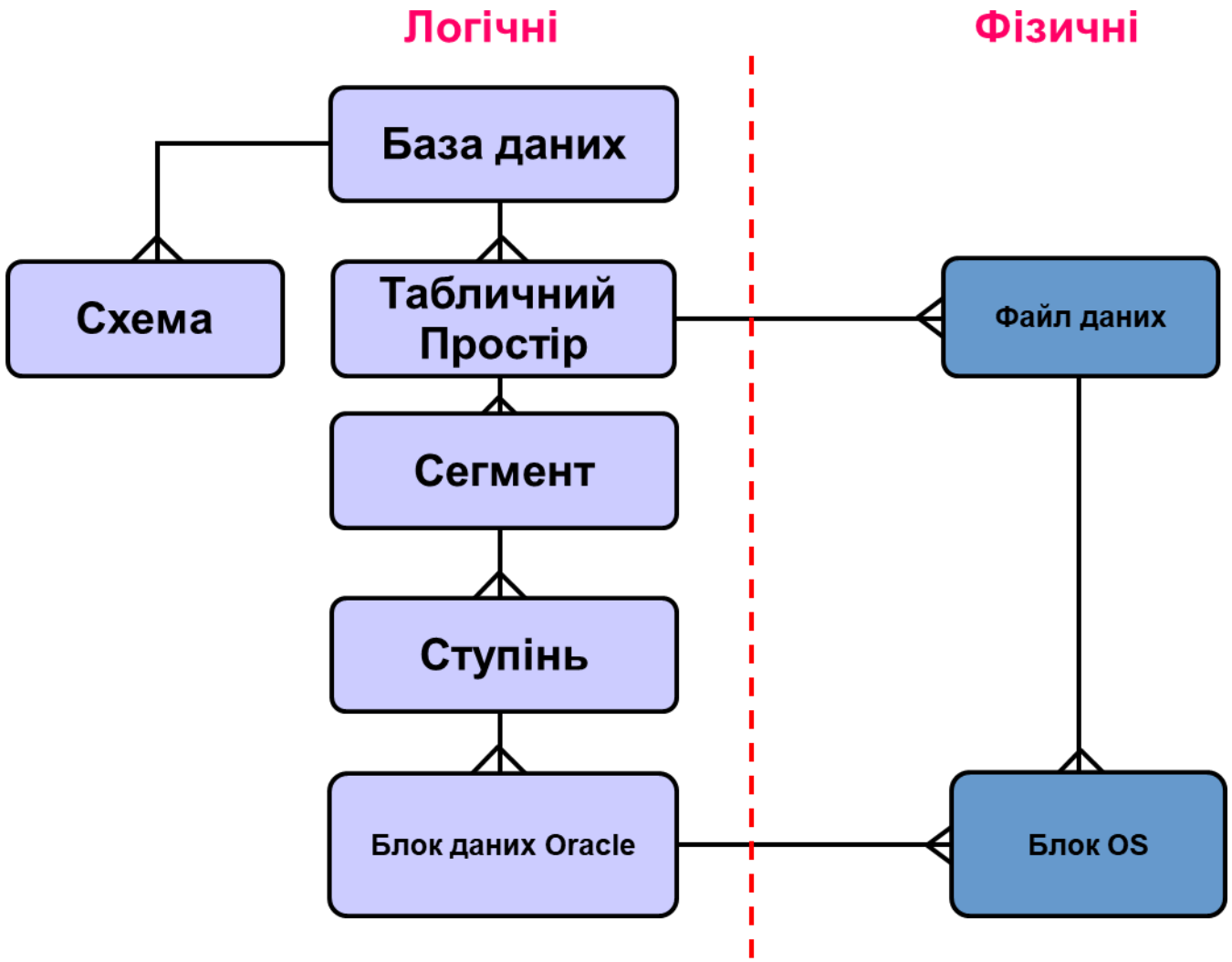




Логічні та фізичні структури бази даних$_1$



Логічні та фізичні структури бази даних$_2$



Обробка SQL-оператора

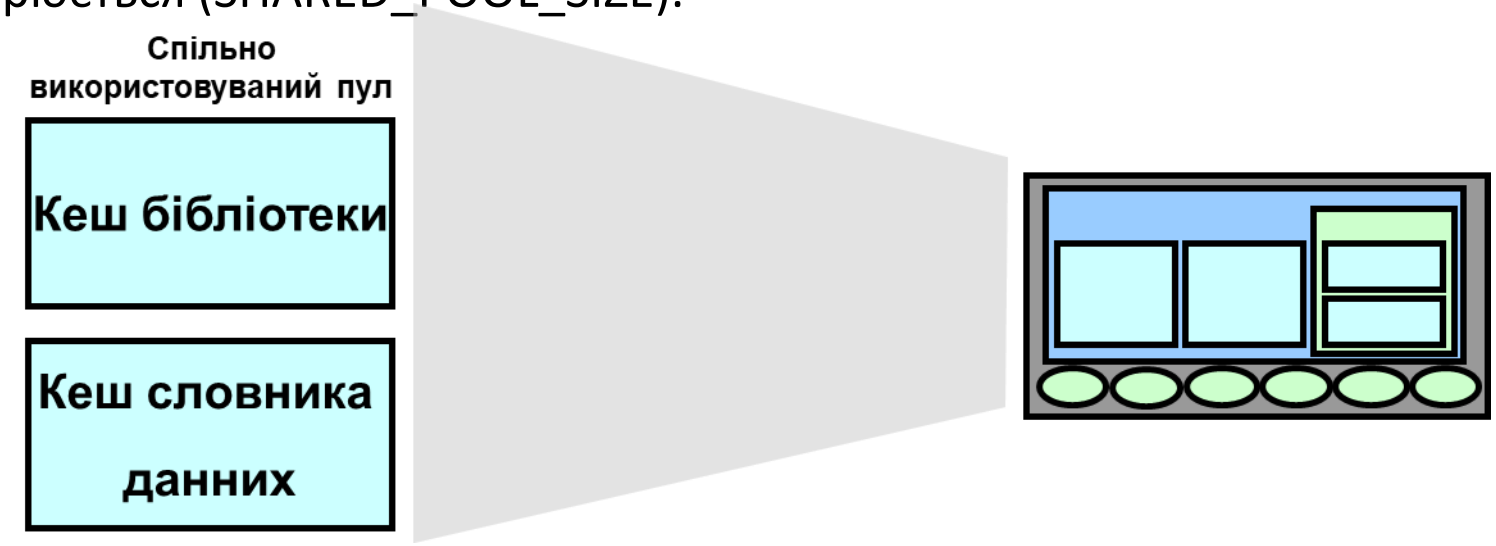
- ***З'єднання з Instance***
  - користувальницький процес
  - серверний процес
- ***Серверні компоненти Oracle, які використовуються, залежать від типу SQL-оператора:***
  - запити повертають рядок
  - оператори мови маніпулювання даними (DML) реєструють зміни
- ***Фіксація гарантує відновлення транзакції***
- ***Деякі серверні компоненти Oracle не беруть участь в обробці SQL-оператора***

Обробка запиту

- ***Синтаксичний аналіз***
  - пошук ідентичного оператора
  - перевірка синтаксису, імена об'єктів та повноважень
  - використання об'єктів блокування під час синтаксичного аналізу
  - створення та зберігання плану виконання
- ***Виконання***
  - ідентифікація обраних рядків
- ***Вибірка***
  - Повернення рядків процесу користувача

Спільно використовуваний пул

- Кеш бібліотеки містить текст SQL-оператора, проаналізований код і план виконання
- Кеш словника даних містить таблицю, стовпець, і інші визначення об'єктів та повноваження
- Спільно використовуваний пул вимірюється (SHARED_POOL_SIZE).



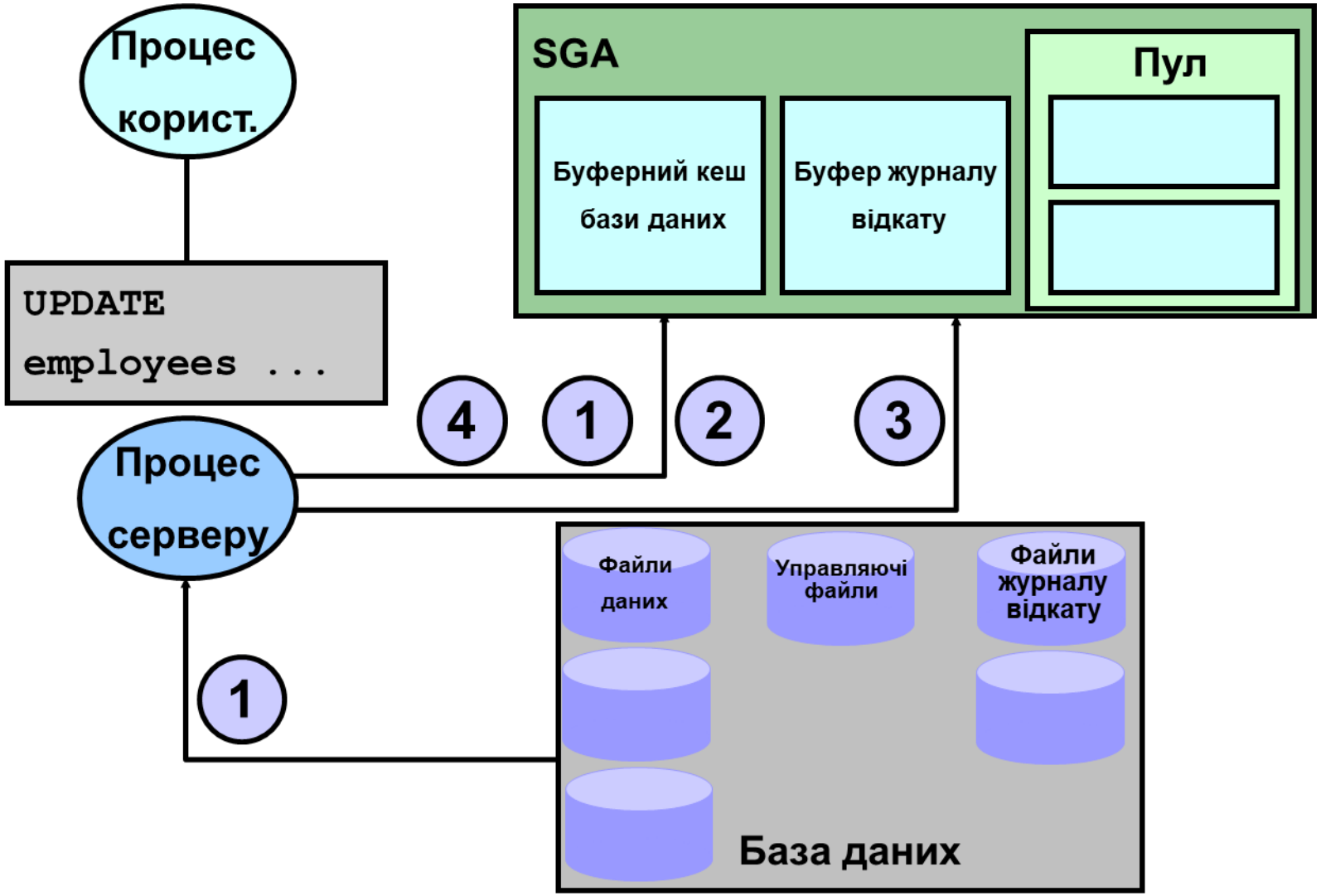
Буферний кеш бази даних

- Буферний кеш бази даних зберігає останній раз використовувані блоки
- Розмір буфера ґрунтується на DB_BLOCK_SIZE
- Число буферів визначено DB_BLOCK_BUFFERS

Глобальна область програми(PGA)

- Не використовується спільно
- Перезапис тільки серверним процесом
- Містить:
  - область Sort
  - інформацію про сеанс
  - стан курсору
  - стековий простір

Обробка оператора DML

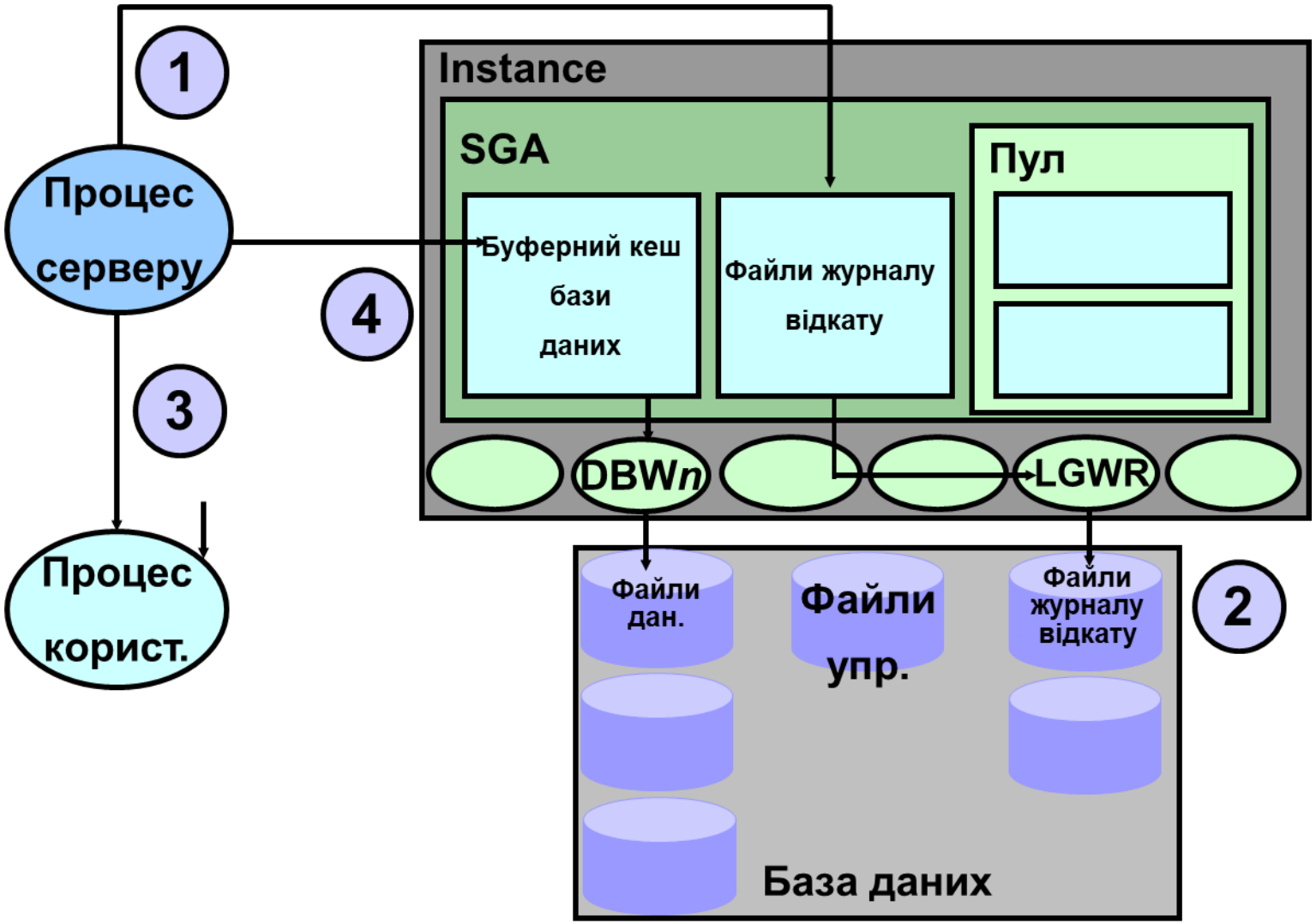


Буфер журналу відкату

- Розмір визначається LOG_BUFFER
- Зміни записів внесені через Instance
- Використовується послідовно
- Круговий буфер

Сегмент відкату

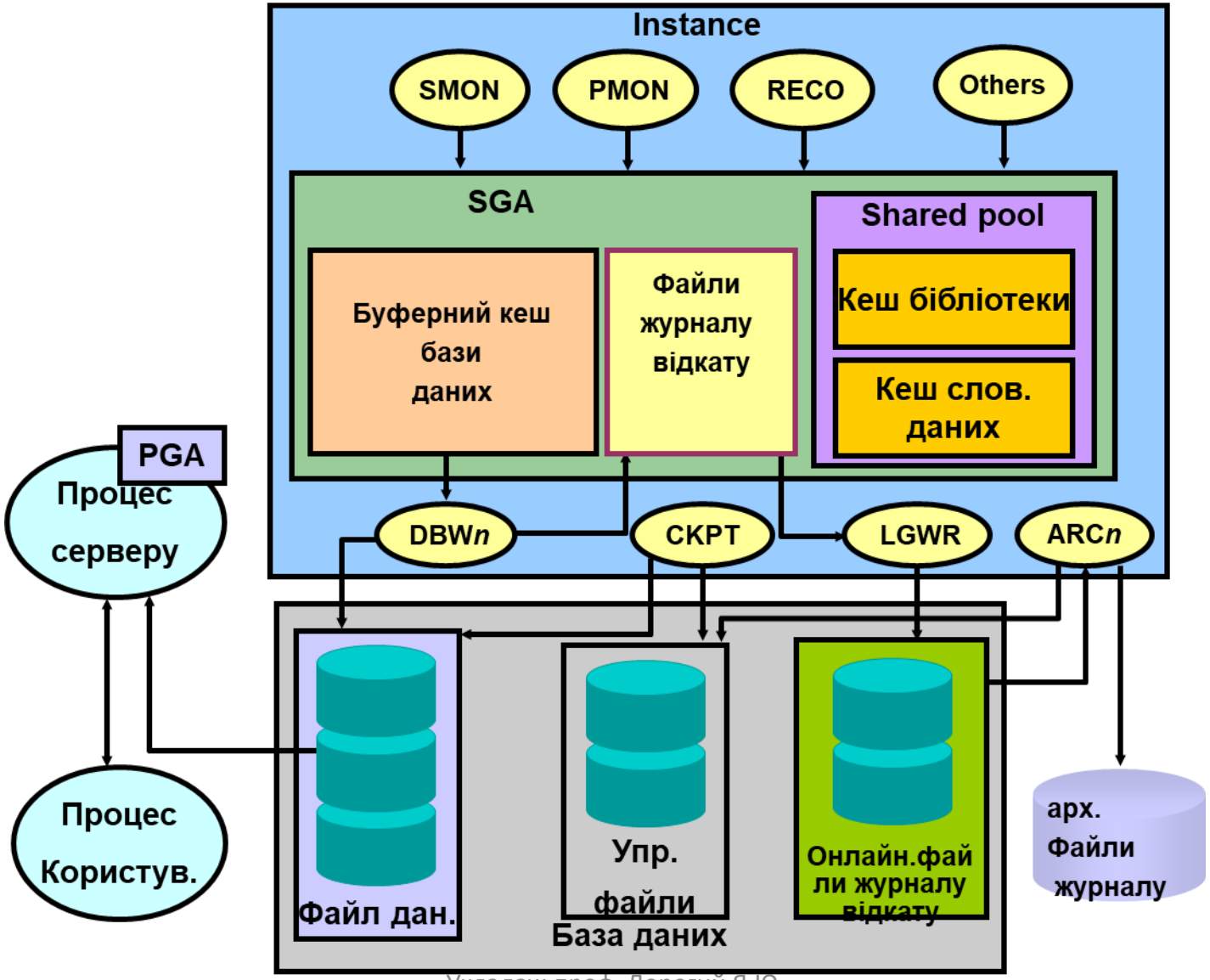
Обробка фіксації



Швидка фіксація. Переваги

- ***Ефективна синхронізація транзакцій***
- ***Мінімізація синхронних записів***
- ***Залежність від розміру транзакції***
- ***Швидкість записів у файли журналу***
- ***Мінімальний обсяг запису до файлів журналу***

Підсумок архітектури бази даних Oracle



Донецький національний технічний університет

Тема 4. Автентифікація та авторизація в БД Oracle. Лекція 5. Управління доступом

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Мета лекції
- Системні повноваження
- Створення ролі
- Об'єктні повноваження
- Скасування об'єктних повноважень

Мета лекції

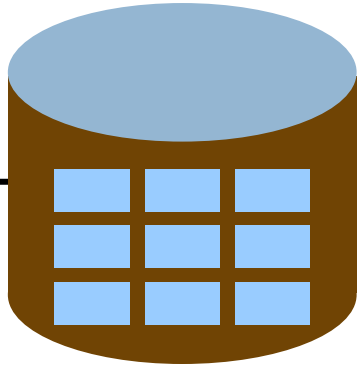
- Ознайомитися з системними та об'єктними повноваженнями
- Навчитися надавати повноваження
- Навчитися надавати ролі
- Навчитися розрізняти повноваження і ролі

Управління користувальницьким доступом

Адміністратор
Бази данних

Ім'я користувача та пароль

повноваження



Користувачі

Повноваження

- Безпека бази даних:
  - Безпека системи
  - Безпека даних
- Системні повноваження – виконання певної дії в базі даних
- Об'єктні повноваження - управління вмістом об'єктів бази даних
- Схема – набір об'єктів, таких як таблиці, представлення і послідовності

Системні повноваження

- В СУБД доступно більше, ніж 100 системних повноважень
- У адміністратора бази даних є високорівневі системні повноваження для таких задач, як:
  - Створення/видалення користувачів
  - Створення/видалення таблиць
  - Інші

Створення користувачів

- Адміністратор бази даних (DBA) створює користувачів за допомогою оператора CREATE USER

```
CREATE USER user
IDENTIFIED BY password;
```

```
CREATE USER demo
IDENTIFIED BY demo;
```

Системні повноваження користувача

- Після того, як користувача створено, адміністратор може надати йому певні системні повноваження

```
GRANT privilege [, privilege...]
TO user [, user| role, PUBLIC...];
```

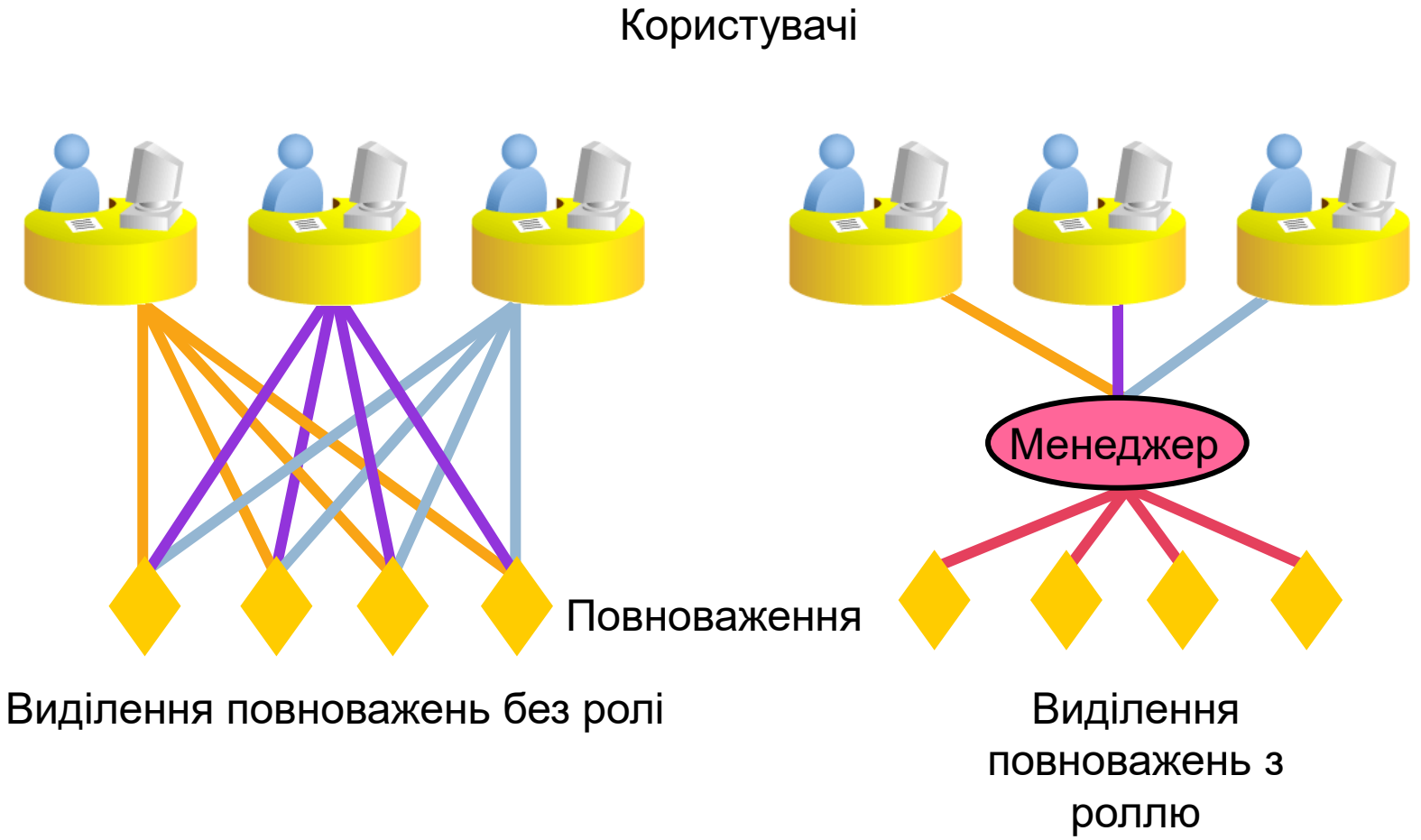
Системні повноваження	Авторизовані операції
CREATE SESSION	Приєднатися до бази даних
CREATE TABLE	Створити таблицю
CREATE SEQUENCE	Створити послідовність у схемі користувача
CREATE VIEW	Створити певний вид (представлення)
CREATE PROCEDURE	Створити процедуру, функцію або пакет у схемі користувача

Надання системних повноважень

- Після того, як користувача створено, адміністратор може надати йому певні системні повноваження

```
GRANT   create session, create table,
        create sequence, create view
TO      demo;
```

Що таке роль?


Користувачі
Менеджер
Повноваження
Виділення повноважень без ролі
Виділення повноважень з роллю

Створення і надання повноважень для ролі

- Створення ролі

```
CREATE ROLE manager;
```

- Надання повноважень ролі

```
GRANT create table, create view
TO manager;
```

- Надання ролі користувачам

```
GRANT manager TO BELL, KOCHHAR;
```

Зміна вашого пароля

- Адміністратор створює ваш обліковий запис користувача і ініціалізує початковий пароль
- Далі Ви можете змінити свій пароль за допомогою оператора ALTER USER

```
ALTER USER demo
IDENTIFIED BY employ;
```

Об’єктні повноваження$_1$

Об’єктні повноваження	TABLE	VIEW	SEQUENCE
ALTER	✓		✓
DELETE	✓	✓	
INDEX	✓		
INSERT	✓	✓	
REFERENCES	✓		
SELECT	✓	✓	✓
UPDATE	✓	✓	

Об’єктні повноваження$_2$

- Об'єктні повноваження варіюються від об'єкту до об'єкту
- У власника є всі повноваження на об'єкт
- Власник може дати певні повноваження на об'єкт іншому користувачу

```
GRANT       object_priv [(columns)]
ON          object
TO          {user|role|PUBLIC}
[WITH GRANT OPTION];
```

Надання об'єктних повноважень

- Надання повноважень на запити до таблиці EMPLOYEES

```
GRANT   select
ON      employees
TO      demo;
```

- Надання повноважень на оновлення певних стовпців користувачу і ролі

```
GRANT   update (department_name, location_id)
ON      departments
TO      demo, manager;
```

Передача повноважень

- Надання повноважень користувачу на передачу повноважень іншим користувачам

```
GRANT   select, insert
ON      departments
TO      demo
WITH    GRANT OPTION;
```

- Надання повноважень всім користувачам в системі вибирати дані з таблиці DEPARTMENTS схеми alice

```
GRANT   select
ON      alice.departments
TO      PUBLIC;
```

Підтвердження наданих повноважень

Представлення словника даних	Опис
ROLE_SYS_PRIVS	Системні повноваження, надані ролям
ROLE_TAB_PRIVS	Табличні повноваження, надані ролям
USER_ROLE_PRIVS	Ролі доступні користувачу
USER_SYS_PRIVS	Системні повноваження, надані користувачеві
USER_TAB_PRIVS_MADE	Об'єктні повноваження надані на об'єктах користувача
USER_TAB_PRIVS_RECD	Об'єктні повноваження, надані користувачеві
USER_COL_PRIVS_MADE	Об'єктні повноваження, надані на стовпцях об'єктів користувача
USER_COL_PRIVS_RECD	Об'єктні повноваження, надані користувачеві на певних стовпцях

Скасування об’єктних повноважень$_1$

- Для відкликання повноважень, наданих іншим користувачам, використовується оператор REVOKE
- Повноваження, надані іншим через опцію WITH GRANT OPTION відміняються також

```
REVOKE {privilege [, privilege...]|ALL}
ON     object
FROM   {user[, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

Скасування об’єктних повноважень$_2$

- Скасування повноважень SELECT і INSERT користувачу demo для таблиці DEPARTMENTS

```
REVOKE   select, insert
ON       departments
FROM     demo;
```

Донецький національний технічний університет

Тема 5. Захист та шифрування даних в БД Oracle.
Лекція 6. Захист даних у Oracle

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Мета лекції
- Шифрування даних у СУБД Oracle
- Управління ключами в СУБД Oracle

Мета лекції

- Ознайомитися з бібліотеками шифрування даних у СУБД Oracle
- Управління ключами в СУБД Oracle

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_1$

- При використанні підпрограм шифрування та алгоритму MD5 пакета DBMS_OBFUSCATION_TOOLKIT (можна використовувати до версії Oracle 20c) повинні виконуватись такі вимоги:
- довжина даних, що шифруються, повинна бути кратна 8 (так 9-байтове поле типу VARCHAR2 треба буде доповнювати до 16 байт);
- ключ, який використовується для шифрування даних, повинен мати довжину 8 байт для процедури DESEncrypt та 16 або 24 байт для процедур DES3Decrypt;
- залежно від виду шифрування, що використовується, необхідно викликати різні підпрограми (при 56-бітовому шифруванні викликаються підпрограми DESENCRYPT і DESDECRYPT, при 112/168-бітовому - DES3ENCRYPT і DES3DECRYPT;
- стандартні підпрограми шифрування дозволяють безпосередньо шифрувати дані обсягом до 32 Кбайт;
- підпрограми, що реалізують алгоритм MD5, перевантажені аналогічно і не можуть використовуватися в SQL-операторах.

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_2$

Приклад 1 . Зашифрування рядка «SHHH..TOP SECRET»:

```
DECLARE
       l_enc_val VARCHAR2 (200);
BEGIN
       DBMS_OBFUSCATION_TOOLKIT.des3encrypt
              (input_string => 'SHHH..TOP SECRET',
              key_string => 'ABCDEFGHIJKLMNOP',
              encrypted_string => l_enc_val
              );
       DBMS_OUTPUT.put_line('Encrypted Value = ' || l_enc_val);
END;
/

Encrypted Value = jVжюTF .(e)♥ьo╠0
PL/SQL процедура успішно здійснена.
```

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_3$

Модифікація прикладу 1:

```
DECLARE
      l_enc_val VARCHAR2 (200);
BEGIN
      DBMS_OBFUSCATION_TOOLKIT.des3encrypt
      (input_string    => 'SHHH..TOP SECRET',
      key_string      => 'ABCDEFGHIJKLMNOP',
      encrypted_string => l_enc_val);
      l_enc_val := RAWTOHEX(UTL_RAW.cast_to_raw(l_enc_val));
      DBMS_OUTPUT.put_line('Encrypted Value = ' || l_enc_val);
END;
/

Encrypted Value = A86A56A6EE92462E28652903ECAEC730
PL/SQL процедура успішно здійснена.
```

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_4$

Грунтуючись на функціях шифрування з пакета DBMS_OBFUSCATION_TOOLKIT , складемо кілька функцій, щоб зробити процес шифрування більш простим та гнучким.

```
CREATE OR REPLACE FUNCTION get_enc_val
(p_in_val IN VARCHAR2, p_key IN VARCHAR2)
RETURN VARCHAR2
IS
l_enc_val VARCHAR2 (200);
BEGIN
      l_enc_val := DBMS_OBFUSCATION_TOOLKIT.des3encrypt
            (input_string => p_in_val,
            key_string => p_key);
      l_enc_val := RAWTOHEX(UTL_RAW.cast_to_raw(l_enc_val));
      RETURN l_enc_val;
END;
```

Цю функцію шифрування можна використовувати різними способами – для вставки даних у зашифровані стовпці, передачі зашифрованих даних до інших процедур і функцій тощо.

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_5$

Протестуємо цю функцію на різних вхідних значеннях.
Приклад 1.

```
DECLARE
        v_enc VARCHAR2(200);
BEGIN
        v_enc := get_enc_val('SHHH..TOP SECRET',
                'ABCDEFGHIJKLMNOP');
        DBMS_OUTPUT.put_line('Encrypted value = ' || v_enc);
END;
/

Encrypted value = A86A56A6EE92462E28652903ECAEC730
PL/SQL процедура успішно здійснена.
```

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_5$

У першому прикладі ми зашифровували рядок «SHHH..TOP SECRET». Тепер зашифруємо інше значення :

```
DECLARE
        v_enc VARCHAR2 (200);
BEGIN
        v_enc := get_enc_val('The DIFFERENT VALUE ', 'ABCDEFGHIJKLMNOP');
        DBMS_OUTPUT.put_line('Encrypted value = ' || v_enc);
END;
/
ERROR at line 1:
ORA-28232: invalid input length for obfuscation toolkit
ORA-06512: at "SYS.DBMS_OBFUSCATION_TOOLKIT_FFI", line 59
ORA-06512: y "SYS.DBMS_OBFUSCATION_TOOLKIT", line 216
ORA-06512: at "SCOTT.GET_ENC_VAL", line 7
ORA -06512: at line 4
```

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_6$

Зміни у функції:

```
CREATE OR REPLACE FUNCTION get_enc_val
(p_in_val IN VARCHAR2, p_key IN VARCHAR2)
RETURN VARCHAR2
IS
      l_enc_val VARCHAR2(200);
      l_in_val VARCHAR2(200);
BEGIN
      l_in_val := RPAD(p_in_val, (8*ROUND(LENGTH(p_in_val)/8,0)+8));
      l_enc_val := DBMS_OBFUSCATION_TOOLKIT.des3encrypt
            (input_string => l_in_val,
            key_string => p_key);
      l_enc_val := RAWTOHEX(UTL_RAW.cast_to_raw(l_enc_val));
      RETURN l_enc_val ;
END;
/
Function created.
```

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_6$

Змінена функція шифрування:

```
CREATE OR REPLACE FUNCTION get_enc_val(
        p_in_val IN VARCHAR2,
        p_key IN VARCHAR2,
        p_iv IN VARCHAR2 := NULL
)
RETURN VARCHAR2
IS
        l_enc_val VARCHAR2(200);
        l_in_val VARCHAR2(200);
        l_iv VARCHAR2(200);
BEGIN
        l_in_val := RPAD(p_in_val,(8*ROUND(LENGTH(p_in_val)/8,0)+8));
        l_iv := RPAD(p_iv,(8*ROUND(LENGTH(p_iv)/8,0)+8));
        l_enc_val := DBMS_OBFUSCATION_TOOLKIT.des3encrypt
                (input_string => l_in_val,
                key_string => p_key,
                iv_string => l_iv);
        l_enc_val := RAWTOHEX(UTL_RAW.cast_to_raw(l_enc_val));
        RETURN l_enc_val ;
END;
/
```

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_7$

Розшифрування даних

```
DECLARE
        l_enc_val VARCHAR2(2000);
        l_dec_val VARCHAR2(2000) := 'Clear Text Data';
        l_key VARCHAR2(2000) := 'ABCDEFGHIJKLMNOP';
BEGIN
        l_enc_val := get_enc_val(l_dec_val, l_key, '12345678');
        l_dec_val := DBMS_OBFUSCATION_TOOLKIT.des3decrypt
                (input_string => UTL_RAW.cast_to_varchar2(HEXTORAW(
                l_enc_val)),
        key_string => l_key);
        DBMS_OUTPUT.put_line('Decrypted Value = ' || l_dec_val);
END;
/

Decrypted Value = s}_2OS®Ixt Data
PL/SQL процедура успішно здійснена.
```

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_8$

Розшифрування даних **виправлення помилки:**

```
DECLARE
        l_enc_val VARCHAR2(2000);
        l_dec_val VARCHAR2(2000) := 'Clear Text Data';
        l_key VARCHAR2(2000) := 'ABCDEFGHIJKLMNOP';
BEGIN
        l_enc_val := get_enc_val(l_dec_val, l_key, '12345678');
        l_dec_val := DBMS_OBFUSCATION_TOOLKIT.des3decrypt
                        (input_string => UTL_RAW.cast_to_varchar2(HEXTORAW(
                        l_enc_val)),
        key_string => l_key,
        iv_string => '12345678'
        );
        DBMS_OUTPUT.put_line('Decrypted Value = ' || l_dec_val);
END;
/
Decrypted Value = Clear Text Data
PL/SQL процедура успішно здійснена.
```

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_9$

***Шифрування даних типу* RAW**

Коли потрібно використовувати шифрування у форматі RAW?

- *По-перше*, шифрування у форматі RAW використовується для даних типу BLOB.
- *По-друге*, шифрування у форматі RAW використовується у випадку, якщо в базі даних використовуються літери не англійського алфавіту. Якщо ви користуєтеся функціональністю *Oracle Globalization Support* (раніше *National Language Support – NLS*), то шифрування у форматі RAW забезпечить обробку таких символів без виконання будь-яких додаткових операцій (зокрема, при експорті та імпорті даних). Зашифровані дані можна буде переміщати з однієї бази даних до іншої, не побоюючись їх пошкодження.

Однак у загальному випадку уникайте маніпуляцій з типом RAW, *якщо спочатку відомо, що ваші дані мають символьний тип, і ви використовуєте лише один набір символів.*

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_{10}$

Отже, основні правила шифрування в молодших версіях Oracle використовують пакет DBMS_OBFUSCATION_TOOLKIT:

- Шифрування може виконуватися за алгоритмом DES або DES3 (*Triple DES*), причому DES3 є кращим.
- Шифрування DES3 може бути дво- чи трипрохідним. За замовчуванням використовуються два проходи.
- Довжина рядка, що шифрується, повинна бути кратна восьми.
- Ключ, який використовується для зашифрування, повинен використовуватися і для розшифровування.
- Довжина ключа повинна бути не менше 16 для DES та двопрохідного DES3, і не менше 24 для трипрохідного DES3.
- Для посилення захисту даних можливе використання вектора ініціалізації. Якщо вектор ініціалізації використаний під час зашифрування, його також необхідно вказати при розшифруванні.
- Так як вектор ініціалізації приєднується на початок вхідного значення, довжина отриманого об'єднаного рядка повинна бути кратна восьми.

Зауваження. Неможливо зашифрувати дані, які вже зашифровані засобами пакета DBMS_OBFUSCATION_TOOLKIT. При подібній спробі пакет генерує помилку ORA-28233 через неможливість подвійного шифрування.

Шифрування даних у СУБД Oracle (DBMS_OBFUSCATION_TOOLKIT)$_{11}$

При виборі ключа шифрування слід пам'ятати:

- чим довше ключ, тим складніше його вгадати (двопрохідний метод допускає використання ключа з 128 біт, а трипрохідний - з 192 біт; слід вибирати найдовший з можливих ключів);
- ключ повинен бути не лише довгим, він ще не повинен відповідати жодному шаблону, який легко було б вгадати.

У пакет DBMS_OBFUSCATION_TOOLKIT включені дві функції DESGETKEY та DES3GETKEY (а також їм відповідні процедури, причому всі вони перевантажені з кількома типами даних: RAW та VARCHAR2), які дозволяють згенерувати криптографічно допустимий ключ.

Приклад виклику функції:

```
l_ret := DBMS_OBFUSCATION_TOOLKIT.desgetkey
(seed _ string => l _ seed);
```

Значення змінної l_seed – випадковий рядок довжиною 80 символів (довше значення буде прийнято, але використано буде лише 80 символів). Значення, що повертається записується в змінну l_ret.

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_1$

Пакет **DBMS_CRYPTO** має ряд переваг перед **DBMS_OBFUSCATION_TOOLKIT**:

- більший вибір алгоритмів шифрування, зокрема, підтримка останнього стандарту AES (*Advanced Encryption Standard*);
- можливість потокового шифрування;
- підтримка алгоритму SHA-1 (*Secure Hash Algorithm 1*);
- здатність створення коду MAC;
- шифрування великих об'єктів (LOB) у їхньому власному форматі.

Пакет DBMS_CRYPTO, доступний з версії Oracle 10g, містить функцію GETRANDOMBYTES, яка може використовуватися для формування криптографічно випадкових ключів.

Крім генерування ключів у форматі RAW (за допомогою функції RANDOMBYTES) пакет DBMS_CRYPTO забезпечує формування числових значень, а також двійкових цілих. Функція RANDOMINTEGER генерує двійковий ключ, наприклад:

```
l_ret := DBMS_CRYPTO.randominteger;
```

Функція RANDOMNUMBER генерує ключ цілого типу довжиною 2^{128}:

```
l_ret := DBMS_CRYPTO.randomnumber;
```

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_2$

Зашифрування даних

Після того як ключ готовий, приступимо до шифрування даних. Для цього скористаємося програмою `ENCRYPT` пакета `DBMS_CRYPTO`.

`ENCRYPT` перевантажена та існує і як функції, так і процедури. На відміну від пакета `DBMS_OBFUSCATION_TOOLKIT`, таке навантаження для неї в пакеті `DBMS_CRYPTO` обґрунтовано: функція приймає як вхідне значення лише тип даних `RAW`, тоді як процедура приймає на вхід тільки значення типів `CLOB` і `BLOB`.

Приклад інтерфейсу функції шифрування `ENCRYPT`, що повертає значення `RAW`:

```
DBMS_CRYPTO.encrypt(
  src IN RAW,
  typ IN PLS_INTEGER,
  key IN RAW,
  iv IN RAW DEFAULT NULL)
RETURN RAW;
```

де `src` - вхідне значення, яке підлягає шифруванню;

`key` – ключ шифрування;

`iv` – вектор ініціалізації;

`typ` – тип шифрування.

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_3$

Пакети DBMS_OBFUSCATION_TOOLKIT та DBMS_CRYPTO відрізняються методами підтримки різних типів шифрування. Пакет DBMS_OBFUSCATION_TOOLKIT пропонує спеціальні функції (і відповідні процедури) для кожного алгоритму, наприклад DESENCRYPT для DES та DES3ENCRYPT для Triple DES. Пакет DBMS_CRYPTO надає лише одну функцію, а тип шифрування вказується у параметрі. Підтримувані алгоритми шифрування та відповідні їм константи наведено у таблиці.

Константа	Опис	Реальна довжина ключа
ENCRYPT_DES	Data Encryption Standard (DES)	56
ENCRYPT_3DES_2KEY	Modified Triple Data Encryption Standard (3DES); обробляє кожний блоки тричі, використовуючи 2 ключа	112
ENCRYPT_3DES	Triple Data Encryption Standard (3DES); обробляє кожний блок тричі	156
ENCRYPT_AES128	Advanced Encryption Standard (AES)	128
ENCRYPT_AES192	Advanced Encryption Standard (AES)	192
ENCRYPT_AES256	Advanced Encryption Standard (AES)	256
ENCRYPT_RC4	Потокове шифрування	

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_4$

Режими шифрування

При шифруванні даних кожен блок, що шифрується, може бути зашифрований залежно від режиму шифрування (ECB, CBC, CFB, OFB). Щоб вибрати шифрування, що вас цікавить, вкажіть відповідну константу з таблиці у значенні параметра typ, наприклад, DBMS_CRYPTO.CHAIN_OFB.

Константа	Опис
CHAIN_CBC	Зчеплення блоків шифротексту – Cipher Block Chaining (CBC)
CHAIN_ECB	Електронна книга кодів – Electronic Code Book (ECB)
CHAIN_CFB	Шифрування зі зворотнім зв'язком за шифротекстом – Cipher Feedback (CFB)
CHAIN_OFB	Шифрування зі зворотнім зв'язком за виходом – Output Feedback (OFB)

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_5$

Типи доповнення

При використанні пакета DBMS_OBFUSCATION_TOOLKIT необхідно явно доповнити дані так, щоб їх довжина була кратна розміру блоку. Однак, цей підхід не є криптографічно надійним. DBMS_CRYPTO дозволяє вказати необхідний тип доповнення («набивання» – див. таблицю). Більшість компаній для доповнення використовує метод PKCS #5 – для цього потрібно вказати відповідну константу (PAD_PKCS5) з таблиці у значенні параметра typ - DBMS_ RYPTO.PAD_PKCS5.

Константа	Опис
PAD_PKCS5	Доповнення засобами криптографічної системи з спільним ключом (Public Key Cryptography System #5)
PAD_ZERO	Доповнення нулями
PAD_NONE	Доповнення відсутнє. Використовується при впевненості в тому, що довжина вже кратна розміру блоку, що шифрується (кратний 8)

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_6$

***Об'єднання опцій у параметрі* typ**

Припустимо, що ви вибрали наступні опції шифрування:

- *метод доповнення*: доповнення нулями (PAD_ZERO);
- *алгоритм шифрування*: 128-бітний ключ, алгоритм *AES* (ENCRYPT_ AES128);
- *режим шифрування*: блочне шифрування зі зворотним зв'язком по шифртексту *Cipher Feedback* (CHAIN_CFB);

Об'єднуємо ці опції у значенні параметра typ наступним чином:

```
typ => DBMS_CRYPTO.pad_zero + DBMS_CRYPTO.encrypt_aes128
          + DBMS_CRYPTO.chain_cfb
```

Аналогічно можна задавати будь-яку комбінацію опцій функції ENCRYPT. Розглянемо приклад повного виклику функції:

```
DECLARE
   l_enc RAW(2000);
   l_in RAW(2000);
   l_key RAW(2000);
BEGIN
   l_enc :=
      DBMS_CRYPTO.encrypt(src => l_in,
          KEY => l_key,
          typ => DBMS_CRYPTO.pad_zero
                    + DBMS_CRYPTO.encrypt_aes128
                    + DBMS_CRYPTO.chain_cfb
      );
END;
```

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_7$

Для зручності роботи пакет пропонує дві константи з визначеними комбінаціями значень для опцій шифрування, режиму та доповнення (див. Табл.).

Константа	Шифрування	Доповнення	Режим
DES_CBC_PKCS5	ENCRYPT_DES	PAD_PKCS5	CHAIN_CBC
DES3_CBC_PKCS5	ENCRYPT_3DES	PAD_PKCS5	CHAIN_CBC

Якщо треба використовувати алгоритм шифрування DES, доповнення по системі PKCS #5 і режим шифрування CBC, слід задати константи наступним чином:

```
DECLARE
   l_enc RAW(2000);
   l_in RAW(2000);
   l_key RAW(2000);
BEGIN
   l_enc := DBMS_CRYPTO.encrypt(src => l_in,
                  KEY => l_key,
                     typ => DBMS_CRYPTO.des_cbc_pkcs5
            );
END;
```

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_8$

Для Oracle Database 23c

Package Feature	DBMS_CRYPTO
Cryptographic algorithms	DES, 3DES, AES, RC4, 3DES_2KEY
Padding forms	PKCS5, zeroes
Block cipher chaining modes	CBC, CFB, ECB, OFB
Cryptographic hash algorithms	SHA-2 (SHA-256, SHA-384, SHA-512)
Keyed hash (MAC) algorithms	HMAC_SH256, HMAC_SH384, HMAC_SH512
Cryptographic pseudo-random number generator	`RAW`, `NUMBER`, `BINARY_INTEGER`
Database types	`RAW`, `CLOB`, `BLOB`

Шифрування даних у СУБД Oracle (DBMS_CRYPTO)$_9$

Функція **ENCRYPT** приймає вхідні значення у форматі RAW, тому вихідні дані доведеться перетворювати до типу RAW. Зробимо наступне:

```
l_in := UTL_I18N.string_to_raw(p_in_val, 'AL32UTF8');
```

Раніше використовувався вбудований пакет UTL_RAW для перетворення значень типу VARCHAR на RAW. У цьому випадку використовується для такого перетворення функція UTL_I18N.

Чому STRING_TO_RAW, а не UTL_RAW.CAST_TO_RAW?

Функція **ENCRYPT** приймає на вхід значення RAW і, крім того, вимагає використання спеціального набору символів AL32UTF8, який не обов'язково є набором символів бази даних. Отже, фактично необхідно виконати два перетворення:

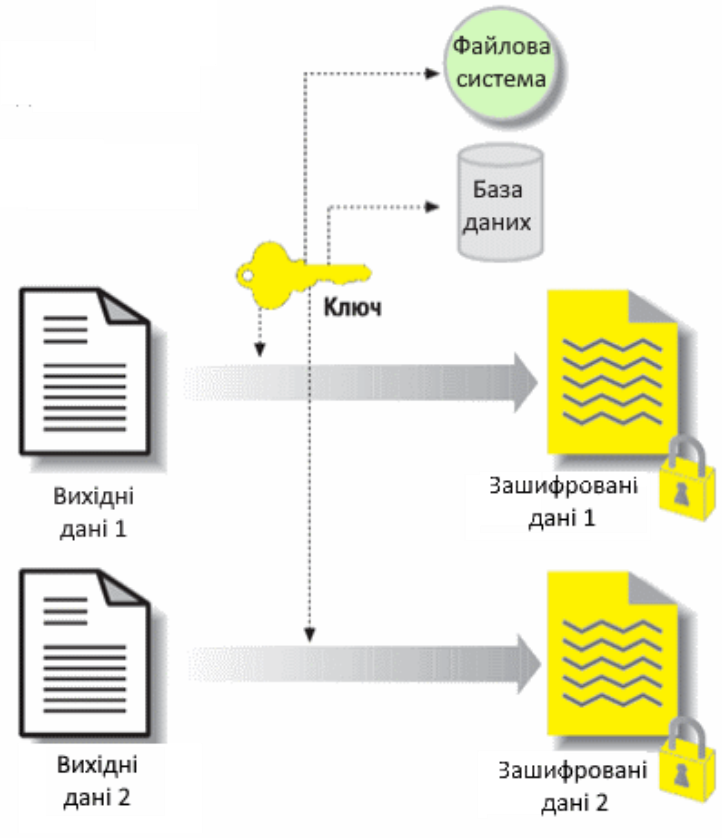
- з поточного набору символів бази даних до набору символів AL32UTF8;
- з VARCHAR2 до RAW.

Функція CAST_TO_RAW не вміє виконувати перетворення набору символів, а функція STRING_TO_RAW вбудованого пакета UTL_i18n може виконати обидва перетворення.

Управління ключами в СУБД Oracle$_1$

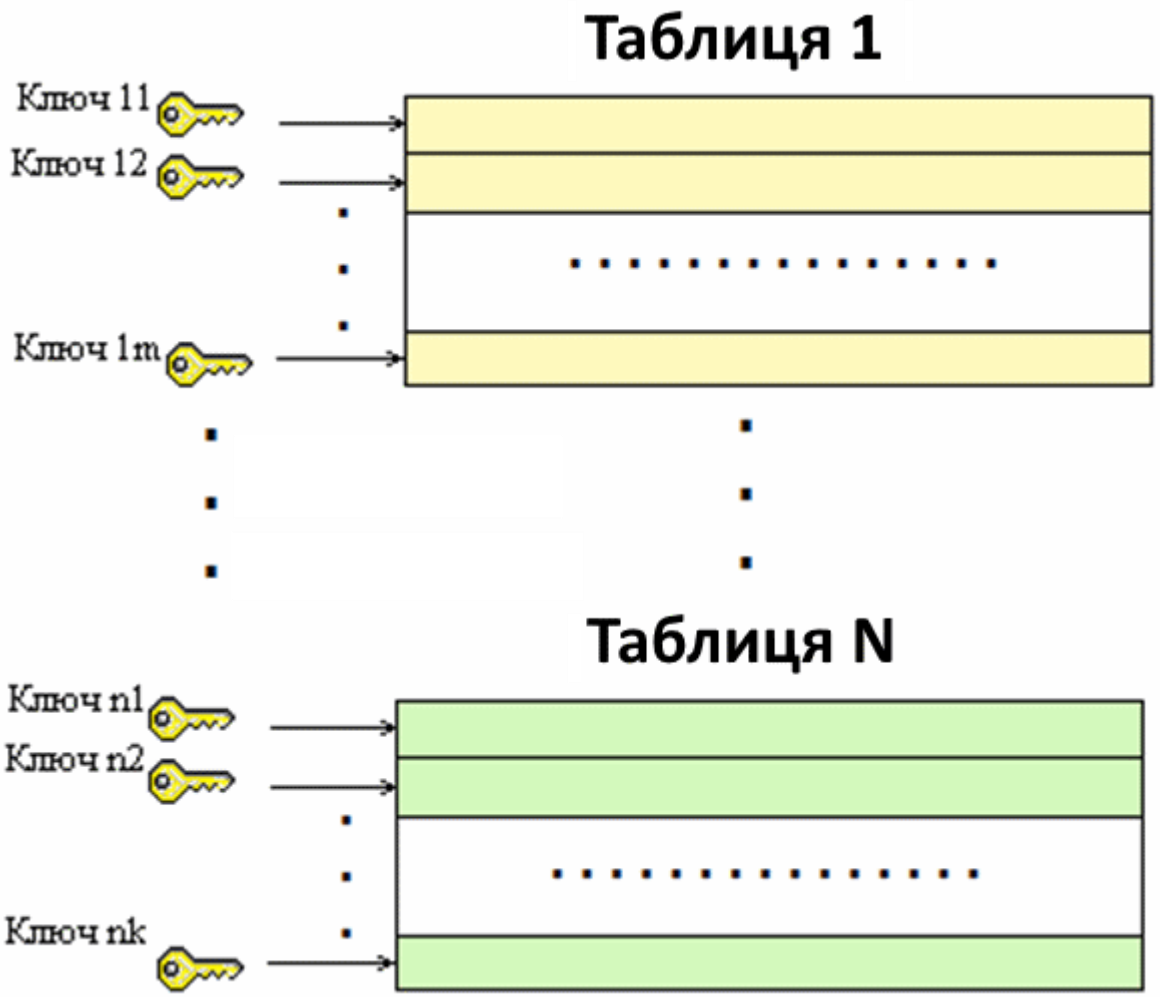
Існує три основні підходи до управління ключами:

1) використання одного й того самого ключа для всієї бази даних;
2) використання різних ключів для різних рядків таблиць БД;
3) комбінований підхід.



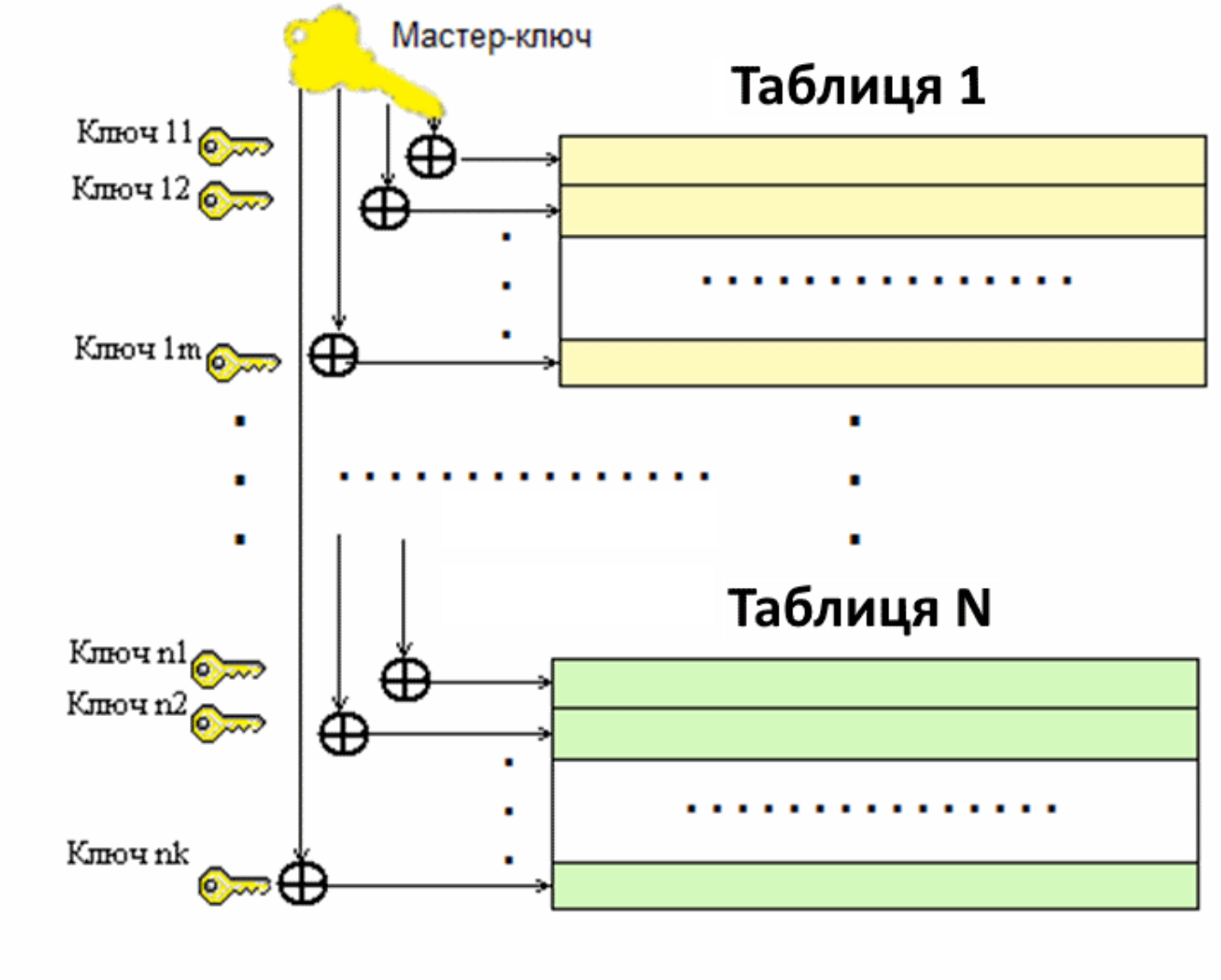
Використання одного ключа для всієї бази даних

Управління ключами в СУБД Oracle,



Використання нового ключа для кожного рядка

Управління ключами в СУБД Oracle$_3$



Використання майстер-ключа

Управління ключами в СУБД Oracle$_4$


```
REM Визначаємо змінну для зберігання зашифрованого значення
VARIABLE enc_val varchar2(2000);
DECLARE
           l_key VARCHAR2(2000) := '0123456789012345';
           l_master_key VARCHAR2(2000) := '& master_key';
           l_in_val VARCHAR2(2000) := 'Секретні дані';
           l_mod NUMBER := DBMS_CRYPTO.encrypt_aes128
                           + DBMS_CRYPTO.chain_cbc
                           + DBMS_CRYPTO.pad_ pkcs5;
           l_enc RAW(2000);
           l_enc_key RAW(2000);
BEGIN
           l_enc_key := UTL_RAW.bit_xor(utl_i18n.string_to_raw(l_key, 'AL32UTF8'),
                           utl_i18n.string_to_raw(l_master_key, 'AL32UTF8')
           );
           l_enc := DBMS_CRYPTO.encrypt(utl_i18n.string_to_raw(l_in_val, 'AL32UTF8'),
                                        l_mod,
                                        l_enc_key
           );
           DBMS_OUTPUT.put_line('Encrypted=' || l_enc);
           :enc_val := RAWTOHEX(l_enc);
END;
/

Enter value for master _key : Master-Key012345
old 3: l_master_key VARCHAR2(2000) := '& master_key' ;
New 3: l_master_key VARCHAR2(2000) := 'Master-Key012345';
Encrypted= 299E749638D5B64E1FED590AFABA5439B313DD0659D44BABFACA2F66804753B38AEA9A
58162B2E9309031BD22A258A83
```

Донецький національний технічний університет

Тема 5. Захист та шифрування даних в БД Oracle. Лекція 7.Прозоре шифрування даних у Oracle

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Мета лекції
- Прозоре шифрування даних у СУБД Oracle
- Криптографічне хешування

Мета лекції

- Прозоре шифрування даних у СУБД Oracle
- Криптографічне хешування

Прозоре шифрування даних у СУБД Oracle$_1$

Якщо ви зберігаєте ключ шифрування та зашифровані дані в базі даних, виникає ще одна потенційна загроза безпеці. При крадіжці дисків, де зберігається вся база даних, всі дані відразу ж розсекречуються.

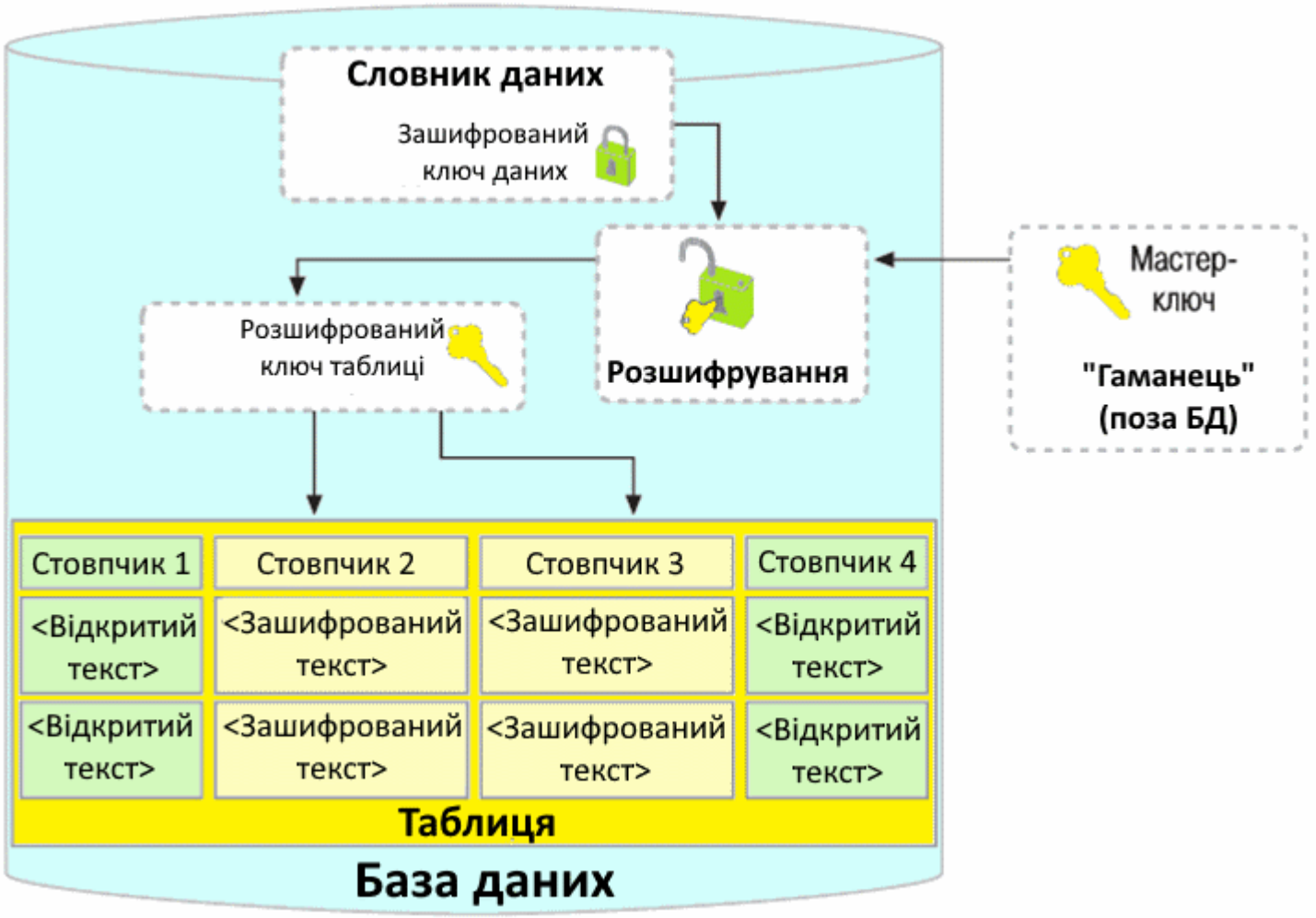
Щоб обійти цю проблему, слід зберігати ключ окремо поза диском, на якому зберігаються зашифровані цим ключем дані. СУБД Oracle дозволяє окремо зберігати зашифровані дані та секретний ключ за допомогою застосування так званого **прозорого шифрування** (*Transparent Data Encryption – TDE*). Чому прозорого, тому що ця функціональна можливість не вимагає будь-якої зміни коду програми.

Суть прозорого шифрування у тому, що використовується поєднання двох ключів:

- *ключа кожної таблиці бази даних*, що є унікальним;
- *майстер-ключа*, який зберігається поза базою даних у гаманці – `Oracle Wallet`.

Не можна використовувати прозоре шифрування в таблицях, що належать користувачеві SYS.

Прозоре шифрування даних у СУБД Oracle$_2$



Модель прозорого шифрування даних

TDE. Процес управління «гаманцями»$_1$

1. *Вибір розташування «гаманця»* .

Перед першим застосуванням TDE необхідно створити «гаманець», у якому зберігатиметься майстер-ключ. За умовчанням «гаманець» створюється в каталозі $ORACLE_BASE/admin/$ORACLE_SID/wallet. Ви можете вибрати інший каталог, вказавши його у файлі SQLNET.ORA. Наприклад, якщо ви хочете, щоб «гаманець» зберігався в каталозі /wallet, додайте наведені нижче рядки у файл SQLNET.ORA.

Для певності будемо вважати, що обрано каталог за замовчуванням.

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE=
(METHOD=file)
      (METHOD_DATA=
(DIRECTORY =/wallet)))
```

TDE. Процес управління «гаманцями»$_2$

2. Встановлення пароля для «гаманця».

Для цього спочатку підключаємося до бази даних із правами `SYS`. Потім створюємо гаманець і вказуємо пароль для доступу до нього. Для чого виконуємо наступний команду:

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "password-wallet";
```

Якщо виникли проблеми з виконанням цього оператора, наприклад, видається помилка:

```
ORA-28368: cannot auto-create wallet
```

спробуйте завершити сеанс і підключитися знову з правами `SYS`.

Цей оператор виконує три дії:

1) створює «гаманець» у каталозі, визначеному у кроці 1 (гаманець Oracle Wallet являє собою двійковий файл `ewallet.p12`, складений за промисловим стандартом PKCS #12);
2) встановлює для "гаманця" пароль - "`password-wallet`";
3) відкриває «гаманець» для збереження та вилучення ключів засобами `TDE`.

Пароль є реєстрозалежним і повинен включатися в подвійні лапки.

TDE. Процес управління «гаманцями»$_3$

3. Відкриття «гаманця».

На попередньому етапі гаманець був відкритий для роботи з ним.

Однак після того, як гаманець створений, вам не доведеться перестворювати його. Після запуску бази даних потрібно буде лише відкрити гаманець, використовуючи встановлений пароль:

```
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password-wallet";
```

або

```
ALTER SYSTEM SET ENCRYPTION WALLET OPEN AUTHENTICATED BY "password-wallet";
```

Закрити «гаманець» можна наступною командою:

```
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;
```

Відкриття «гаманця» необхідне для роботи засобів TDE.

Якщо «гаманець» не відкритий, всі незашифровані стовпці будуть доступні, а зашифровані стовпці – недоступні.

Використання засобів TDE у СУБД Oracle$_1$

Для того, щоб скористатися перевагами TDE, додайте пропозицію ENCRYPT (доступна тільки з версії Oracle 10 g Release 2) для кожного стовпця, що шифрується, в оператор створення вашої таблиці:

```
CREATE TABLE accounts
(
acc_no NUMBER NOT NULL,
first_name VARCHAR2(30) NOT NULL,
last_name VARCHAR2(30) NOT NULL,
SSN VARCHAR2(9) ENCRYPT USING 'AES128',
acc_type VARCHAR2(1) NOT NULL,
folio_id NUMBER ENCRYPT USING 'AES128',
sub_acc_type VARCHAR2(30),
acc_open_dt DATE NOT NULL,
acc_mod_dt DATE,
acc_mgr_id NUMBER
);
```

Використання засобів TDE у СУБД Oracle$_2$

Використання TDE для вже існуючих таблиць

Вище було показано, як можна використовувати TDE для створення нової таблиці. Також можна зашифрувати і стовпець існуючої таблиці. Для шифрування стовпця LAST_NAME таблиці ACCOUNTS використовуємо такий оператор:

```
ALTER TABLE accounts MODIFY (LAST_NAME ENCRYPT);
```

Цей оператор виконує дві дії: створює ключ для стовпця SSN та перетворює всі значення стовпця на зашифрований формат.

Шифрування виконується всередині бази даних. За замовчуванням (спочатку) використовується алгоритм AES зі 192-бітним ключем. Можна вибрати інший алгоритм шифрування, вказавши його назву оператора. Наприклад, щоб вибрати алгоритм AES зі 128-бітним ключем, використовуйте наступний оператор:

```
ALTER TABLE accounts MODIFY (LAST_NAME ENCRYPT USING
'AES128');
```

В якості параметрів можна було б вказати AES128, AES256 або 3DES168 (для трипрохідного алгоритму DES зі 168-бітним ключем).

Використання засобів TDE у СУБД Oracle$_3$

Подивимося на таблицю після шифрування стовпця:

```
DESC argus.accounts;
Name Null? Type
----------------------------------- -------- ------ ------
ACC_NO NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(30)
LAST_NAME NOT NULL VARCHAR2(30) ENCRYPT
SSN VARCHAR2(9) ENCRYPT
ACC_TYPE NOT NULL VARCHAR2(1)
FOLIO_ID NUMBER ENCRYPT
SUB_ACC_TYPE VARCHAR2(30)
ACC_OPEN_DT NOT NULL DATE
ACC_MOD_DT DATE
ACC_MGR_ID NUMBER
```

Зауваження. Не можна вказувати для різних стовпчиків однієї таблиці різні алгоритми шифрування. Так, у цьому випадку не можна використовувати оператор:

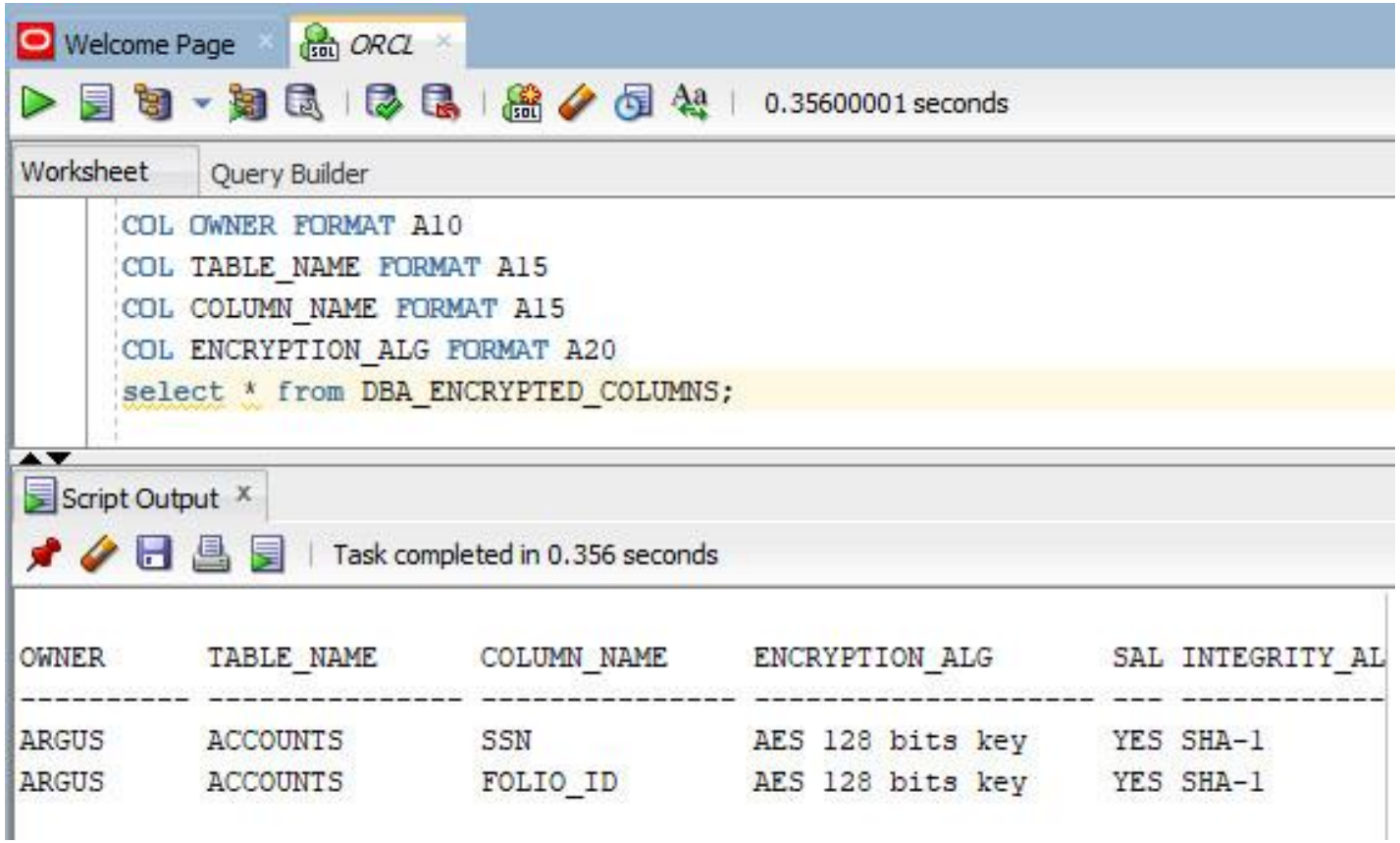
```
ALTER TABLE accounts MODIFY (LAST_NAME ENCRYPT USING 'AES256');
```

так як стовпці `SSN` і `folio_id` використовують алгоритм `AES 128`, і що викликає помилку:

```
ORA-28340: different encryption algoritm has been chosen for the table.
```

Використання засобів TDE у СУБД Oracle$_4$

Для пошуку зашифрованих стовпців бази даних слід використати нове представлення словника даних `DBA_ENCRYPTED_COLUMNS`.



Використання засобів TDE у СУБД Oracle$_5$

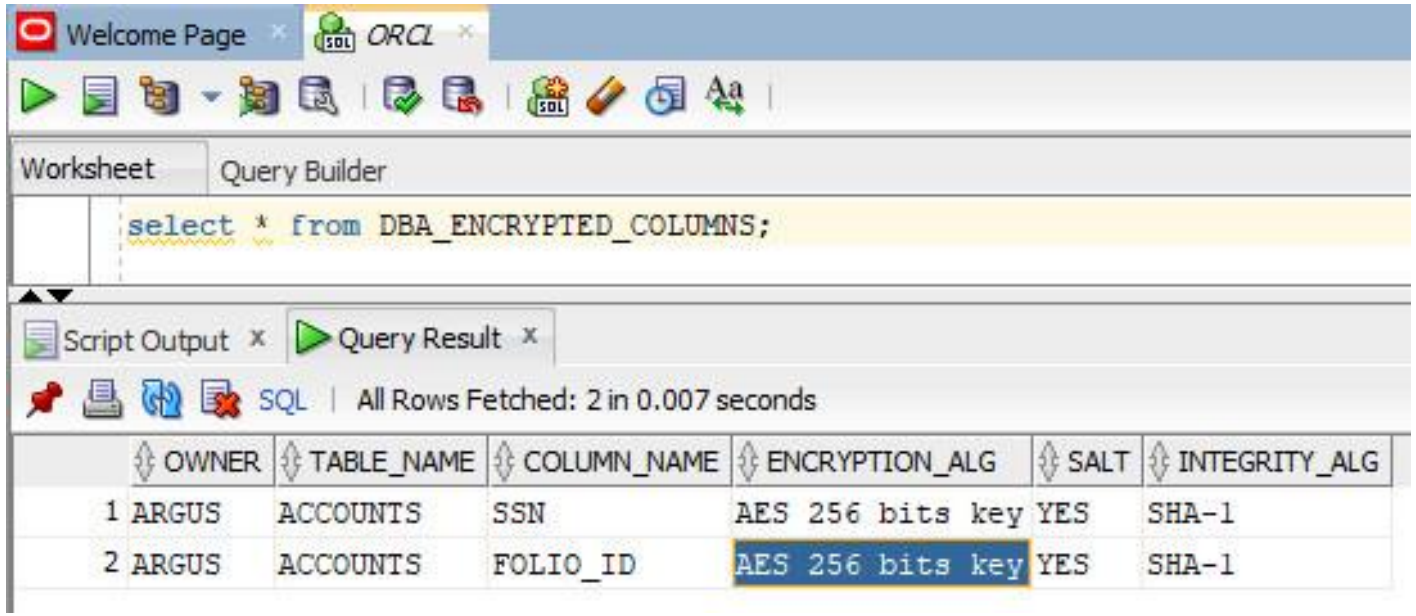
Керування ключами та паролями для TDE

Що буде, якщо хтось якимось чином дізнається ваші ключі `TDE`?

Можна перестворити зашифровані значення, виконавши лише один оператор.

У цьому операторі також можна вибрати інший алгоритм шифрування, наприклад `AES256`:

```
ALTER TABLE accounts REKEY USING 'aes256';
Table altered.
```



Використання засобів TDE у СУБД Oracle$_6$

Фактично команда

```
ALTER TABLE accounts REKEY USING 'aes256';
```

змінює як алгоритм, так і ключ шифрування (також на всю таблицю).
Цей власний ключ шифрування таблиці відразу сам шифрується майстер-ключом з гаманця і запам'ятовується для неї в БД, в системній таблиці `ENC$`. Звідти він братиметься щоразу при зашифруванні/розшифруванні значень полів рядків таблиці.

Так, при зверненні до зашифрованого поля таблиці СУБД візьме з `ENC$` зашифрований ключ цієї таблиці, розшифрує його майстер-ключом з гаманця і вже відновленим ключем таблиці розшифрує значення поля.
Якщо потрібно змінити зашифровані значення, що зберігаються, можна просто виконати наступний оператор:

```
ALTER TABLE closed REKEY;
```

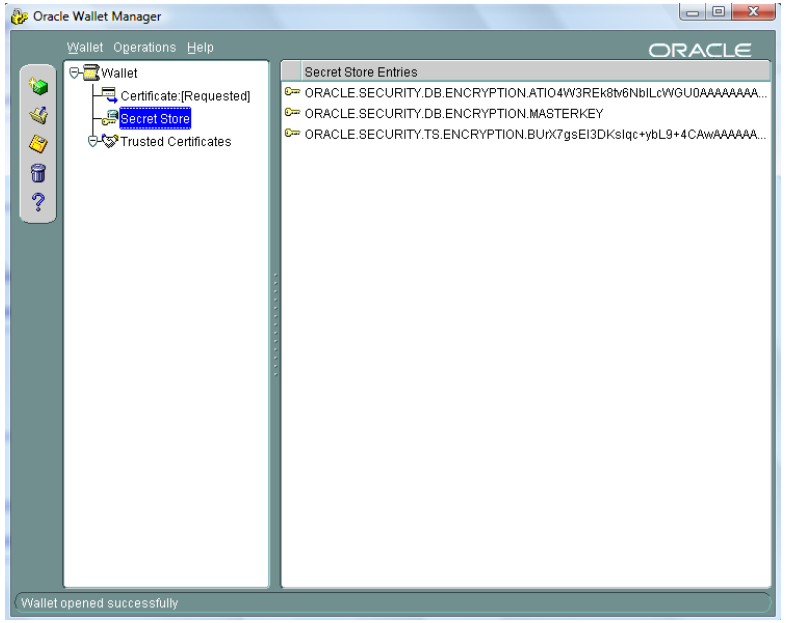
Використання засобів TDE у СУБД Oracle$_7$

Якщо хтось дізнається пароль «гаманця», його можна змінити за допомогою Oracle Wallet Manager (якщо він встановлений). Запустивши диспетчер «гаманця», виберіть у головному меню *Wallet→Open* і вкажіть місце зберігання «гаманця» та його пароль. Щоб змінити пароль, виберіть *Wallet → Change Password*.

Майте на увазі, що зміна пароля не призводить до зміни ключів.

У разі відсутності OWM можна скористатися наступною командою:

```
orapki wallet change_pwd -wallet
$ORACLE_BASE/ADMIN/ORCL/WALLET
```



Використання засобів TDE у СУБД Oracle$_8$

Додамо «прив'язку»

Шифрування призначене для приховування даних, але буває так, що зашифровані дані легко вгадати через повторення вихідних даних.

У цьому випадку зашифровані значення також будуть однаковими.

Для запобігання таким ситуаціям до даних додається, так звана «*прив'язка*» або її ще називають *сіль* (`salt`), завдяки якій зашифровані значення відрізнятимуться навіть для однакових вихідних даних. `TDE` використовує стандартну *сіль* (див. запит до подання словника даних)

```
select * from DBA_ENCRYPTED_COLUMNS;
```

При цьому слід пам'ятати, що в деяких випадках структури даних можуть сприяти підвищенню продуктивності бази даних, а додавання солі може її погіршити.

Від застосування прив'язки можна відмовитись:

```
ALTER TABLE accounts MODIFY (SSN ENCRYPT NO SALT);
```

Використання засобів TDE у СУБД Oracle$_9$

Подивитися стан та місцезнаходження гаманця можна тільки, починаючи з 11 версії Oracle у таблиці V$ENCRYPTION_WALLET.

```
COL WRL_PARAMETER FORMAT A55;
select * from V$ENCRYPTION_WALLET;

WRL_TYPE WRL_PARAMETER STATUS
---------- ---------------------------------------- ----- ------
-
file d:\app\HOST-name\admin\ora-sid\wallet OPEN
```

Зауваження. Використання засобів TDE неможливе для стовпців, які мають одну з наведених нижче властивостей:

✓тип даних BLOB чи CLOB;

✓використання в індексах, відмінних від звичайних індексів b-tree, таких як bitmap-індекси, індекси на основі функцій тощо;

✓використання у ключах секціонування.

Відсутність можливості використання засобів TDE у подібних випадках є ще однією причиною, через яку TDE не може використовуватися для всіх видів шифрування.

Використання засобів TDE у СУБД Oracle$_{10}$

Створення табличних просторів із зашифрованими даними

Коли виникає необхідність шифрувати дані багатьох стовпців у багатьох таблицях, традиційним способом, розглянутим вище це робити не завжди зручно та раціонально. З 11 версії Oracle для таких випадків є узагальнене рішення: зашифрування всіх даних, розміщених у конкретному табличному просторі. Така властивість табличного простору незмінна і вказується один раз при його створенні, наприклад:

```
CREATE TABLESPACE encrypted_data
DATAFILE 'd:\app\hostname\oradata\orasid\encrf.dat' SIZE
1M
ENCRYPTION USING '3DES168'
DEFAULT STORAGE (ENCRYPT);

Tablespace created.
```

Воно відображається у полі `ENCRYPTED` таблиці `DBA_TABLESPACES`.

Криптографічне хешування у СУБД Oracle$_1$

Відмінність хешування від шифрування у цьому, що хешування – це однонаправлений процес. Можна розшифрувати зашифровані дані, але розхешувати хеш-значення неможливо. Якщо кілька разів хешується один і той же самий елемент даних, результат буде незмінним за будь-якої кількості виконань. Якщо дані якимось чином змінюються, хеш-значення, що генерується, зміниться, вказуючи на «псування» даних.

Хешування до версії 20c

```
DECLARE
        l_hash VARCHAR2(2000);
        l_in_val VARCHAR2(2000);
BEGIN
        l_in_val := 'Account Balance is 12345.67';
        l_hash := DBMS_OBFUSCATION_TOOLKIT.MD5(input_string => l_in_val
                        );
        l_hash := RAWTOHEX(UTL_RAW.cast_to_raw(l_hash));
        DBMS_OUTPUT.put_line('Hashed Value = ' || l_hash);
END;
/
Hashed Value = A09308E539C35C97CD612E918BA58B4C
```

Криптографічне хешування у СУБД Oracle$_2$

Як можна помітити хешування має дві важливі відмінності від шифрування:

- при хешуванні вхідний рядок не потрібно доповнювати до певної довжини, як це робиться під час шифрування;
- при хешуванні не використовується ключ. А якщо ключа немає, його не потрібно зберігати або вводити, що робить систему хешування надзвичайно простою.

Теоретично можливе отримання одного й того ж хеш-значення для двох різних вхідних значень. Однак, використовуючи такі загальнопоширені алгоритми, як MD5 та SHA-1, забезпечується надзвичайно мала ймовірність хеш-конфлікту – близько 1/1038 (залежно від обраного алгоритму). Якщо не можна допустити наявності навіть такої малої ймовірності, доведеться реалізувати логіку вирішення конфліктів для хеш-функцій.

Криптографічне хешування у СУБД Oracle$_3$

Хешування з версії Oracle 10g

На сьогоднішній день алгоритми хешування MD5 вважається недостатньо надійним для сучасного захисту даних, і замість нього часто використовується алгоритм SHA-1. Але алгоритм SHA-1 не підтримується у пакеті DBMS_OBFUSCATION_TOOLKIT. Він доступний лише, починаючи з версії Oracle 10g у пакеті DBMS_CRYPTO функція HASH. Оголошення цієї функції :

```
DBMS_CRYPTO.HASH(
   src in raw,
   typ in pls_integer )
return raw;
```

Функція HASH приймає на вхід тільки значення типу RAW, тому необхідно перетворити вхідний символьний рядок до типу RAW, що реально робиться так само, як і для шифрування:

```
l_in := utl_i18n.string_to_raw(p_in_val, 'AL32UTF8');
```

Криптографічне хешування у СУБД Oracle$_4$

Name	Description
HASH_MD4	Produces a 128-bit hash, or message digest of the input message.
HASH_MD5	Also produces a 128-bit hash, but is more complex than MD4.
HASH_SH1	Secure Hash Algorithm (SHA-1). Produces a 160-bit hash.
HASH_SH256	SHA-2, produces a 256-bit hash.
HASH_SH384	SHA-2, produces a 384-bit hash.
HASH_SH512	SHA-2, produces a 512-bit hash.

Наприклад, щоб отримати хеш-значення для змінної типу RAW, складемо функцію:

```
CREATE OR REPLACE FUNCTION get_sha1_hash_val(p_in RAW)
RETURN RAW
IS
        l_hash RAW(4000);
BEGIN
        l_hash := DBMS_CRYPTO.HASH(src => p_in, typ =>
                        DBMS_CRYPTO.HASH_SH1);
        RETURN l_hash;
END;
```

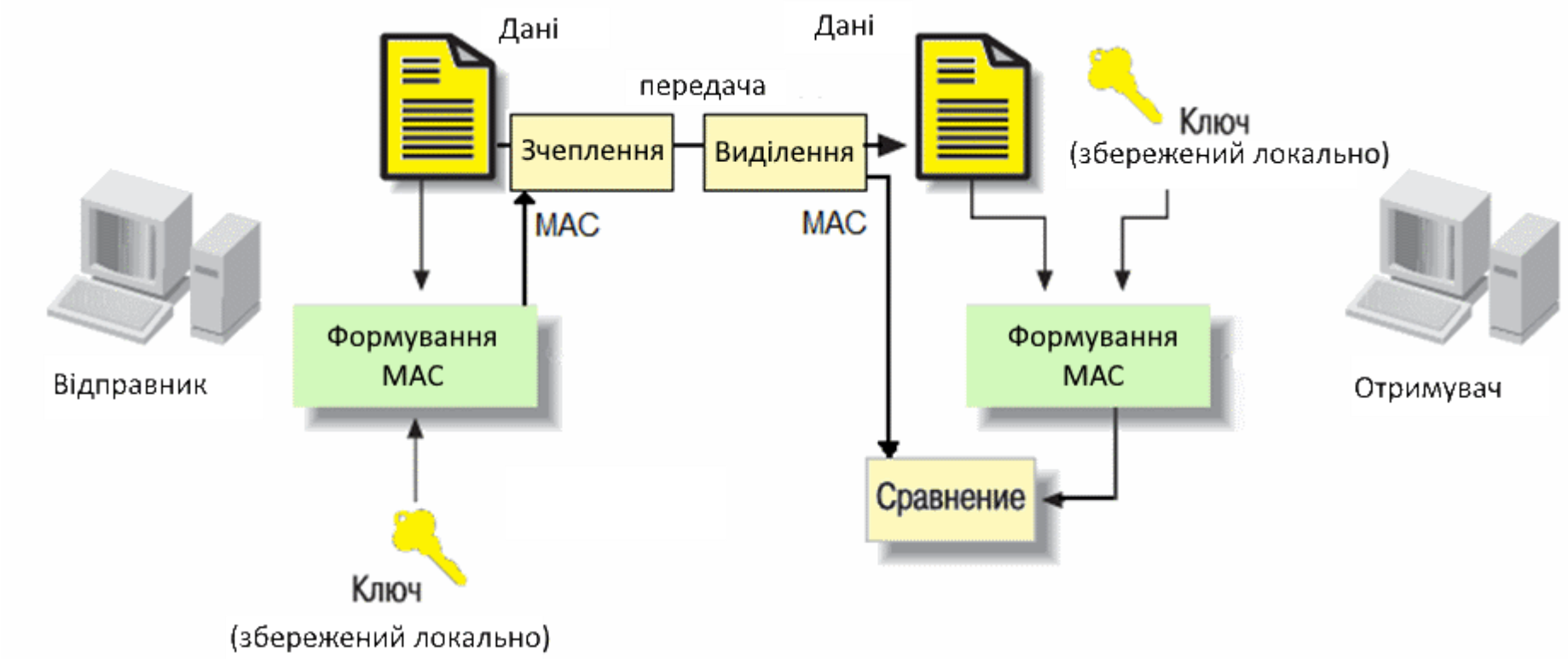
Криптографічне хешування у СУБД Oracle$_5$

Код аутентифікації повідомлення (MAC)

Розглянутий раніше метод хешування дуже корисний, але має деякі недоліки:

- перевірити справжність переданих даних за допомогою хеш-функції може будь-хто. Це може бути неприпустимим для деяких систем підвищеної безпеки, які передбачають, що автентичність повідомлення чи даних перевіряє лише певний одержувач;
- дізнавшись алгоритм хешування, зловмисник може обчислити правильне хеш-значення та підмінити їм реальне, приховавши тим самим зміну даних;
- з причини, зазначеної в попередньому пункті, зберігання хеш-значення разом з даними не є безпечним. Кожен, хто має привілей зміну таблиці, зможе змінити і хеш-значення. Аналогічно хтось може згенерувати хеш-значення та змінити дані «в дорозі». Тому хеш-значення не повинно супроводжувати дані, а обов'язково має передаватися окремо, що ускладнює систему.

Криптографічне хешування у СУБД Oracle$_6$



Використання коду автентифікації повідомлення

Криптографічне хешування у СУБД Oracle$_7$

Як і для хешування, `DBMS.CRYPTO` підтримує різні алгоритми `MAC` (див. таблицю).

Name	Description
`HMAC_MD5`	Same as MD5 hash function, except it requires a secret key to verify the hash value.
`HMAC_SH1`	Same as SHA hash function, except it requires a secret key to verify the hash value. Complies with IETF RFC 2104 standard.
`HMAC_SH256`	Same as SHA-2 256-bit hash function, except it requires a secret key to verify the hash value.
`HMAC_SH384`	Same as SHA-2 384-bit hash function, except it requires a secret key to verify the hash value.
`HMAC_SH512`	Same as SHA-2 512-bit hash function, except it requires a secret key to verify the hash value.

Криптографічне хешування у СУБД Oracle$_7$

Приклад функції, що обчислює значення *MAC* для різних алгоритмів:

```
CREATE OR REPLACE FUNCTION get_mac_val(
   p_in_val      IN   VARCHAR2,
   p_key         IN   VARCHAR2,
   p_algorithm   IN   VARCHAR2 := 'SH1'
)
RETURN VARCHAR2
IS
l_mac_val    RAW(4000);
l_key        RAW(4000);
l_mac_algo   PLS_INTEGER;
l_in         RAW(4000);
l_ret        VARCHAR2 (4000);
BEGIN
   l_mac_algo :=
      CASE p_algorithm
         WHEN 'SH1'
            THEN DBMS_CRYPTO.hmac_sh1
         WHEN 'MD5'
            THEN DBMS_CRYPTO.hmac_md5
      END;
l_in := utl_i18n.string_to_raw(p_in_val, 'AL32UTF8');
l_key := utl_i18n.string_to_raw(p_key, 'AL32UTF8');
l_mac_val := DBMS_CRYPTO.mac(src => l_in, typ => l_mac_algo, key=>l_key);
l_ret := RAWTOHEX(l_mac_val);
RETURN l_ret;
END;
```

Донецький національний технічний університет

Тема 6. Моніторинг та аудит безпеки в БД Oracle.
Лекція 8. Техніка аудиту в Oracle

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач : проф. Дорогий Я.Ю.

Лекційні питання

- Мета лекції
- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит

Мета лекції

- Ознайомитися з загальними поняттями аудиту
- Ознайомитися з процесом аудиту застосунків
- Розглянути підхід аудиту на основі тригерів
- Ознайомитися з методикою аудиту користувачів системи
- Розглянути процес стандартного аудиту

Методи аудиту для Oracle

- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит
- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Огляд аудиту

- Що перевіряти
- Як провести аудит
- Як використовувати записи аудиту
- Вирішення проблем продуктивності
- Аудит також може показати шаблони роботи, частоту використання тощо
  - Дані можуть бути «вкрадені» без відомості
- Проведення вибору аудиту
  - Загальний аудит може негативно вплинути на продуктивність
  - Також створюються масивні, складні для обробки журналів аудиту
- Останній етап у циклі безпеки – ніколи не обходьте його

Методи аудиту для Oracle

- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит
- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Аудит застосунків

- Запрограмований у програмі
  - Часто реалізується в застосунках сторонніх розробників
- Часто робиться для переносимості між СУБД або коли аудит бази даних недостатньо зрозумілий
- Усі аспекти можна перевірити
  - Надзвичайно гнучкий і розширюваний
- Обслуговування коду може бути обтяжливим
- Великі програми часто стають мішенню для хакерів
- Програму можна обійти, що зробить аудит марним
  - Це орієнтовано на застосування

Приклад аудиту програми$_1$


```
CREATE TABLE emp_under_audit AS SELECT * FROM empcopy;
```

- Створіть таблицю (`aud_emp`), призначену для збору інформації аудиту для таблиці `emp_under_audit`

```
CREATE TABLE aud_emp (
     username    VARCHAR2(30)
    ,action      VARCHAR2(12)
    ,empno       NUMBER(4),
    ,column_name VARCHAR2(255)
    ,call_stack  VARCHAR2(4000)
    ,client_id   VARCHAR2(255)
    ,old_value   VARCHAR2(25)
    ,new_value   VARCHAR2(25)
    ,action_date DATE);
```

Приклад аудиту програми$_2$

- Створіть процедуру для *генерації* аудиторської інформації

```
CREATE OR REPLACE PROCEDURE proc_audit_emp (
   pi_username    IN VARCHAR2
  ,pi_action      IN VARCHAR2
  ,pi_empno       IN NUMBER
  ,pi_column_name IN VARCHAR2
  ,pi_old_value   IN VARCHAR2
  ,pi_new_value   IN VARCHAR2)
AS
BEGIN
   INSERT INTO aud_emp
      (username,action,empno,column_name,call_stack,
       client_id,old_value,new_value,action_date)
   VALUES (pi_username
          ,pi_action
          ,pi_empno
          ,pi_column_name
          ,dbms_utility.format_call_stack
          ,sys_context('userenv','client_identifier')
          ,pi_old_value
          ,pi_new_value
          ,sysdate);
END;
```

Приклад аудиту програми$_3$

- Створіть процедуру для *відображення та форматування* інформації аудиту

```
CREATE OR REPLACE PROCEDURE proc_format_aud_emp AS
BEGIN
   FOR r IN (SELECT *
               FROM aud_emp
               ORDER BY action_date DESC)
   LOOP
      dbms_output.put_line('User:       '||r.username);
      dbms_output.put_line('Client ID: '||r.client_id);
      dbms_output.put_line('Action:     '||r.action);
      dbms_output.put_line('Empno:      '||r.empno);
      dbms_output.put_line('Column:
'||r.column_name);
      dbms_output.put_line('Old Value: '||r.old_value);
      dbms_output.put_line('New Value: '||r.new_value);
      dbms_output.put_line('Date:       '||
          TO_CHAR(r.action_date,'MON-DD-YYYY
HH24:MI'));
   END LOOP;
END;
```

Приклад аудиту програми$_4$

- Створіть процедуру подання заявки, яка перевіряється

```
CREATE OR REPLACE PROCEDURE proc_update_sal(
   pi_empno IN NUMBER,
   pi_salary IN NUMBER)
AS
   v_old_sal VARCHAR2(25);
BEGIN
   SELECT sal INTO v_old_sal FROM emp_under_audit
   WHERE empno = p_empno
   FOR UPDATE;
   UPDATE emp_under_audit SET sal = pi_salary
   WHERE empno = pi_empno;
   proc_audit_emp
      (pi_username => user
      ,pi_action => 'UPDATE'
      ,pi_empno => pi_empno
      ,pi_column_name => 'SAL'
      ,pi_old_value => v_old_sal
      ,pi_new_value => pi_salary);
END;
/
```

Приклад аудиту програми$_5$

- Запустіть програму, виконайте процедуру оновлення та перевірте зміни

```
BEGIN
   proc_update_sal(p_empno => 7369,p_salary => 950);
   proc_format_aud_emp;
END;
/
```

➢ Показати отриманий стек викликів

```
SELECT username,call_stack FROM aud_emp;

USERNAME CALL_STACK
-------- --------------------------------
SCOTT    ----- PL/SQL Call Stack -----
           object      line  object
           handle    number  name
         664B1434         1  anonymous block
         6A1DFC34        10  procedure SCOTT.PROC_AUDIT_EMP
         66614FA0        11  procedure SCOTT.PROC_UPDATE_SAL
         6651D620         2  anonymous block
```

Приклад аудиту програми$_6$

- Показати отриману аудиторську інформацію
  - Ідентифікатор `клієнта` повертає IP-адресу, якщо користувач був віддаленим
  - Можна охопити набагато більше про контекст користувача

```
BEGIN
   proc_format_aud_emp;
END;
/

User:       SMITH
Client ID: 127.0.0.1
Action:     UPDATE
Empno:      7369
Column:     SAL
Old Value: 800
New Value: 950
Date:       SEP-07-2012 18:37
```

Методи аудиту для Oracle

- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит
- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Аудит на основі тригерів

- Орієнтований на базу даних – дуже популярний
  - Іноді його називають аудитом на основі вартості
- Можна використовувати для подій `INSERT`, `UPDATE`, `DELETE` (але не для подій `SELECT`)
- Прозорий для всіх програм
- Гнучкий і розширюваний
- Не завжди спрацьовує
  - Не спрацьовує для `TRUNCATE`
- Неможливо отримати параметри – обмежено значеннями стовпців
- Необхідно створити для кожного об’єкта
  - Можна викликати загальні процедури

Аудит на основі тригерів – простий приклад

- Фіксація зміни зарплати, хто вніс зміни за допомогою простого тригера

```
CREATE TRIGGER trg_a_idu_r_emp_sal
AFTER INSERT OR DELETE OR UPDATE OF sal ON emp
FOR EACH ROW
BEGIN
   IF (:NEW.sal > :OLD.sal * 1.10)
      THEN INSERT INTO emp_sal_audit VALUES (:OLD.empno
                                            ,:OLD.sal
                                            ,:NEW.sal
                                            ,user
                                            ,sysdate);
   END IF;
END;
/
```

- Тригери не можуть захопити оператор ініціювання
- Не можна використовувати для визначення дій попередження
- Точний аудит може бути кращим варіантом

Тригер для заповнення таблиці аудиту

- Тригер спрацьовує під час оновлень `sal`
  - Виконує `proc_audit_emp` для заповнення попередньо створеної таблиці аудиту (`aud_emp`)

```
CREATE OR REPLACE TRIGGER trg_b_u_r_emp_copy_sal
BEFORE UPDATE OF sal
ON emp_copy
FOR EACH ROW
DECLARE
BEGIN
   proc_audit_emp (p_username => user,
                   ,p_action => 'UPDATE'
                   ,p_empno => :OLD.empno
                   ,p_column_name => 'SAL'
                   ,p_old_value => TO_CHAR(:OLD.sal)
                   ,p_new_value => TO_CHAR(:NEW.sal));
END;
/
```

Запуск тригера аудиту

- Сміт виконує оновлення, яке запускає тригер

```
SMITH> UPDATE scott.emp_copy
  2  SET sal = sal*1.1
  3  WHERE job = 'ANALYST';
```

- **Стек викликів показує спрацьовування тригера**

```
SCOTT> SELECT DISTINCT call_stack FROM aud_emp;

CALL_STACK
--------------------------------------------------
----- PL/SQL Call Stack -----
  object      line  object
  handle    number  name
665C3F84         1  anonymous block
6675288C        10  procedure SCOTT.PROC_AUDIT_EMP
6A297C30         3  SCOTT.TRG_B_U_R_EMP_COPY_SAL
```

Ініційовані аудитом записи

- Інформація аудиту показує, що два записи зазнають оновлення

```
SCOTT> BEGIN
  2     proc_format_aud_emp;
  3  END;
  4  /

User:           SMITH
Client ID:
Action:         UPDATE
Empno:          7902
Column:         SAL
Old Value:      3000
New Value:      3300
Date:           NOV-11-2023 19:37

User:           SMITH
Client ID:
Action:         UPDATE
Empno:          7788
Column:         SAL
Old Value:      3000
New Value:      3300
Date:           NOV-11-2023 19:37
```

Обробка відкату - автономні транзакції

- Сценарій
  - Користувач робить оновлення, перевіряє значення, а потім відкочує транзакцію
  - Записи в таблиці аудиту також буде відкачено
    - Втрата аудиторської інформації

- Неможливо розмістити `COMMIT` у тригері
  - Але можна використовувати автономні транзакції
    - Дозволяє фіксувати дії тригерів незалежно від оператора запуску
    - Зберігає інформацію аудиту під час відкату

Обробка відкату – автономні транзакції (продовження)

- Усі оновлення будуть перевірені

```
CREATE OR REPLACE PROCEDURE proc_audit_emp (
   p_username    IN VARCHAR2
  ,p_action      IN VARCHAR2
  ,p_empno       IN NUMBER
  ,p_column_name IN VARCHAR2
  ,p_old_value   IN VARCHAR2
  ,p_new_value   IN VARCHAR2)
AS
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   INSERT INTO aud_emp
      (username,action,empno,column_name,call_stack,
       client_id,old_value,new_value,action_date)
   VALUES (p_username
          ,p_action
          ,p_empno
          ,p_column_name
          ,dbms_utility.format_call_stack
          ,sys_context('userenv','client_identifier')
          ,p_old_value
          ,p_new_value
          ,sysdate);
   COMMIT;
END;
```

Методи аудиту для Oracle

- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит
- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Обов'язковий аудит бази даних

- Обов'язковий аудит завжди веде записи
  - Запуск
  - Зупинка
  - Вхід і вихід користувача з привілеями `SYSDBA` і `SYSOPER`
    - Показує, якщо адміністратор вимкнув аудит

    `AUDIT_TRAIL = FALSE` (або `NONE`)

- Записи повинні зберігатися в операційній системі, оскільки база даних недоступна під час запуску або зупинки
  - У Windows у журналах подій
  - У Linux і unix у `$ORACLE_HOME/rdbms/audit`

Аудит користувача SYS за допомогою AUDIT_SYS_OPERATIONS


```
ALTER SYSTEM SET AUDIT_SYS_OPERATIONS = TRUE
SCOPE = SPFILE;
```

- Дії користувачів, які мають SYSDBA або SYSOPER, записуються у файли ОС (у відповідних випадках XML), а не в базу даних
  - Усі успішні дії sys SQL верхнього рівня перевіряються
    - Можна побачити в засобі перегляду подій Windows (не в aud$)
  - Ці користувачі бази даних не повинні мати доступу до записів аудиту
- Параметр свідомо не динамічний
  - База даних має бути «відкинута», щоб змінити її значення
  - Перешкоджає адміністратору баз даних просто вимкнути аудит, виконати зловмисну дію, а потім знову ввімкнути аудит
    - Відкид бази даних фіксує відключення аудиту

Приклад системного аудиту


```
CONNECT / AS SYSDBA
ALTER SYSTEM FLUSH SHARED_POOL;
UPDATE scott.emp SET sal=1000 WHERE ename='SCOTT';
```

- Якщо `sys аудит` ввімкнено, оператори `ALTER SYSTEM` і `UPDATE` відображаються у файлі аудиту ОС або журналі подій:

```
Audit trail: LENGTH: '177' ACTION :[7] 'CONNECT'
DATABASE USER:[1] '/' PRIVILEGE :[6] 'SYSDBA' CLIENT
USER:[10] 'UNV\in8308' CLIENT TERMINAL:[14] 'WXPLT-
ITR12680' STATUS:[1] '0' DBID:[10] '1318485259' .

Audit trail: LENGTH: '201' ACTION :[30] 'ALTER SYSTEM
FLUSH SHARED_POOL' DATABASE USER:[1] '/' PRIVILEGE
:[6] 'SYSDBA' CLIENT USER:[10] 'UNV\in8308' CLIENT
TERMINAL:[14] 'WXPLT-ITR12680' STATUS:[1] '0'
DBID:[10] '1318485259' .

Audit trail: LENGTH: '220' ACTION :[49] 'UPDATE
scott.emp SET sal=1000 WHERE ename='SCOTT'' DATABASE
USER:[1] '/' PRIVILEGE :[6] 'SYSDBA' CLIENT USER:[10]
'UNV\in8308' CLIENT TERMINAL:[14] 'WXPLT-ITR12680'
STATUS:[1] '0' DBID:[10] '1318485259' .
```

Методи аудиту для Oracle

- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит
- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Значення AUDIT_TRAIL для стандартного аудиту

Значення AUDIT_TRAIL	**Аудиторська діяльність**
FALSE (за замовчуванням) (NONE на Oracle10 *g*)	Немає стандартної аудиторської діяльності
OC	Записи записуються в ОС Зробіть це з тих самих причин, що й для AUDIT_SYS_OPERATIONS Доступно, коли база даних не працює
XML (або XML,EXTENDED)	Записи записуються як XML Нове подання (V$XML_AUDIT_TRAIL) показує ці записи у реляційному форматі та робить їх "доступними для запиту " через SQL EXTENDED викликає перевірку тексту sql і значень змінних прив'язки
DB (або DB,EXTENDED)	Записи записуються в таблицю бази даних (sys.aud$) Полегшує створення звітів

- Журнал аудиту *не* зберігає значення даних

Встановлення стандартного аудиту


```
ALTER SYSTEM SET AUDIT_TRAIL = DB,EXTENDED SCOPE =
SPFILE;
```

- Параметр свідомо не динамічний
  - База даних має бути «відскочена», щоб змінити її значення
  - Установіть це значення під час створення бази даних, щоб уникнути «відмов» бази даних пізніше
- Якщо `AUDIT_FILE_DEST` не вказано, розташуванням ОС за замовчуванням є
  - Unix

    `$ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump`
  - Windows

    `$ORACLE_BASE\admin\$DB_UNIQUE_NAME\adump`

Обсяг діяльності з аудиту – стандартний аудит

- Конкретні об'єкти
- Виконання процедур
- Використання системних привілеїв
- Конкретні користувачі
- Успішні та/або невдалі дії
- За дію або за сеанс (за сеанс нереалістичний варіант для 11g)
- Дозволяє зосередити аудиторську діяльність
  - Важливо точно налаштувати це, щоб уникнути проблем із продуктивністю та пам’яттю
- Дозволяє контролювати привілейованих користувачів – DBAS тощо.

Аудит підключень$_1$

- Потрібно знати, хто і коли
  - Більшість збоїв пов’язані з діяльністю людини
  - Непросто для користувачів застосунків, які використовують пули підключень
- Генерує багато записів
  - Потрібен достатній дисковий простір і політика очищення
- Для перевірки з’єднань необхідно встановити *два критерії*
  - Переконайтеся, що `AUDIT_TRAIL = DB,EXTENDED`
  - Під користувачем system введіть команду

```
AUDIT SESSION;
```

  - Аудит зв’язків лише для Скотта та Сміта

```
AUDIT SESSION BY scott,smith;
```

- Немає особливого захисту від відсутності цієї інформації

Аудит підключень$_2$

- Сценарій звіту про аудит входів і виходів користувачів

```
BEGIN
   FOR r IN
      (SELECT username
             ,action_name
             ,TO_CHAR(timestamp, 'DD-MON HH24:MI') LOGON
             ,TO_CHAR(logoff_time, 'DD-MON HH24:MI') LOGOFF
             ,priv_used
             ,comment_text
       FROM dba_audit_trail)
   LOOP
      dbms_output.put_line('User:      '||r.username);
      dbms_output.put_line('Action:    '||r.action_name);
      dbms_output.put_line('Logon:     '||r.LOGON);
      dbms_output.put_line('Logoff:    '||r.LOGOFF);
      dbms_output.put_line('Priv:      '||r.priv_used);
      dbms_output.put_line('Comments:  '||r.comment_text);
      dbms_output.put_line('-----End of audit record-----');
   END LOOP;
END;
```

Звіт про аудит

- Користувач `scott` створив сеанс, а потім майже одразу вийшов

```
SYS>/
User:          SCOTT
Action:        LOGON
Logon:         12-NOV 09:24
Logoff:
Priv:          CREATE SESSION
Comments:      Authenticated by: DATABASE; Client address:
(ADDRESS=(PROTOCOL=tcp)(HOST=169.254.207.135)(PORT=2817))
------End of audit record------

PL/SQL procedure successfully completed.

SYS> /
User:          SCOTT
Action:        LOGOFF
Logon:         12-NOV 09:24
Logoff:        12-NOV 09:25
Priv:          CREATE SESSION
Comments:      Authenticated by: DATABASE; Client address:
(ADDRESS=(PROTOCOL=tcp)(HOST=169.254.207.135)(PORT=2817))
---------End of audit record---------
```

- На Oracle11*g* можна побачити активність `DBSNMP` і `SYSMAN`

Аудит операторів$_1$

- Використання будь-якого типу оператора SQL можна перевірити
  - Може ґрунтуватися на тому, чи є оператор успішним, невдалим...
- Приклади аудиту на основі операторів, що впливають на типи об'єктів
  - Аудити `CREATE` , `ALTER` , `DROP` будь-якої ролі чи таблиці

```
AUDIT ROLE,TABLE;
```

  - Аудит операторів `CREATE TABLE`

```
AUDIT CREATE TABLE;
```

  - Перевіряє оператори `ALTER TABLE` лише тоді, коли вони невдалі

```
AUDIT ALTER TABLE WHENEVER NOT SUCCESSFUL;
```

Аудит операторів$_2$

- Аудит усіх невдалих операторів `SELECT` , `INSERT` , `DELETE` для всіх таблиць і будь-якої невдалої спроби виконання процедури

```
AUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
EXECUTE PROCEDURE BY ACCESS WHENEVER NOT SUCCESSFUL;
```

- **Можна вказати для кожного користувача**

```
AUDIT DELETE TABLE BY tt BY ACCESS;
```

Аудит відстежень операторів

- Аудит на рівні оператора показано в `dba_stmt_audit_opts`

```
AUDIT CREATE EXTERNAL JOB BY tt;

SELECT * FROM dba_stmt_audit_opts;

USER_NAME  PROXY_NAME  AUDIT_OPTION          SUCCESS    FAILURE
---------  ----------  --------------------  ---------  ---------
  :                          :                  :          :
                        DROP ANY TABLE        BY SESSION BY SESSION
                        CREATE EXTERNAL JOB   BY SESSION BY SESSION
TT                      CREATE EXTERNAL JOB   BY SESSION BY SESSION
  :                          :                  :          :
```

```
NOAUDIT CREATE EXTERNAL JOB;

SELECT * FROM dba_stmt_audit_opts;

USER_NAME  PROXY_NAME  AUDIT_OPTION          SUCCESS    FAILURE
---------  ----------  --------------------  ---------  ---------
  :                          :                  :          :
                        DROP ANY TABLE        BY SESSION BY SESSION
TT                      CREATE EXTERNAL JOB   BY SESSION BY SESSION
  :                          :                  :          :
```

Аудит на основі привілеїв

- Приклади аудиту за видами привілеїв
  - Аудит будь-якої успішної чи неуспішної дії, яка залежить від привілею `DELETE ANY TABLE`

```
AUDIT DELETE ANY TABLE;
```

  - Перевіряє кожне невдале використання привілею `UPDATE ANY TABLE`

```
AUDIT UPDATE ANY TABLE BY ACCESS WHENEVER NOT SUCCESSFUL;
```

- AUDIT `SYSTEM` потрібний будь-якому користувачеві, який налаштовує систему або аудит на основі привілеїв
  - Зазвичай це адміністратор безпеки і ніхто інший

Об'єктно-орієнтований аудит$_1$

- Власники та адміністратори об’єктів можуть налаштувати аудит на основі об’єктів
  - `AUDIT ANY` дозволяє встановити аудит будь-якого об’єкта

- Приклади аудиту по конкретних об'єктах
  - Аудит успішних спроб запиту таблиці `emp` на основі сеансу

```
AUDIT SELECT ON emp WHENEVER SUCCESSFUL;
```

  - Аудит всіх невдалих спроб запиту таблиці `scott` за доступом

```
AUDIT SELECT, INSERT, DELETE ON scott.dept
BY ACCESS WHENEVER NOT SUCCESSFUL;
```

Об'єктно-орієнтований аудит$_2$

Object Option	Table	View	Seq	Proc Func Pkg	Mview	Dir	Lib	Object Type	Context
ALTER	X	--	X	--	X	--	--	X	--
AUDIT	X	X	X	X	X	X	--	X	X
COMMENT	X	X	--	--	X	--	--	--	--
DELETE	X	X	--	--	X	--	--	--	--
EXECUTE	--	--	--	X	--	--	X	--	--
FLASHBACK	X	X	--	--	--	--	--	--	--
GRANT	X	X	X	X	--	X	X	X	X
INDEX	X	--	--	--	X	--	--	--	--
INSERT	X	X	--	--	X	--	--	--	--
LOCK	X	X	--	--	X	--	--	--	--
READ	--	--	--	--	--	X	--	--	--
WRI	????								
RENAME	X	X	--	--	--	--	--	--	--
REF	Застарілий, не використовується								
SELECT	X	X	X	--	X	--	--	--	--
UPDATE	X	X	--	--	X	--	--	--	--
CRE	????								

Аудит на рівні об'єкта

- **Не можна вказати для кожного користувача**

```
AUDIT DELETE,UPDATE ON SCOTT.EMP WHENEVER NOT SUCCESSFUL;
AUDIT SELECT ON SCOTT.EMP BY ACCESS;
AUDIT INSERT ON SCOTT.EMP BY ACCESS WHENEVER SUCCESSFUL;
```

```
SELECT * FROM dba_obj_audit_opts;

                 OBJECT_
OWNER OBJECT_NAME TYPE    ALT AUD COM DEL GRA IND INS LOC REN SEL UPD REF EXE CRE REA WRI FBK
----- ----------- ------- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
SCOTT EMP         TABLE   -/- -/- -/- -/S -/- -/- A/- -/- -/- A/A -/S -/- -/- -/- -/- -/- -/-
```

`AUDIT ALL ON dept;`

- Аудит всіх можливих варіантів на об'єкті

`NOAUDIT ALL ON dept;`

- Видаляє ***КОЖЕН*** параметр аудиту на об’єкті

BY SESSION vs BY ACCESS

- Спочатку `BY SESSION` створив один запис аудиту для тверджень, які викликають ту саму діяльність аудиту
  - `BY ACCESS` створює запис аудиту кожного разу, коли запускається твердження аудиту, який можна перевірити
- В Oracle 11g `BY SESSION` викликає аудит стільки ж разів, скільки `BY ACCESS`, але записує менше інформації
  - Все ще залишається за замовчуванням
- Oracle рекомендує використовувати `BY ACCESS`
  - Схожі накладні витрати, але більше інформації

Аудит об'єкта моніторингу

❑Стан об'єктів аудиту

```
SELECT upd "Update Option"
FROM dba_obj_audit_opts
WHERE object_name IN ('DEPT','EMP');

OBJECT_NAME Update Option
----------- -------------
DEPT        A/-
EMP         S/A
```

S = за сеансом
A = за доступом
- = відсутність аудиту

Перший символ показує, чи аудит увімкнено для успішних спроб

Символ після ' / ' показує, чи аудит увімкнено для невдалих спроб

➢ Аудит об'єкта в журналі аудиту

```
SELECT ses_actions,returncode FROM dba_audit_object;

SES_ACTIONS     RETURNCODE
--------------- ----------
---------F------      913
---F-----------      913
---------S------        0
---S-----------        0
```

delete select

```
DELETE FROM emp WHERE empno =
  (SELECT * FROM emp
   WHERE ename = 'x');
ORA-00913: too many values

DELETE FROM emp WHERE empno =
  (SELECT empno FROM emp
   WHERE ename = 'x');
```

Аудит системи моніторингу

❑Показати статус перевірки входів користувачів за допомогою `dba_stmt_audit_opts`

```
SELECT * FROM dba_stmt_audit_opts;

USER_NAME  PROXY_NAME  AUDIT_OPTION          SUCCESS    FAILURE
---------- ---------- --------------------- ---------- ----------
                      CREATE SESSION BY ACCESS BY ACCESS
```

➢ **Показати статус перевірки системних привілеїв за допомогою `dba_priv_audit_opts`**

```
SELECT * FROM dba_priv_audit_opts;

USER_NAME  PROXY_NAME  PRIVILEGE             SUCCESS    FAILURE
---------- ---------- --------------------- ---------- ----------
                      SELECT ANY TABLE BY ACCESS BY ACCESS
```

➢ **Перегляд `dba_common_audit_trail` показує всю інформацію аудиту**

- Включає детальний аудит

Аудит аудиторського сліду

- Якщо звичайні користувачі мають доступ до `sys.aud$` , цей доступ потрібно перевірити

```
AUDIT SELECT, INSERT, UPDATE, DELETE ON sys.aud$ BY ACCESS;
```

  - Усі дії, які виконуються користувачами , `які не є SYSDBA`, тепер перевірятимуться

- Простий `SELECT на sys.aud$` створить запис аудиту в `aud$`
  - DELETE цього запису аудиту буде успішним, але буде створено інший запис операції `видалення`
- Будь-які записи DML, виконані на `aud$`, не можуть бути видалені звичайними користувачами
- Налаштування цього типу аудиту діє як функція безпеки, потенційно виявляючи незвичайні або неавторизовані дії

Представлення з питань аудиту

STMT_AUDIT_OPTION_MAP	Містить інформацію про коди типів параметрів аудиту. Створено сценарієм SQL.BSQ під час CREATE DATABASE .
AUDIT_ACTIONS	Містить опис кодів типів дій журналу аудиту
ALL_DEF_AUDIT_OPTS	Містить параметри перевірки об'єкта за замовчуванням, які застосовуються під час створення об'єкта
DBA_STMT_AUDIT_OPTS	Описує поточні параметри аудиту системи в системі та за користувачами
DBA_PRIV_AUDIT_OPTS	Описує поточні системні привілеї, які перевіряються системою та користувачем
DBA_OBJ_AUDIT_OPTS USER_OBJ_AUDIT_OPTS	Описує параметри аудиту для всіх об'єктів Представлення USER показує параметри аудиту для всіх об'єктів, якими володіє поточний користувач
DBA_AUDIT_TRAIL USER_AUDIT_TRAIL	Перелічує всі записи журналу аудиту Представлення USER показує записи контрольного журналу, що стосуються поточного користувача.
DBA_AUDIT_OBJECT USER_AUDIT_OBJECT	Містить записи журналу аудиту для всіх об'єктів у системі Представлення USER містить записи журналу аудиту для заяв щодо об'єктів, які доступні для поточного користувача
DBA_AUDIT_SESSION USER_AUDIT_SESSION	Перераховує всі записи журналу аудиту щодо CONNECT і DISCONNECT У представленні USER перераховані всі записи контрольного журналу, що стосуються підключень і відключень для поточного користувача
DBA_AUDIT_STATEMENT , USER_AUDIT_STATEMENT	Перераховує записи журналу аудиту щодо GRANT, REVOKE, AUDIT, NOAUDIT і оператори ALTER SYSTEM по всій базі даних або для перегляду USER , виданого користувачем.
DBA_AUDIT_EXISTS DBA_AUDIT_POLICIES	Перераховує записи журналу аудиту, створені AUDIT NOT EXISTS Показує всі політики аудиту в системі.
DBA_FGA_AUDIT_TRAIL	Перераховує записи журналу аудиту для аудиту на основі вартості
DBA_COMMON_AUDIT_TRAIL	Поєднує стандартні та детальні записи журналу аудиту та включає системні і обов'язкові аудиторські записи, написані у форматі XML

Налаштування інформації про аудит

❑Тригер встановлює ідентифікатор клієнта для кожного сеансу під час входу

```
CREATE OR REPLACE TRIGGER trg_set_client_info
AFTER LOGON ON DATABASE
   DECLARE
   v_module v$session.module%TYPE;
BEGIN
   SELECT module INTO v_module
   FROM v$process p, v$session s
   WHERE p.addr = s.paddr
   AND s.audsid = USERENV('sessionid');
   dbms_session.set_identifier(sys_context
                  ('userenv','ip_address')
                  ||' - '||v_module);
END;
/
```

- Можна встановити багато інших критеріїв, наприклад метод автентифікації

Аудит тверджень

- У прикладі показано три оператори SELECT, які генеруватимуть записи аудиту
  - Твердження, введені `system`, створять ***три*** записи аудиту

```
CONN system/manager

SELECT ename,sal
FROM scott.emp
WHERE sal < (SELECT sal
             FROM scott.emp
             WHERE ename = 'WARD')
AND job = (SELECT job
           FROM scott.emp
           WHERE ename = 'WARD');

CONN scott/tiger

SELECT job FROM scott.emp;

SELECT empno FROM scott.emp;
```

Кореляція записів аудиту


```
CREATE OR REPLACE PROCEDURE format_aud
AS
BEGIN
   FOR r IN (SELECT db_user
                    ,client_id
                    ,object_schema
                    ,object_name
                    ,extended_timestamp
                    ,sql_text
                    ,statementid
             FROM dba_common_audit_trail
             GROUP BY db_user
                      ,statementid
                      ,sql_text
                      ,object_schema
                      ,object_name
                      ,client_id
                      ,extended_timestamp
             ORDER BY extended_timestamp ASC)
   LOOP
      dbms_output.put_line('Who: '||r.db_user);
      dbms_output.put_line('What: '||r.object_schema||'.'||r.object_name);
      dbms_output.put_line('Where: '||r.client_id);
      dbms_output.put_line('When: '
                           ||TO_CHAR(r.extended_timestamp,'MON-DD HH24:MI'));
      dbms_output.put_line('How:    '||r.sql_text);
      dbms_output.put_line('--------------End of audit record--------------');
   END LOOP;
END;
```

Групування `statementid` спричинить відображення `оператору system` у вигляді одного запису

Корельований результат аудиту

- Системний `запис` відображається лише один раз
- `SQLPLUS` відображається як частина ідентифікатора (встановлюється тригером)

```
Who: SYSTEM
What: SCOTT.EMP
Where: 127.0.0.1 - sqlplus.exe
When: NOV-2023 11:15
How:     SELECT ename,sal
FROM scott.emp
WHERE sal < (SELECT sal
FROM scott.emp
WHERE ename = 'WARD')
AND job = (SELECT job
FROM scott.emp
                WHERE ename = 'WARD')
--------------End of audit Record--------------
Who: SCOTT
What: SCOTT.EMP
Where: 127.0.0.1 - sqlplus.exe
When: NOV-2023 11:15
How:     SELECT job FROM scott.emp
--------------End of audit Record--------------
Who: SCOTT
What: SCOTT.EMP
Where: 127.0.0.1 - sqlplus.exe
When: NOV-2023 11:15
How:     SELECT ename FROM scott.emp;
```

Оператор `SELECT` є захоплений тому що `AUDIT_TRAIL = DB,EXTENDED`

Вплив аудиту на продуктивність$_1$

- З аудитом

```
CREATE OR REPLACE PROCEDURE perf_test_aud
AS
BEGIN
   FOR rec IN 1..50000
   LOOP
      FOR inner_rec IN (SELECT ename FROM scott.emp)
      LOOP
         NULL;
      END LOOP;
    END LOOP;
 END;
/

EXEC perf_test_aud

Elapsed : 9.93 seconds
```

Вплив аудиту на продуктивність$_2$

- Вимкніть аудит, а потім виконайте ще раз

```
NOAUDIT SELECT ON scott.emp;
```

```
CREATE OR REPLACE PROCEDURE perf_test_aud
AS
BEGIN
   FOR rec IN 1 ..50000
   LOOP
      FOR inner_rec IN (SELECT ename FROM scott.emp)
      LOOP
         NULL;
      END LOOP;
    END LOOP;
 END;
/

EXEC perf_test_aud

Elapsed : 2.26 seconds
```

Обмеження аудиту

- Неможливо показати, які дані насправді бачив користувач
  - Але фіксує SCN операції
  - Можна використовувати Flashback на основі SCN, щоб переглянути дані
    - Залежить від `UNDO_RETENTION`
  - Oracle Consultancy має Selective Audit як «повне» рішення
- Дані аудиту зберігаються після відкату
- Аудит створюється, навіть якщо немає змін рядків
- Неможливо перевірити певні стовпці чи умови
- Детальний аудит дає додаткові можливості

Донецький національний технічний університет

Тема 6. Моніторинг та аудит безпеки в БД Oracle.
Лекція 9. Техніка аудиту в Oracle (продовження)

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач : проф. Дорогий Я.Ю.

Лекційні питання

- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Мета лекції

- Розглянути процес точного аудиту
- Ознайомитися з управлінням аудиторським слідом
- Ознайомитися з рекомендаціями щодо аудиту

Методи аудиту для Oracle

- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит
- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Функції тонкого аудиту (FGA)

1. Перевірка логічної умови
   - Все, що вказано в SQL
   - Порівняння результатів виклику функції
2. Чутливість колонки
3. Оброблювач подій
4. Захоплення SQL
   - Умовний аудит допомагає зменшити непотрібні аудити
   - Бізнес-правило
     - Співробітники повинні бачити лише свої записи
       - Аудит таблиці генерує інформацію аудиту для всіх `SELECT`
       - Просіювання через аудит є громіздким

Налаштування FGA

- FGA не залежить від параметрів ініціалізації
- Впровадити аудит для користувачів, які мають доступ до записів інших співробітників

```
BEGIN
   dbms_fga.add_policy(object_schema => 'SCOTT'
                      ,object_name => 'EMPTAB_FGA'
                      ,policy_name => 'fga_emp'
                      ,audit_condition => 'ENAME != USER'
                      ,audit_trail => dbms_fga.db +
                                      dbms_fga.extended);
END;
```

- **`audit_trail` розмістить аудит у `fga_log$` (з текстами sql) (за замовчуванням)**
  - Альтернативно можна вказати як XML
- **Якщо `audit_condition` залишити `NULL`, аудит відбувається *завжди***
- **Вимагає використання оптимізатора на основі витрат і генерації статистики**

Генерація записів FGA

- **Два оператори з використанням таблиці `emptab_fga`**

```
SELECT sal,ename FROM scott.emptab_fga WHERE ename = 'SCOTT';

       SAL ENAME
---------- ----------
      3000 SCOTT

SELECT sal,ename FROM scott.emptab_fga WHERE ename = 'SMITH';

       SAL ENAME
---------- ----------
      2850 SMITH
```

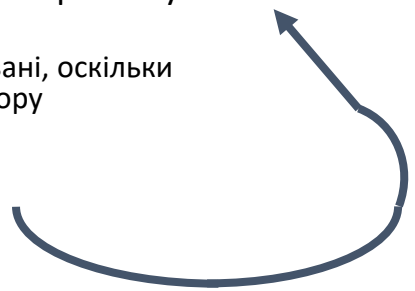
- У таблицю аудиту поміщено лише один запис (SMITH ***не є*** користувачем SCOTT)

```
Who: SCOTT
What: SCOTT.EMPTAB_FGA
Where: - sqlplusw.exe
When: Nov-23 12:52
How:    SELECT sal,ename FROM scott.emptab_fga WHERE ename = 'SMITH'
--------------End of audit record--------------
```

Точні функції аудиту

- Коли `AUDIT_TRAIL = DB` або `DB,EXTENDED`, записи FGA зберігаються в таблиці `fga_log$`, яка за замовчуванням належить `sys`
  - Показано в `dba_fga_audit_trail`
  - Також відображається в `dba_common_audit_trail` з іншою інформацією аудиту
    - Видалення `aud$` не призведе до видалення записів FGA

- Точний аудит виходить за рамки тригерів
  - Тригери викликають PL/SQL для кожного обробленого рядка
    - Створіть запис аудиту лише тоді, коли відповідний стовпець ***змінено за допомогою*** DML
  - FGA не викликається для кожного рядка – лише один раз для кожної політики
    - Аудит відбувається, коли вказаний стовпець змінюється ***або*** використовується ***в*** критеріях вибору
      - Виявляє користувачів, які сподіваються, що їхні дії будуть замасковані, оскільки вони використовують конфіденційні стовпці лише в критеріях відбору



Орієнтація на тонкий аудит

- Використовуйте функції для виконання комплексної перевірки
  - Надає політикам гнучкість

```
CREATE OR REPLACE FUNCTION outside_hours
RETURN BINARY_INTEGER
AS
   v_return_val NUMBER;
   v_day_of_week VARCHAR2(1) := TO_CHAR(sysdate,'DY');
   v_hour NUMBER(2):= TO_NUMBER(TO_CHAR(sysdate, 'HH24'));
BEGIN
   IF (v_day_of_week IN ('SAT', 'SUN')
      OR v_hour < 8
      OR v_hour > 17)
   THEN v_return_val := 1;
   ELSE v_return_val := 0;
   END IF;
   RETURN v_return_val;
END;
```

  - Перевіряє час поза робочим часом

Маніпулювання аудиторською політикою

- Аудит відбувається на `emptab_fga`, якщо функція `outside_hours` повертає «1»
  - В неробочий час

```
BEGIN
   dbms_fga.drop_policy(
       object_schema => 'SCOTT'
      ,object_name   => 'EMPTAB_FGA'
      ,policy_name   => 'fga_emp');
END;
/

BEGIN
   dbms_fga.add_policy(
       object_schema   => 'SCOTT'
      ,object_name     => 'EMPTAB_FGA'
      ,policy_name     => 'fga_emp'
      ,audit_condition => 'SECADMIN.OUTSIDE_HOURS = 1');
END;
/
```

Чутливість колонки

- Будь-який працівник має право доступу до записів інших працівників
  - Але не має доступу до чутливого стовпця `sal`
- Для цього правила потрібна чутливість стовпця
  - Аудит відбувається, лише якщо стовпець отримано ***та*** умова виконана
  - Якщо умова відсутня, аудит відбувається, коли стовпець отримується або відбуваються маніпуляції з ним

```
BEGIN
   dbms_fga.add_policy(object_schema   => 'SCOTT'
                      ,object_name     => 'EMPTAB_FGA'
                      ,policy_name     => 'fga_emp'
                      ,audit_condition => 'ENAME != USER'
                      ,audit_column    => 'SAL');
END;
/
```

Параметри для аудиту стовпців

- Використовуйте `audit_column_opts`
  - Релевантно лише за наявності списку стовпців у `audit_column`
  - Визначає, чи перевіряється твердження

Значення `audit_column_opts`	Аудиторська дія
`any_columns` (значення за замовчуванням)	твердження перевіряється, якщо в ньому є посилання на ***будь-які*** стовпці, вказані у параметрі `audit_column`
`all_columns`	вираз повинен посилатися на ***всі*** стовпці, перелічені в `audit_column`, які підлягають перевірці

Тест на чутливість до стовпців

- Які з наступних запитів будуть перевірені?

```
1.  SELECT AVG(sal) FROM emptab_fga;

2.  SELECT sal FROM emptab_fga WHERE ename = 'SCOTT';

3.  SELECT empno,job,sal FROM emptab_fga WHERE deptno = 10;

4.  SELECT empno,job FROM emptab_fga WHERE deptno = 10;

5.  SELECT sal FROM emptab_fga WHERE ename = 'ZULU';

6.  SELECT ename FROM emptab_fga WHERE ename = 'SMITH';

7.  SELECT ename FROM emptab_fga WHERE sal > 2975;
```

Проаудитовані запити

- Твердження 1, 3 і 7

```
Who: SCOTT
What: SCOTT.EMPTAB_FGA
Where: - sqlplusw.exe
When: Nov-11 16:18
How:    SELECT AVG(sal) FROM emptab_fga
--------------End of audit record--------------
Who: SCOTT
What: SCOTT.EMPTAB_FGA
Where: - sqlplusw.exe
When: Nov-11 16:18
How:    SELECT empno,job,sal FROM emptab_fga WHERE deptno = 10
--------------End of audit record--------------
Who: SCOTT
What: SCOTT.EMPTAB_FGA
Where: - sqlplusw.exe
When: Nov-11 16:18
How:    SELECT ename FROM emptab_fga WHERE sal > 2975
--------------End of audit record--------------
```

Надсилання аудиторської інформації

- Перед переглядом записів аудиту може бути затримка
- Надішліть сповіщення адміністраторам баз даних
  - Використовуйте оброблювач подій
  - Немає необхідності запитувати інформацію в журналі аудиту

```
BEGIN
   dbms_fga.add_policy(object_schema   => 'SCOTT',
                       ,object_name    => 'EMPTAB_FGA'
                       ,policy_name    => 'fga_emp'
                       ,audit_condition => 'ENAME != USER'
                       ,audit_column   => 'SAL'
                       ,handler_schema  => 'SECADMIN'
                       ,handler_module  => 'FGA_ALERT');
END;
/
```

Оброблювач подій


```
CREATE OR REPLACE PROCEDURE fga_alert (pi_sch VARCHAR2
                                      ,pi_tab VARCHAR2
                                      ,pi_pol VARCHAR2)
AS
msg VARCHAR2(20000);
BEGIN
  msg := 'Wake up Carl - the '||pi_tab||' table owned by '||
  pi_sch||' has been accessed with this statement :
  '||sys_context('userenv','current_sql')||'. The time is : '
   ||TO_CHAR(SYSDATE, 'Day DD MON, YYYY HH24:MI:SS');
  UTL_MAIL.SEND (
     sender      => 'oracle_database@wlv.ac.uk'
    ,recipients  => 'carl.dudley@wlv.ac.uk'
    ,subject     => 'Salaries in the HR.EMPLOYEES
                     table have been accessed'
    ,message     => msg);
END email_alert;
/
```

Резюме FGA

- FGA не бачитиме змін у запитах, спричинених RLS
- FGA не показує дані, отримані користувачем
- FGA не відбувається, якщо рядки не повертаються або не оновлюються
  - Можна поєднати це зі стандартним аудитом
- Неможливо обійти будь-які детальні заходи аудиту
  - Орієнтований на базу даних

Методи аудиту для Oracle

- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит
- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Управління аудитом – таблиця aud$

- Якщо AUDIT_TRAIL встановлено в DB або DB,EXTENDED
  - Інформація аудиту зберігається в sys.aud$ у ***системному*** табличному просторі
  - Максимальний розмір регулюється параметрами сховища для системного табличного простору

- Періодично видаляйте дані з aud$, щоб звільнити місце та полегшити пошук
  - aud$ (і fga_log$) є ***АБСОЛЮТНО ЄДИНИМИ словниками***, на яких можна виконувати прямий DML
    - ***Не*** вставляйте, не оновлюйте та не видаляйте записи в будь-якій іншій словниковій таблиці
  - Зазвичай лише sys може виконувати видалення таблиці aud$

- Будь-яке видалення в журналі аудиту перевіряється
  - Остерігайтеся, що sys може TRUNCATE таблицю aud$

Журнал аудиту – ручне керування

- Якщо журнал аудиту заповнений і `CREATE SESSION` перевіряється
  - Користувачі не можуть підключитися до бази даних
  - У цьому випадку підключіться як `SYS` і звільніть місце в контрольному журналі

```
DELETE FROM sys.aud$
WHERE ntimestamp# >
      TO_TIMESTAMP ('01-JAN-12 08.08.58.427000 PM')
AND ntimestamp# <
    TO_TIMESTAMP ('31-MAR-12 10.23.59.681000 PM');
```

```
DELETE FROM sys.aud$;
```

- **Згорніть або обріжте `aud$`, щоб відновити простір, якщо потрібно**

```
ALTER TABLE sys.aud$ SHRINK SPACE;
```

```
TRUNCATE TABLE sys.aud$;
```

Параметри керування журналом аудиту


```
SELECT * FROM dba_audit_mgmt_config_params;

PARAMETER_NAME              PARAMETER_VALUE AUDIT_TRAIL
--------------------------- --------------- --------------------
DB AUDIT TABLESPACE         SYSAUX          STANDARD AUDIT TRAIL
DB AUDIT TABLESPACE         SYSAUX          FGA AUDIT TRAIL
AUDIT FILE MAX SIZE         10000           OS AUDIT TRAIL
AUDIT FILE MAX SIZE         10000           XML AUDIT TRAIL
AUDIT FILE MAX AGE          5               OS AUDIT TRAIL
AUDIT FILE MAX AGE          5               XML AUDIT TRAIL
DB AUDIT CLEAN BATCH SIZE 10000             STANDARD AUDIT TRAIL
DB AUDIT CLEAN BATCH SIZE 10000             FGA AUDIT TRAIL
OS FILE CLEAN BATCH SIZE  1000              OS AUDIT TRAIL
OS FILE CLEAN BATCH SIZE  1000              XML AUDIT TRAIL
```

- Значення можна встановити за допомогою `dbms_audit_mgmt`
  - Частина SE та EE від Oracle11gR2
    - Раніше було доступно лише через Audit Vault

Журнал аудиту – автоматичне керування

- Для журналу аудиту бази даних окремі записи аудиту, створені до вказаної позначки часу, можна очистити за допомогою *dbms_audit_mgmt*
- Для журналу аудиту операційної системи ви очищаєте цілі файли аудиту, створені до мітки часу
  - Це може зайняти деякий час
- Попередні кроки:

1. Якщо необхідно, налаштуйте розміри журналів (онлайн, архів)
   - Видалення контрольного сліду може спричинити багато повторів і скасувань
   - Можна перенести контрольний слід до табличного простору `Sysaux`
     - Використовуйте `dbms_audit_mgmt.set_audit_trail_location`
2. Сплануйте часову позначку та стратегію архівування
   - Вирішіть, скільки аудиту залишити
3. Операції очищення журналу аудиту
   - Використовуйте `dbms_audit_mgmt.init_cleanup`

Автоматичне керування за допомогою dbms_audit_trail


```
BEGIN
   dbms_audit_mgmt.init_cleanup(
      audit_trail_type => dbms_audit_mgmt.audit_trail_fga_std
     ,default_cleanup_interval => 24);
END;
```

- Журнал аудиту FGA тепер буде в Sysaux

```
CREATE OR REPLACE PROCEDURE sys.delete_dbfga_records IS
BEGIN
   dbms_audit_mgmt.set_last_archive_timestamp(
      audit_trail_type => dbms_audit_mgmt.audit_trail_fga_std
     ,last_archive_time => SYSTIMESTAMP - 1);
   dbms_audit_mgmt.clean_audit_trail(
      audit_trail_type => dbms_audit_mgmt.audit_trail_fga_std
     ,use_last_arch_timestamp => TRUE);
END;
```

- **Очищає записи аудиту FGA раніше нововстановленої позначки часу**

Створення завдання з розкладом для очищення контрольного сліду

- Процедура тестування очищає контрольний слід кожні десять секунд

```
BEGIN
   dbms_scheduler.create_schedule (
      schedule_name   => 'DELETE_DBFGA_RECORDS_SCHED'
     ,repeat_interval =>'FREQ = SECONDLY; INTERVAL = 10;');
END;
```

```
BEGIN
   dbms_scheduler.create_job (
      job_name => 'DELETE_DBFGA_RECORDS_JOB'
     ,job_type => 'STORED_PROCEDURE'
     ,job_action => 'SYS.DELETE_DBFGA_RECORDS'
     ,enabled => TRUE
     ,auto_drop => false
     ,schedule_name => 'DELETE_DBFGA_RECORDS_SCHED');
END;
```

dbms_audit_mgmt.create_purge_job

- Автоматизує налаштування завдання для очищення (обгортка на dbms_scheduler)

```
BEGIN
   dbms_audit_mgmt.create_purge_job(
       audit_trail_type => dbms_audit_mgmt.audit_trail_fga_std
      ,audit_trail_purge_interval => 1
      ,audit_trail_purge_name => 'HOURLY_PURGE_FGA'
      ,use_last_arch_timestamp => TRUE);
END;
```

- Але неможливо просунути/змінити мітку часу
  - Все ще потрібно налаштувати процедуру та роботу/графік, щоб зробити просунення
- Мінімальний інтервал - одна година

Методи аудиту для Oracle

- Огляд аудиту
- Аудит застосунків
- Аудит на основі тригерів
- Аудит користувачів системи
- Стандартний аудит
- Точний аудит
- Управління аудиторським слідом
- Рекомендації з аудиту

Рекомендації з аудиту

- ❑Використовуйте аудит на рівні системи
- ❑За необхідності використовуйте FGA
- ❑Виконуйте аудит доступу та зміни критичних даних
- ❑Проаналізуйте контрольний слід і журнали
  - Тут можуть допомогти інструменти
- ❑Створюйте звіти
- ❑Створюйте процедури / політики
- ❑Переглядайте зміст звітів
- ❑Налаштуйте сповіщення
- ❑Дійте за змістом

Донецький національний технічний університет

Тема 7. Резервне копіювання та відновлення БД Oracle. Лекція 10. Резервне копіювання та відновлення у Oracle

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Мета лекції
- Найкращі практики для планування резервного копіювання та відновлення
- Типові носії та типи даних, які є частиною стратегії резервного копіювання та відновлення
- Поширені топології резервного копіювання та відновлення

Мета лекції

- Ознайомитися з найкращими практиками для планування резервного копіювання та відновлення
- Ознайомитися з типовими носіями та типами даних, які є частиною стратегії резервного копіювання та відновлення
- Розглянути поширені топології резервного копіювання та відновлення

Резервна копія

- Резервна копія — це додаткова копія даних, яку можна використовувати для відновлення (Restore and Recovery)
- Резервна копія використовується, коли основна копія втрачена або пошкоджена
- Цю резервну копію можна створити як:
  - Проста копія (може бути одна або декілька копій)
  - Дзеркальна копія (копія завжди оновлюється тим, що записане в основну копію)

Стратегії резервного копіювання та відновлення

Доступні кілька способів перенести дані на резервний носій:

- ❑ Просте копіювання даних
- ❑ Створення дзеркала (або знімка), а потім просте копіювання даних
- ❑ Віддалене резервне копіювання
- ❑ Просте копіювання даних з подальшою дуплікацією або дистанційним копіюванням

Відновлення

- Підприємства створюють резервні копії своїх даних, щоб забезпечити їх відновлення у разі потенційної втрати
- Компанії також створюють резервні копії своїх даних відповідно до нормативних вимог
- Типи похідних резервних копій:
  - Аварійне відновлення
  - Архів
  - Оперативна копія

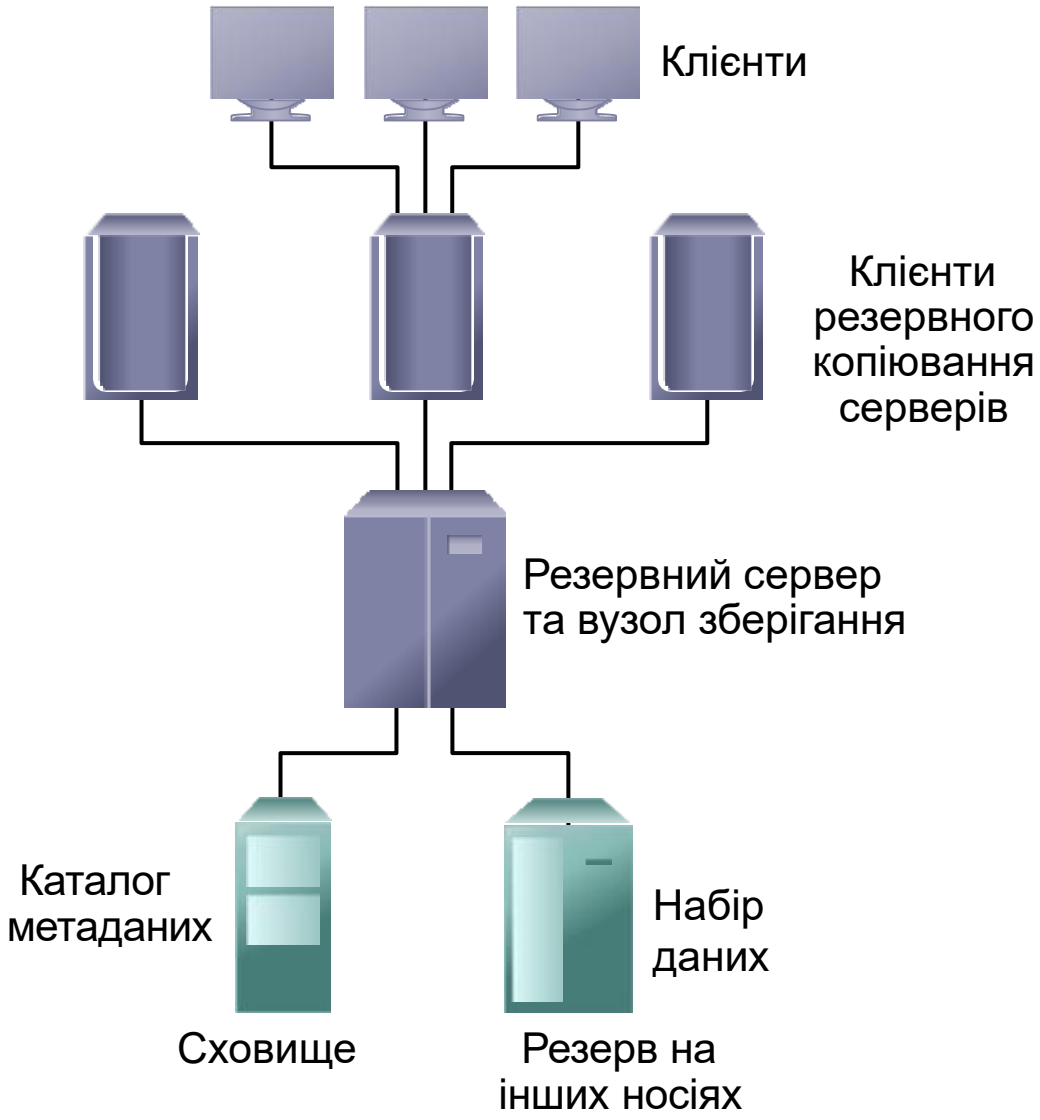
Причини для плану резервного копіювання

- Апаратні збої
- Людський фактор
- Помилки програми
- Порушення безпеки
- Катастрофи
- Нормативні та комерційні вимоги

Як працює резервне копіювання?

- Взаємодія «клієнт/сервер»
  - Сервер
    - Керує операцією
    - Веде резервний каталог
  - Клієнт
    - Збирає дані для резервного копіювання (клієнт резервного копіювання надсилає резервні дані на резервний сервер або вузол зберігання)
- Вузол зберігання

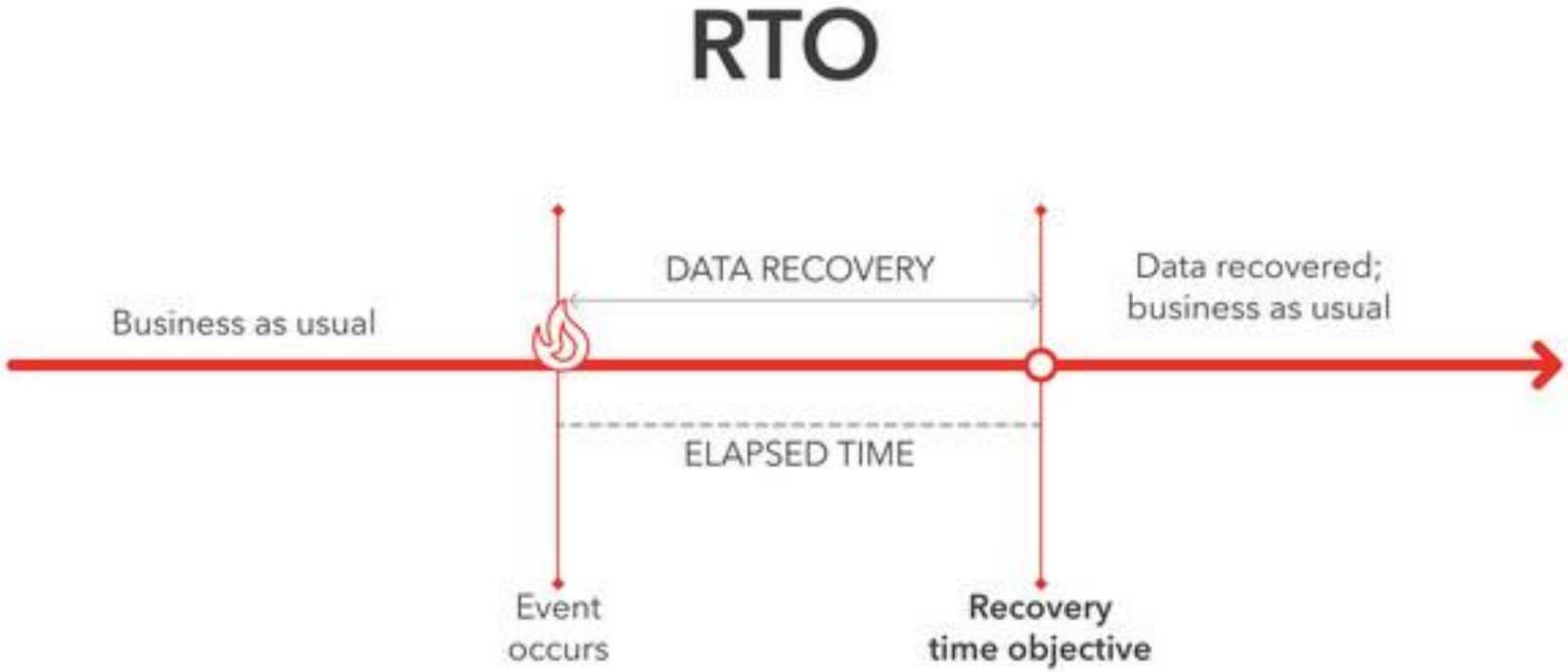
Загальна схема резервного копіювання



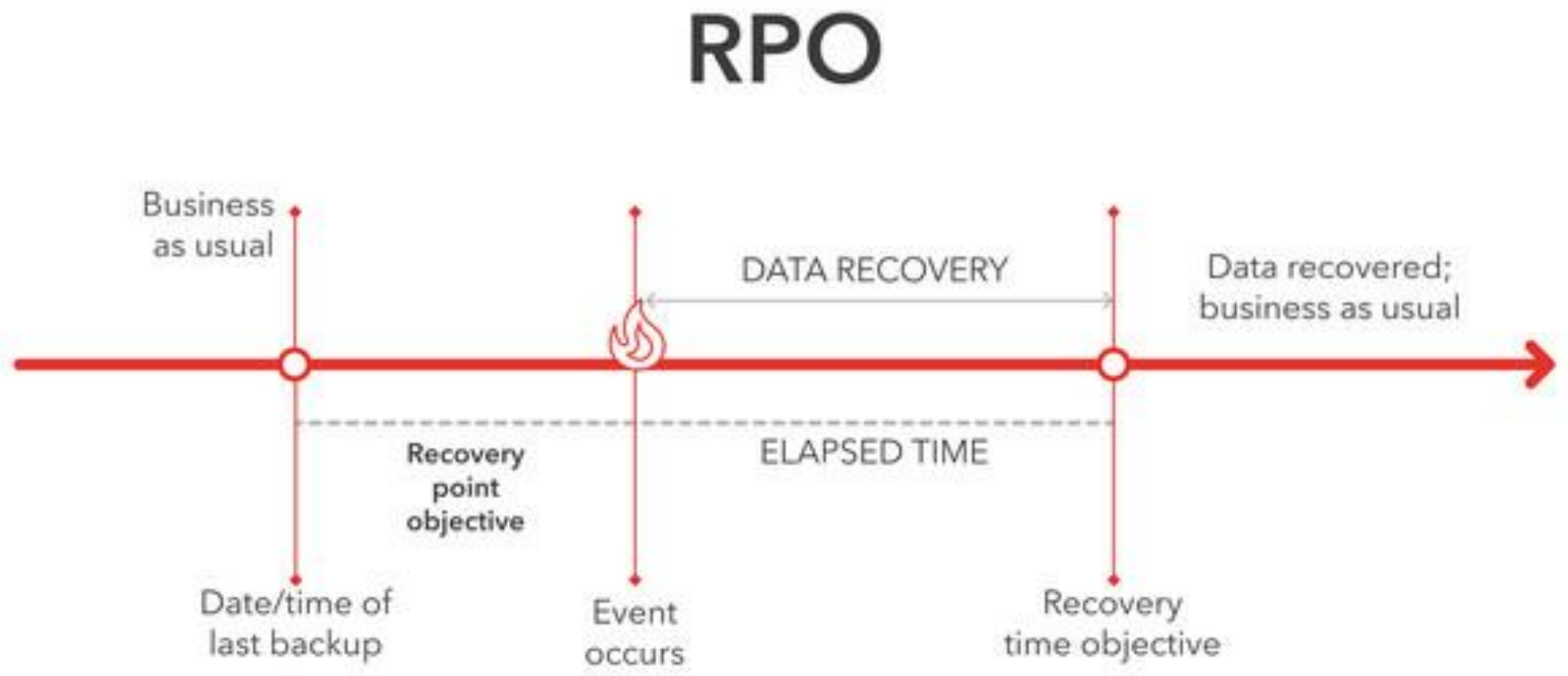
Ділові міркування

- Бізнес-потреби клієнтів визначають:
- Які вимоги до відновлення – RPO та RTO?
- Де і коли відбуватимуться відновлення?
- Які запити на відновлення найчастіше надходять?
- Для яких даних необхідно створити резервну копію?
- Як часто слід створювати резервні копії даних?
  - погодинно,
  - щодня,
  - щотижня,
  - щомісяця
- Скільки часу знадобиться для резервного копіювання?
- Скільки копій створити?
- Як довго зберігати резервні копії?

RTO


RTO
DATA RECOVERY
Business as usual
Data recovered;
business as usual
ELAPSED TIME
Event
occurs
Recovery
time objective

RPO


RPO
Business as usual
DATA RECOVERY
Data recovered; business as usual
Recovery point objective
ELAPSED TIME
Date/time of last backup
Event occurs
Recovery time objective

Основні відмінності між RTO та RPO

- Обидва показники вимагають комплексного планування та проактивного мислення щодо безпеки. Проте є кілька істотних відмінностей.
- Разом вони дозволяють дізнатися, на який термін ви можете дозволити собі переривання процесів та наскільки актуальними будуть дані.
- Важливо: чим коротше RTO або RPO, тим більша вартість процесу, і навпаки.

RTO	RPO
Зосереджується на доступності сервісів та даних	Фокусується на частоті бекапування та допустимих втратах інформації
Враховує всі аспекти IT-інфраструктури та резервного копіювання	Оцінює критичність даних та вартість бекапування
Покладається на best practice у побудові відмовостійкої інфраструктури та моніторингу	Покладається на автоматизацію завдяки програмному забезпеченню
Складний процес, оскільки включає більше рухомих частин та змінних (гарячі та холодні майданчики, місце відновлення, швидкість реагування на інцидент тощо)	Легше розрахувати, оскільки цей показник охоплює лише один аспект – дані

Як розрахувати та досягти бажаного RTO?

- Визначити критичні програми та системи
- Встановити реалістичні очікування
  - Сильна IT-команда
  - Інвестиції у спеціальне ПЗ
  - Поліпшення роботи критично важливих сервісів
  - Оповіщення в реальному часі
  - Обмеження RTO для кількох програм

Як розрахувати та реалізувати необхідний RPO?

- Рекомендації
  - Тести
  - Розподіл програм
  - Розрахунок бюджету
- 4 способи реалізації RPO
  - Бути реалістами
  - Навчити персонал
  - Модернізувати технології
  - Оптимізувати мережі

Зауваження щодо даних: характеристики файлу

- ***Місцезнаходження***
- ***Розмір***
- ***Кількість***

Зауваження щодо даних: стиснення даних

Здатність до стиснення залежить від типу даних:

- Двійкові файли програми – погано стискаються.
- Текст – добре стискається.
- Файли JPEG/ZIP – уже стиснені та розширюються, якщо їх стиснути знову.

Зауваження щодо даних: періоди зберігання

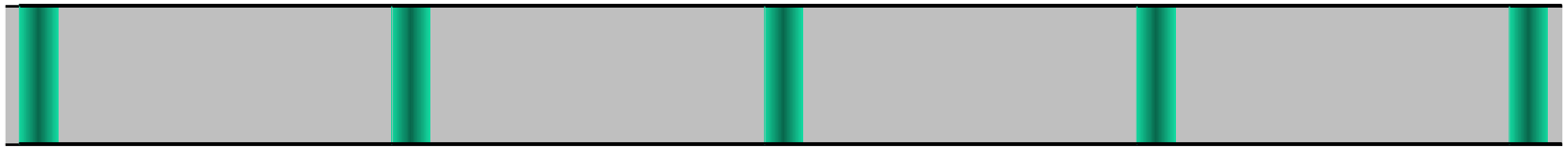
- ***Оперативна копія***
  - Набори даних на первинному носії (диску) до моменту виконання більшості запитів на відновлення, а потім переміщені до вторинного сховища (стрічка).
- ***Копія аварійного відновлення***
  - Відповідно до політики аварійного відновлення організації
    - Портативні носії (стрічки), надіслані в інше розташування/сховище
    - Реплікація в зовнішнє розташування (диск)
    - Резервне копіювання безпосередньо в зовнішнє розташування (диск, стрічка або емульована стрічка)
- ***Архів***
  - Відповідно до політики організації
  - Відповідно до нормативних вимог

Методи резервного копіювання бази даних

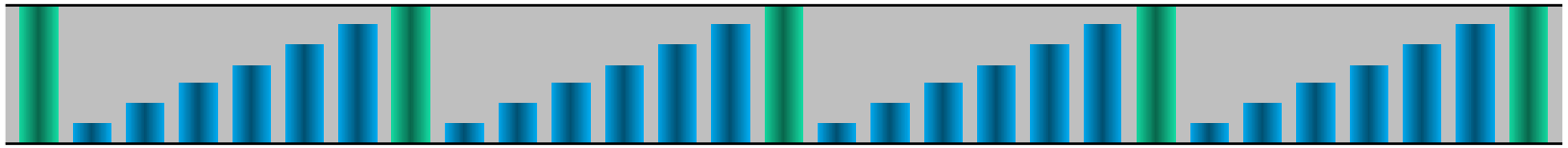
- Гаряче резервне копіювання: виробництво не переривається
- Холодне резервне копіювання: виробництво переривається
- Агенти резервного копіювання керують резервним копіюванням різних типів даних, таких як:
  - Структуровані (такі як бази даних)
  - Напівструктуровані (наприклад, електронна пошта)
  - Неструктуровані (файлові системи)

Деталізація та рівні резервного копіювання

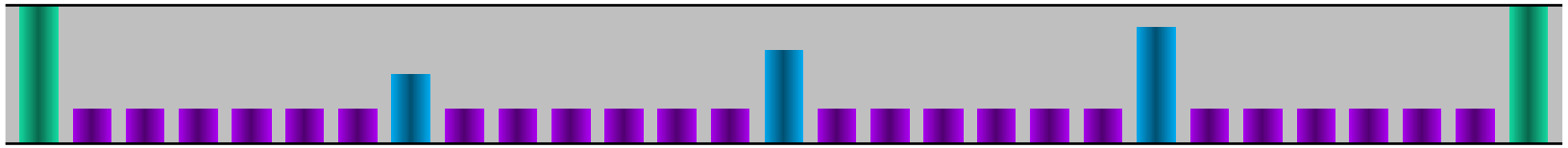
Повна копія

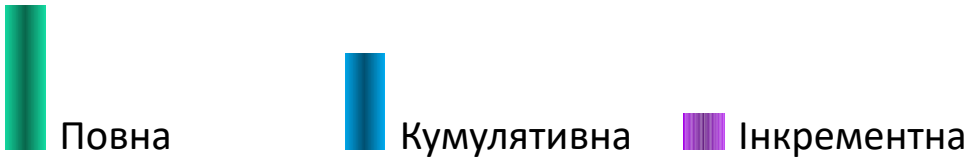


Кумулятивна (диференційна) копія

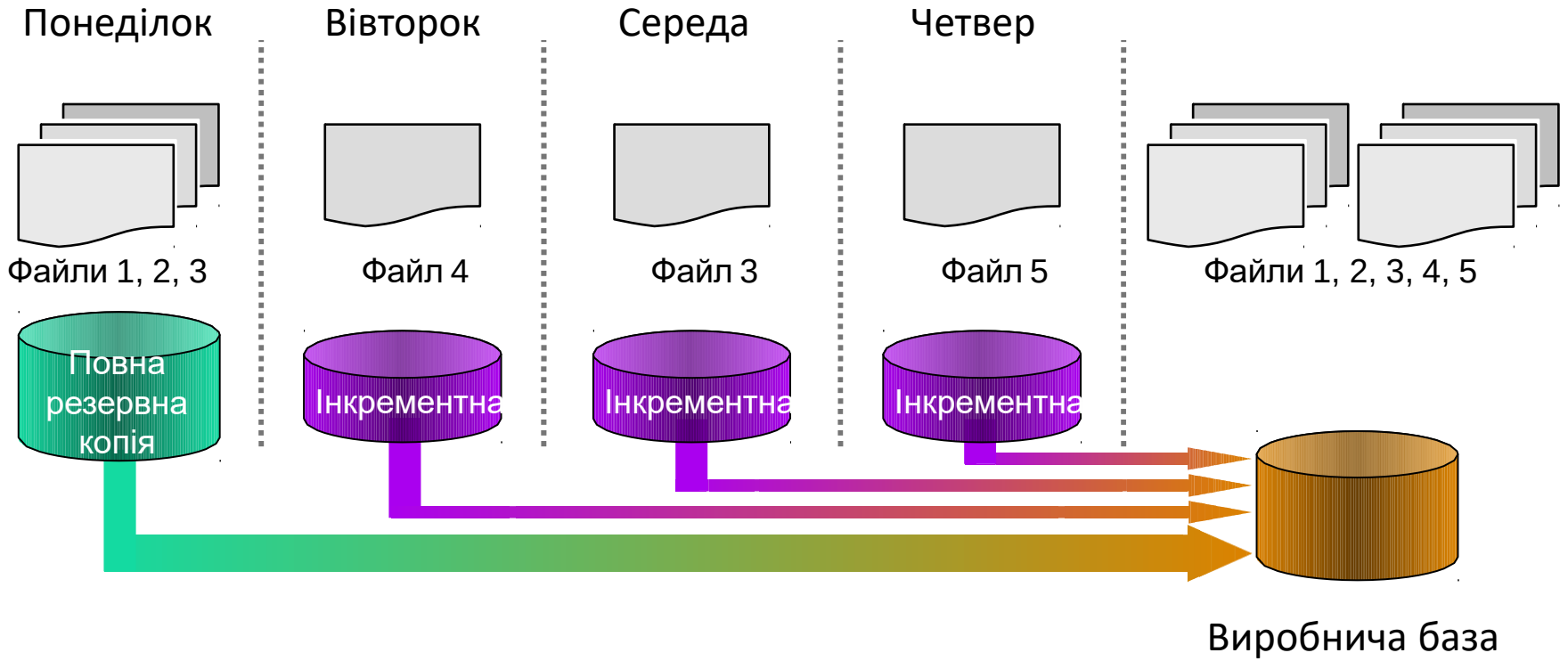


Інкрементна (накопичувальна) копія





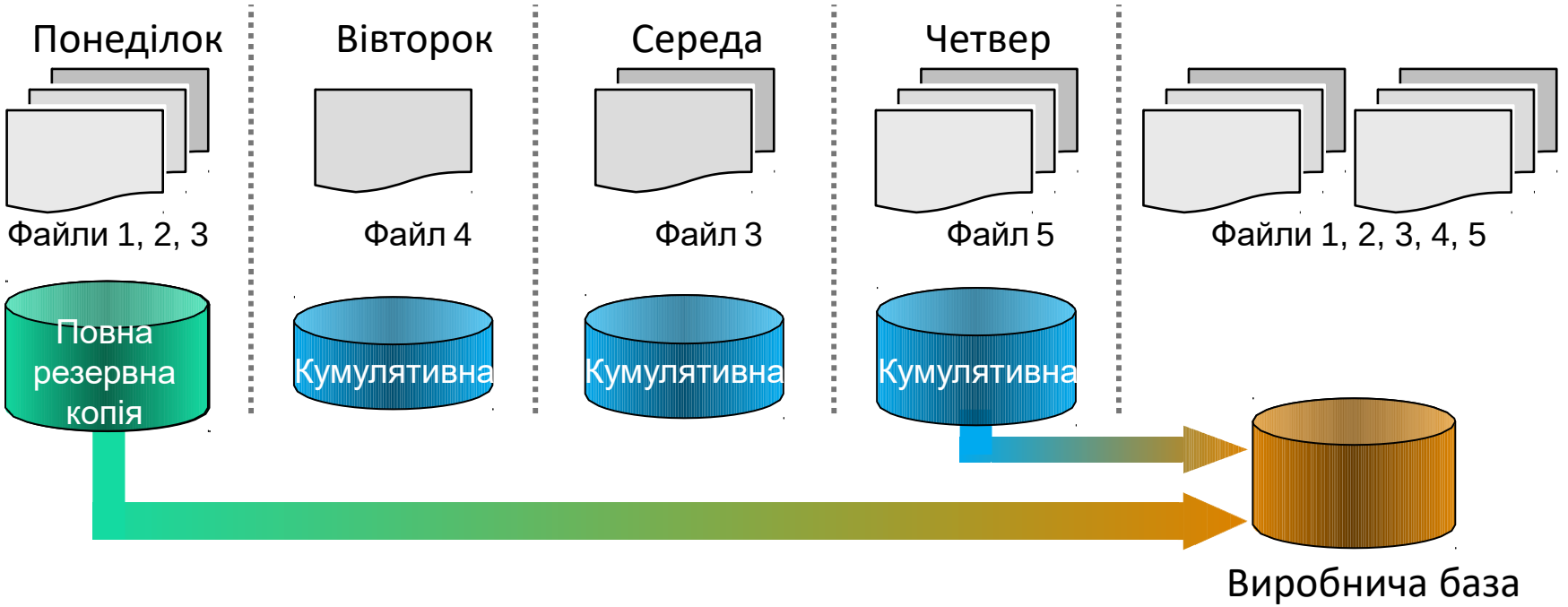
Відновлення інкрементної резервної копії



Ключові особливості

- Зберігаються резервні копії файлів, які змінилися після останнього повного або інкрементного резервного копіювання.
- Найменша кількість файлів для резервного копіювання, отже, швидше резервне копіювання та менше місця для зберігання.
- Довше відновлення, оскільки необхідно застосувати останню повну та всі наступні інкрементні резервні копії.

Відновлення накопичувальної резервної копії



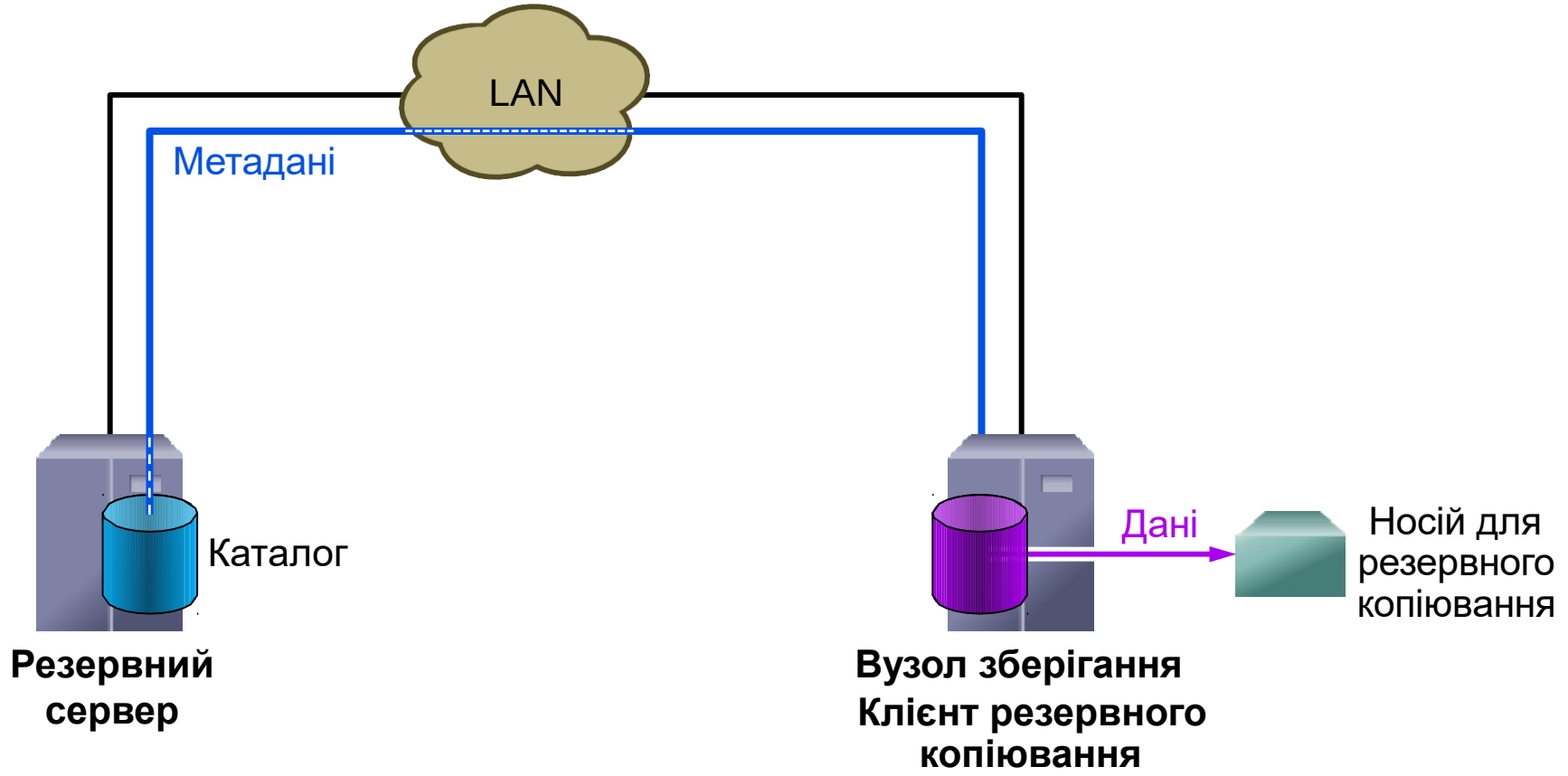
Ключові особливості

- Більше файлів для резервного копіювання, тому для резервного копіювання потрібно більше часу та більше місця для зберігання.
- Набагато швидше відновлення, оскільки потрібно застосовувати лише останню повну та останню накопичувальну резервну копії.

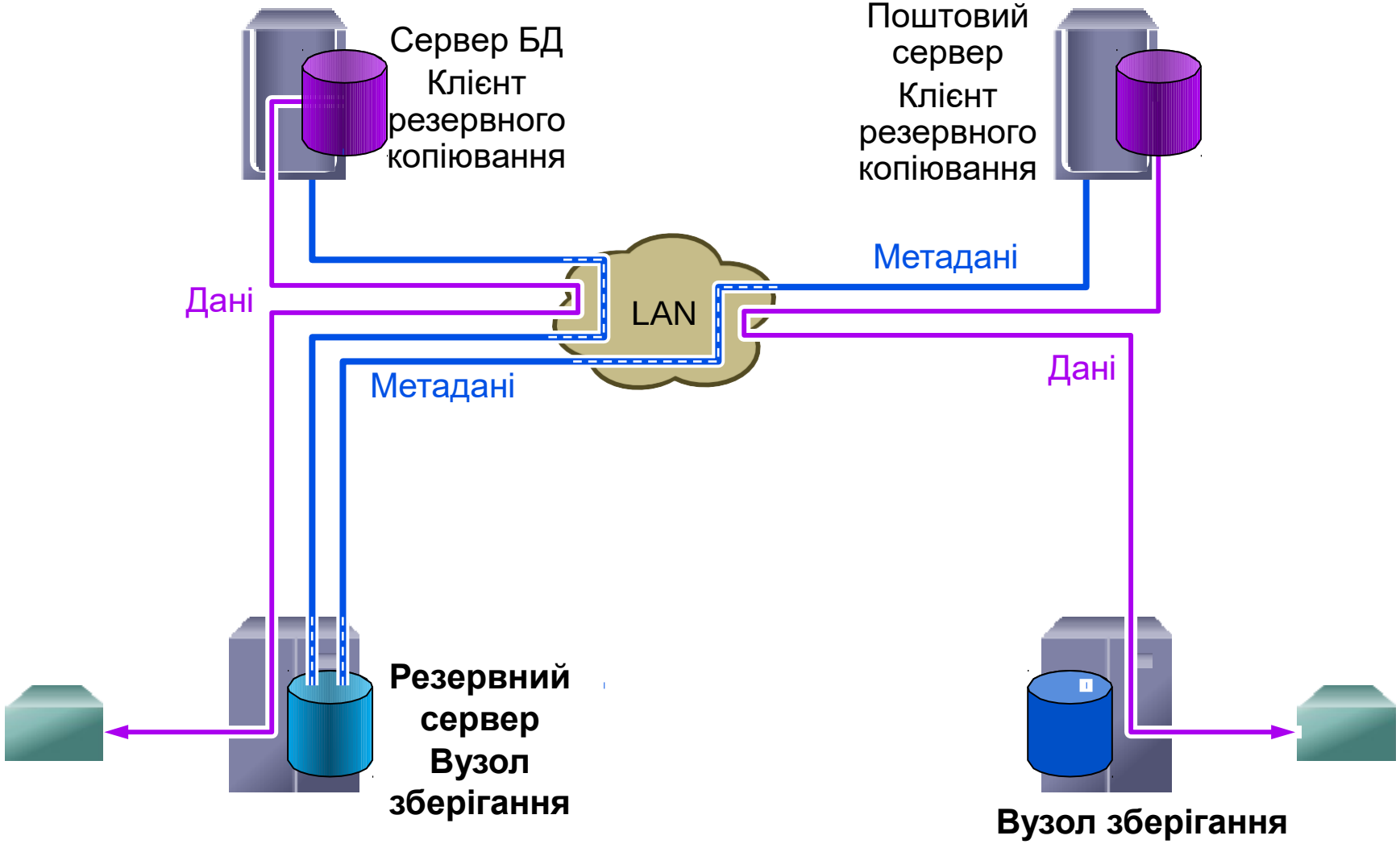
Топології архітектури резервного копіювання

- ***Існує 3 основні топології резервного копіювання:***
  - Резервне копіювання на основі прямого підключення
  - Резервне копіювання на основі локальної мережі
  - Резервне копіювання на основі SAN
- ***Ці топології можна інтегрувати, утворюючи «змішану» топологію***

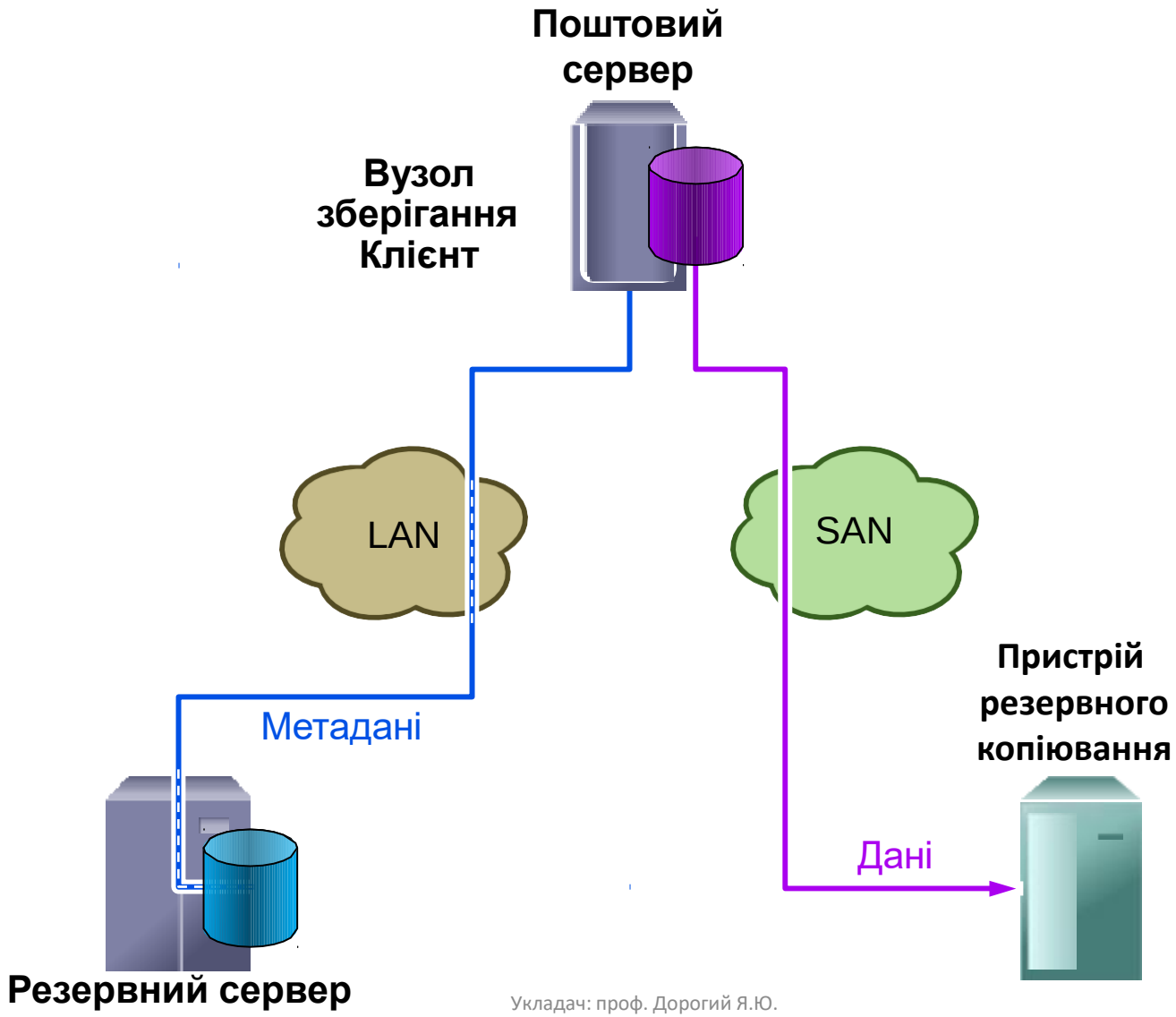
Резервне копіювання на основі прямого підключення



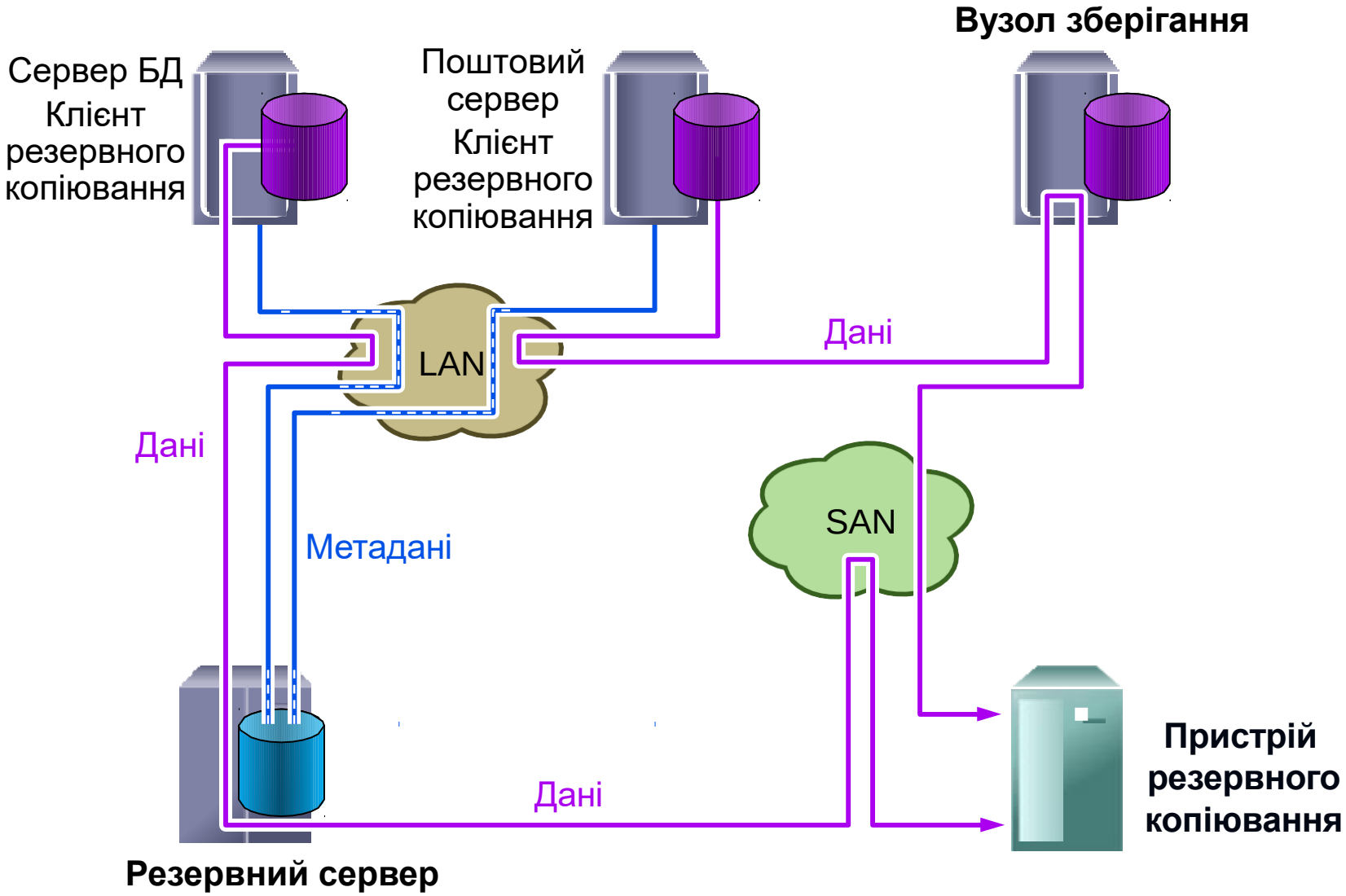
Резервне копіювання на основі локальної мережі



Резервне копіювання на базі SAN (LAN Free)



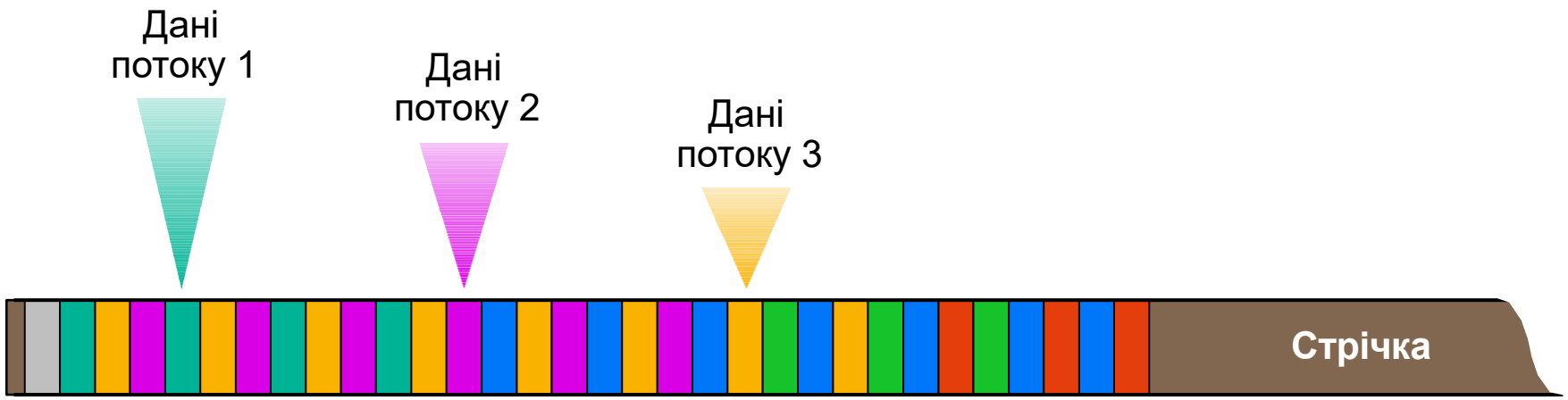
Змішані резервні копії SAN/LAN



Резервний носій

- Стрічка
  - Традиційне місце для резервного копіювання
  - Послідовний доступ
  - Без захисту
- Диск
  - Довільний доступ
  - Захищено масивом зберігання (RAID, гарячий резерв тощо)

Кілька потоків на стрічкових носіях



Кілька потоків чергуються для досягнення більшої пропускної здатності на стрічці

- Забезпечує потокову передачу стрічки для максимальної продуктивності запису
- Допомагає запобігти механічному пошкодженню стрічки
- Значно збільшує час відновлення

Резервне копіювання на диск

- Резервне копіювання на диск мінімізує використання стрічки в середовищах резервного копіювання, використовуючи диск як основний цільовий пристрій
  - Вигода в вартості
  - Зміни процесів не потрібні
  - Кращий рівень обслуговування
- Резервне копіювання на диск узгоджує стратегію резервного копіювання з RTO та RPO

Стрічка проти диска – порівняння відновлення

*Загальний час від моменту збою до повернення послуги користувачам електронної пошти

Типовий сценарій:

- 800 користувачів, поштова скринька 75 Мб
- База даних 60 Гб

Донецький національний технічний університет

Тема 7. Резервне копіювання та відновлення БД Oracle. Лекція 11. Процеси резервного копіювання та відновлення у Oracle

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

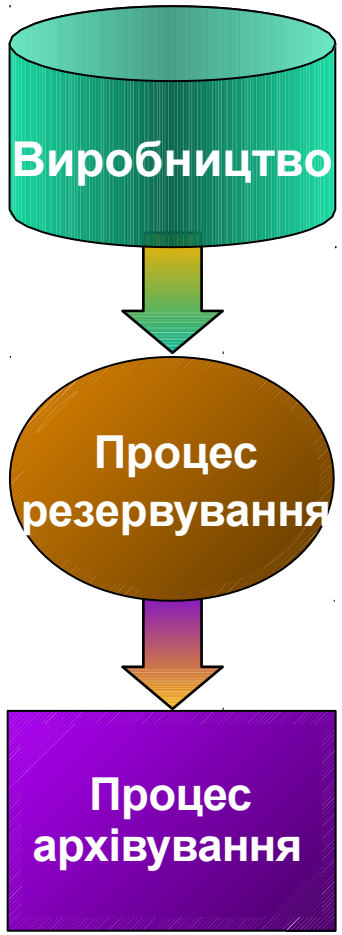
Лекційні питання

- Мета лекції
- Процес резервного копіювання та процес відновлення

Мета лекції

- Розглянути процес резервного копіювання та процес відновлення

Традиційний підхід до резервного копіювання, відновлення та архівування



Виробниче середовище зростає

- Потрібна постійна настройка та розміщення даних для підтримки продуктивності
- Потрібно додати більше сховищ Tier-1

Середовище резервного копіювання зростає

- Вікна резервного копіювання стають довшими, а завдання не виконуються
- Відновлення займає більше часу
- Потрібно більше стрічкових накопичувачів і силосів, щоб підтримувати рівень обслуговування

Архівне середовище зростає

- Вплив гнучкості для отримання вмісту за запитом
- Вимагає більше носіїв, додаючи витрати на керування
- Немає захисту інвестицій для вимог довгострокового утримання

Відмінності між резервним копіюванням/відновленням і архівуванням

Резервне копіювання/відновлення	***Архів***
Вторинна копія інформації, яка використовується для операцій **відновлення**	**Первинна копія** інформації, яка доступна для **пошуку** інформації
Покращує **доступність**, дозволяючи відновлювати програму до певного моменту часу	Додає **операційну ефективність** шляхом переміщення фіксованого/неструктурованого вмісту з робочого середовища
Як правило, **короткострокові** (тижні або місяці)	Як правило, **довгострокові** (місяці, роки або десятиліття)
Дані зазвичай **перезаписуються** періодично (наприклад, щомісяця)	Дані, як правило, **зберігаються для аналізу**, створення цінності або відповідності
Не для відповідності нормативним вимогам, хоча деякі змушені використовувати	Корисно для **відповідності** та має враховувати політику збереження інформації

Як працює типова програма резервного копіювання

- Клієнти резервного копіювання згруповані та пов’язані з розкладом резервного копіювання, який визначає час і тип резервного копіювання.
- Групи пов’язані з пулами, які визначають, які носії резервного копіювання використовуватимуться.
- Кожен резервний носій має унікальну мітку.
- Інформація про резервне копіювання записується в каталог резервних копій під час і після його завершення. У каталозі показано:
  - коли було виконано резервне копіювання та
  - який носій використовувався (мітка).
- Помилки та інша інформація також записуються в журнал.

Інтерфейси користувача програми резервного копіювання

Зазвичай існує два типи інтерфейсів користувача:

- ❑ Інтерфейс командного рядка – CLI
- ❑ Графічний інтерфейс користувача – GUI

Керування процесом резервного копіювання та відновлення

- Запуск програми B/R: резервне копіювання
  - Адміністратор резервного копіювання налаштовує його на автоматичний запуск у більшості випадків (якщо не весь час).
  - Більшість продуктів резервного копіювання пропонують клієнту резервного копіювання можливість ініціювати власне резервне копіювання (зазвичай вимкнено)
- Запуск програми B/R: відновлення
  - Зазвичай існує окремий графічний інтерфейс для керування процесом відновлення
  - Інформація витягується з каталогу резервних копій, коли користувач вибирає файли для відновлення
  - Після завершення вибору сервер резервного копіювання починає читати з необхідного носія резервного копіювання, а файли надсилаються до клієнта резервного копіювання.

Звіти про резервне копіювання

- Продукти резервного копіювання також пропонують функції звітування.
- Ці функції покладаються на резервний каталог і файли журналу.
- Звіти мають бути легкими для читання та надавати важливу інформацію, таку як:
  - Обсяг резервних копій даних
  - Кількість виконаних резервних копій
  - Кількість неповних резервних копій (та невдалих)
  - Типи помилок, які могли виникнути
- Залежно від використовуваного програмного продукту резервного копіювання можуть бути доступні додаткові звіти.

Важливість резервного каталогу

- Операції резервного копіювання сильно залежать від каталогу резервних копій
- Якщо каталог втрачено, програма резервного копіювання сама по собі не зможе визначити, де знайти конкретний файл, резервну копію якого створено два місяці тому, наприклад
- Його можна реконструювати, але зазвичай це означає, що всі носії резервного копіювання (тобто стрічки) мають бути прочитані
- Це хороша практика, щоб захистити каталог
  - Шляхом реплікації файлової системи, де він знаходиться, у віддалене розташування
  - Створивши резервну копію
- Деякі продукти резервного копіювання мають вбудовані механізми захисту свого каталогу (наприклад, автоматичне резервне копіювання)

Інструменти для захисту даних бази даних Oracle

Програмні засоби
Oracle Enterprise Manager,

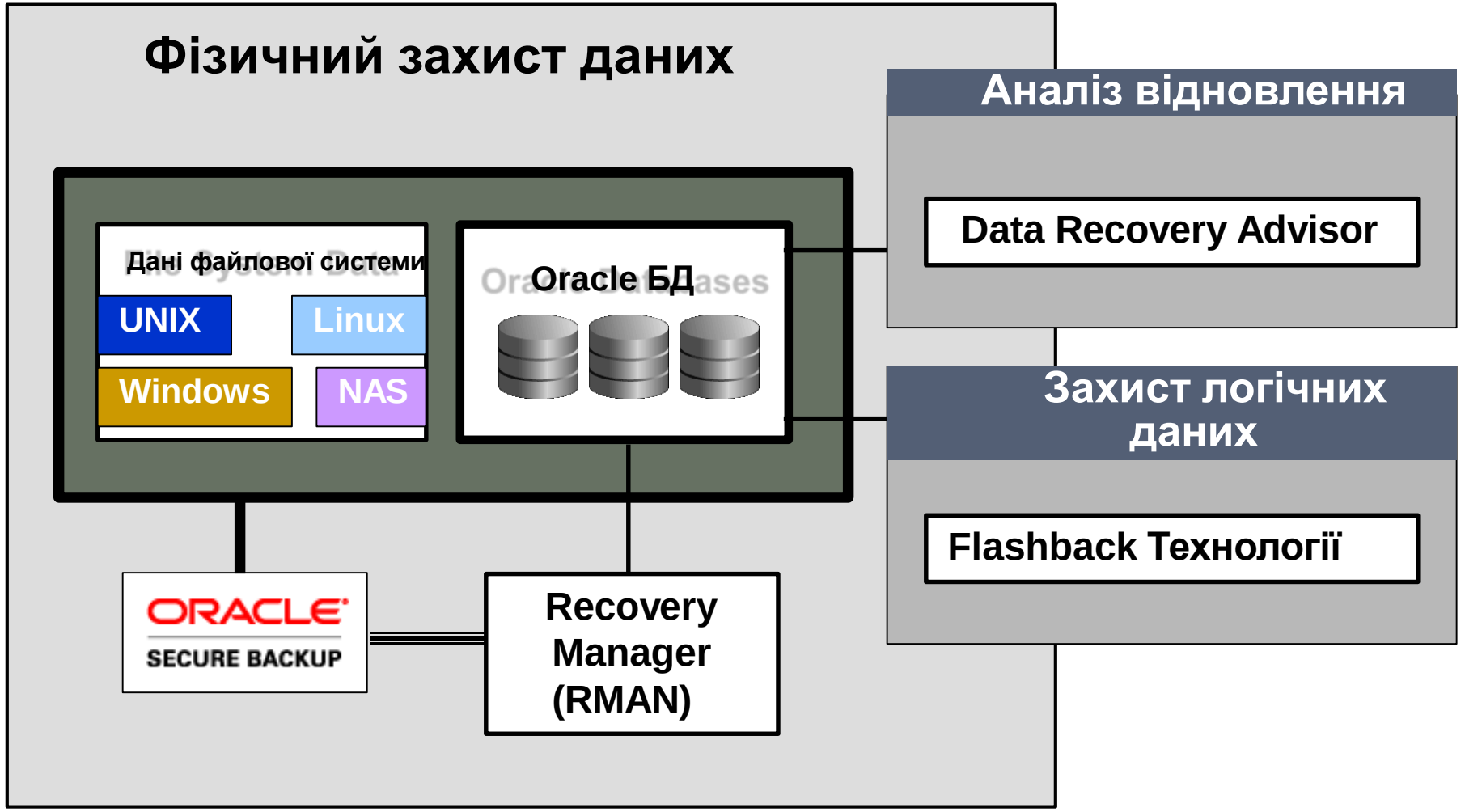
Засоби БД
Data Guard & Active Data Guard Data Recovery Advisor, RMAN

Засоби операційної с-ми
Built in Backup and Recovery Tools, Snapshots, Boot Environments

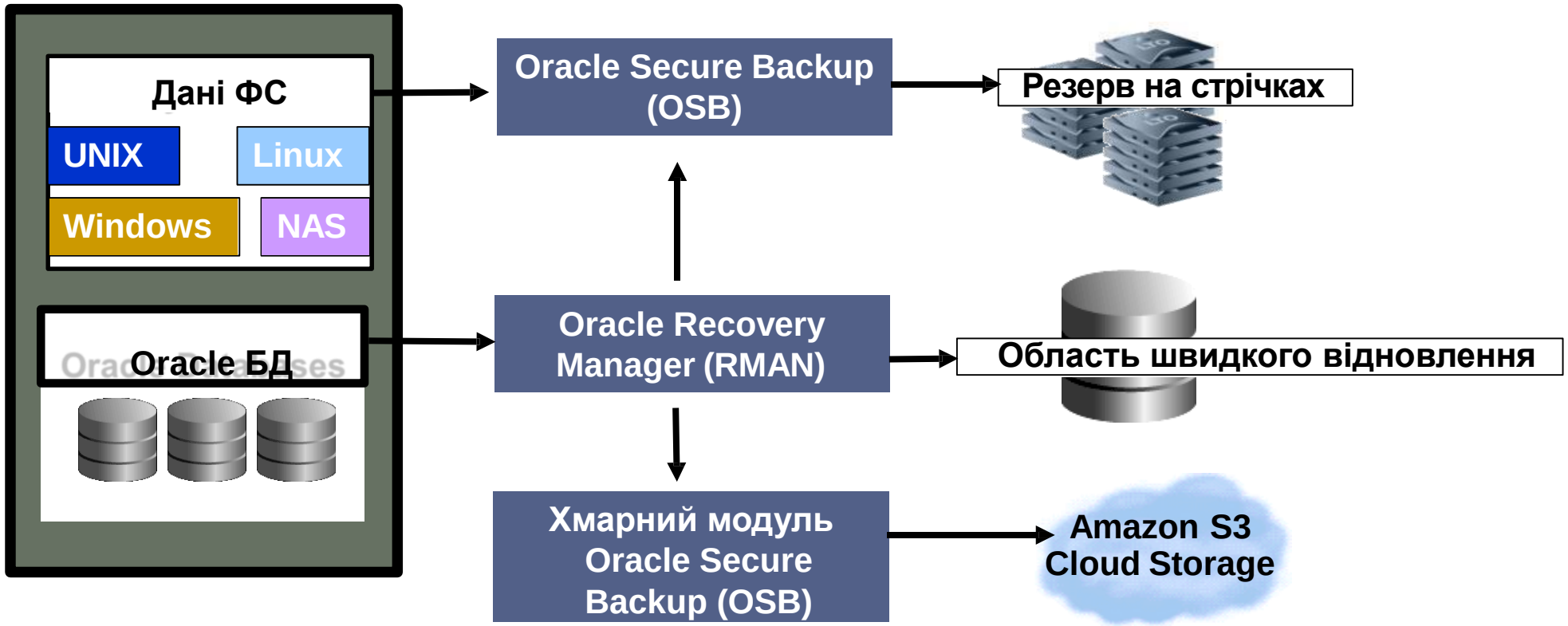
Засоби віртуалізації
Oracle Virtual Machine, Enterprise Manager

Засоби збереження
Snapshots, Replication, Archive, Analytics

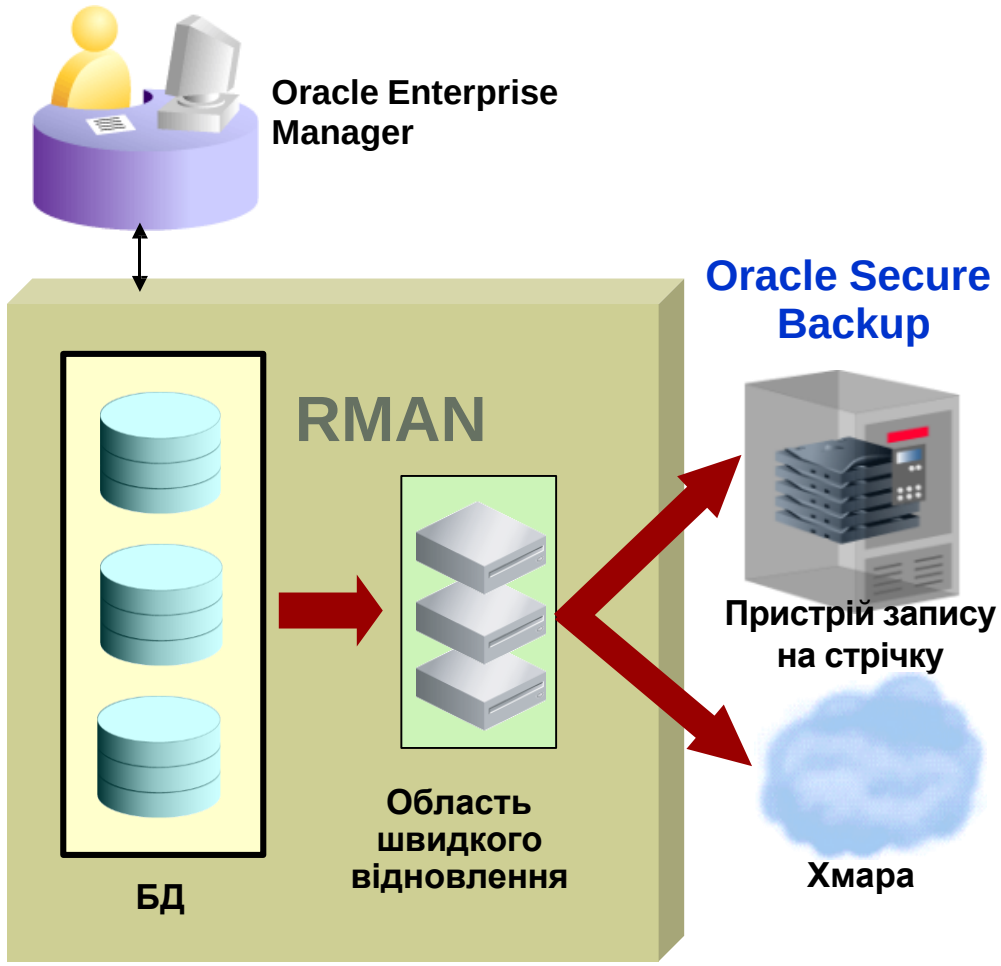
Рішення Oracle для резервного копіювання та відновлення



Основа резервного копіювання та відновлення
Повне рішення Oracle від диска до стрічки



Oracle Recovery Manager (RMAN)
Інтегрований в Oracle механізм резервного копіювання та відновлення



- Внутрішнє знання форматів файлів бази даних і процедур відновлення
  - Перевірка блоку
  - Онлайн-відновлення на рівні блоку
  - Відновлення табличного простору/файлу даних
  - Онлайн, багатопотокове резервне копіювання
  - Невикористане стиснення блоків
  - Нативне шифрування
- Інтегроване резервне копіювання на диск, стрічку та хмару з використанням Fast Recovery Area та Oracle Secure Backup

Рішення Oracle для резервного копіювання та відновлення

❑ *Менеджер відновлення (RMAN)*

- Recovery Manager повністю інтегрований з базою даних Oracle для виконання ряду дій з резервного копіювання та відновлення.
- Можна отримати доступ до RMAN через командний рядок або через Oracle Enterprise Manager.

❑ *Резервне копіювання та відновлення, кероване користувачем*

- У цьому рішенні виконується резервне копіювання та відновлення за допомогою комбінації команд операційної системи хоста та команд відновлення SQL*Plus.
- Користувач несе відповідальність за визначення всіх аспектів того, коли та як виконуються резервне копіювання та відновлення.

Порівняння функцій методів резервного копіювання$_1$

Характеристика	Recovery Manager	Керований користувачами	Data Pump Export
Резервне копіювання закритих баз даних	Підтримується	Підтримується	Не підтримується
Резервне копіювання відкритих баз даних	Підтримується	Підтримується	Потрібні відкат або скасування сегментів для створення узгоджених резервних копій
Інкрементні копії	Підтримується	Не підтримується	Не підтримується
Детекція хибних блоків	Підтримується	Не підтримується	Підтримується
Автоматичне визначення файлів, які потрібно включити в резервну копію	Підтримується	Не підтримується	Не застосовується
Репозиторій резервних копій	Підтримується	Не підтримується	Не підтримується

Порівняння функцій методів резервного копіювання$_2$

Характеристика	Recovery Manager	Керований користувачами	Data Pump Export
Резервне копіювання в медіа-менеджер	Підтримується	Підтримується	Не підтримується
Резервна копія файлу параметрів ініціалізації	Підтримується	Підтримується	Не підтримується
Резервне копіювання паролів і мережевих файлів	Не підтримується	Підтримується	Не підтримується
Платформено-незалежна мова для резервного копіювання	Підтримується	Не підтримується	Підтримується

Запуск RMAN

Для запуска RMAN потрібно виконати наступні команди:

- `sudo docker exec -it 23cfree /bin/bash`
- `rman target “free”`

```
bash-4.4$ rman target "free"

Recovery Manager: Release 23.0.0.0.0 - Developer-Release on Thu Nov 2 14:52:19 2
023
Version 23.2.0.0.0

Copyright (c) 1982, 2023, Oracle and/or its affiliates.  All rights reserved.

target database Password:
connected to target database: FREE (DBID=1415845140)

RMAN>
```

Зміна конфігурації резервного копіювання RMAN

```
RMAN> configure channel device type disk format
'/backup/rman/full_%u_%s_%p';

RMAN> configure retention policy to recovery window of 7
days;
```

- %U – генерує унікальне ім’я
- %d – для DB_NAME
- %t – для мітки часу набору резервних копій
- %s – для номера резервного набору
- %p – для резервного номера частини

```
RMAN> configure channel device type disk format '/backup/rman/full_%u_%s_%p';
configure channel device type disk format '/backup/rman/full_%u_%s_%p';
using target database control file instead of recovery catalog
new RMAN configuration parameters:
CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT   '/backup/rman/full_%u_%s_%p';
new RMAN configuration parameters are successfully stored


RMAN> configure retention policy to recovery window of 7 days;
configure retention policy to recovery window of 7 days;
new RMAN configuration parameters:
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
new RMAN configuration parameters are successfully stored
```

Інкрементне резервне копіювання$_1$

- Зберігає лише блоки, змінені після попереднього резервного копіювання
- Забезпечує більш компактні резервні копії та швидше відновлення
- Зменшується необхідність повторного виконання під час відновлення носія файлу даних

Інкрементне резервне копіювання$_2$

- Створення додаткової резервної копії рівня 0, яка слугуватиме основою для стратегії інкрементального резервного копіювання:

```
backup incremental level 0 database;
```

- Створення накопичувальної інкрементної резервної копії рівня 1:

```
backup incremental level 1 cumulative database;
```

- Створення диференційної інкрементної резервної копії рівня 1:

```
backup incremental level 1 database;
```

Перевірка та перехресна перевірка файлів бази даних і резервних копій

- Перевірка всіх файлів бази даних і заархівованих файлів журналу відновлення на фізичні і логічні спотворення:

```
BACKUP VALIDATE CHECK LOGICAL DATABASE ARCHIVELOG ALL;
```

- Перевірка окремих блоків даних:

```
VALIDATE DATAFILE 4 BLOCK 10 TO 13;
```

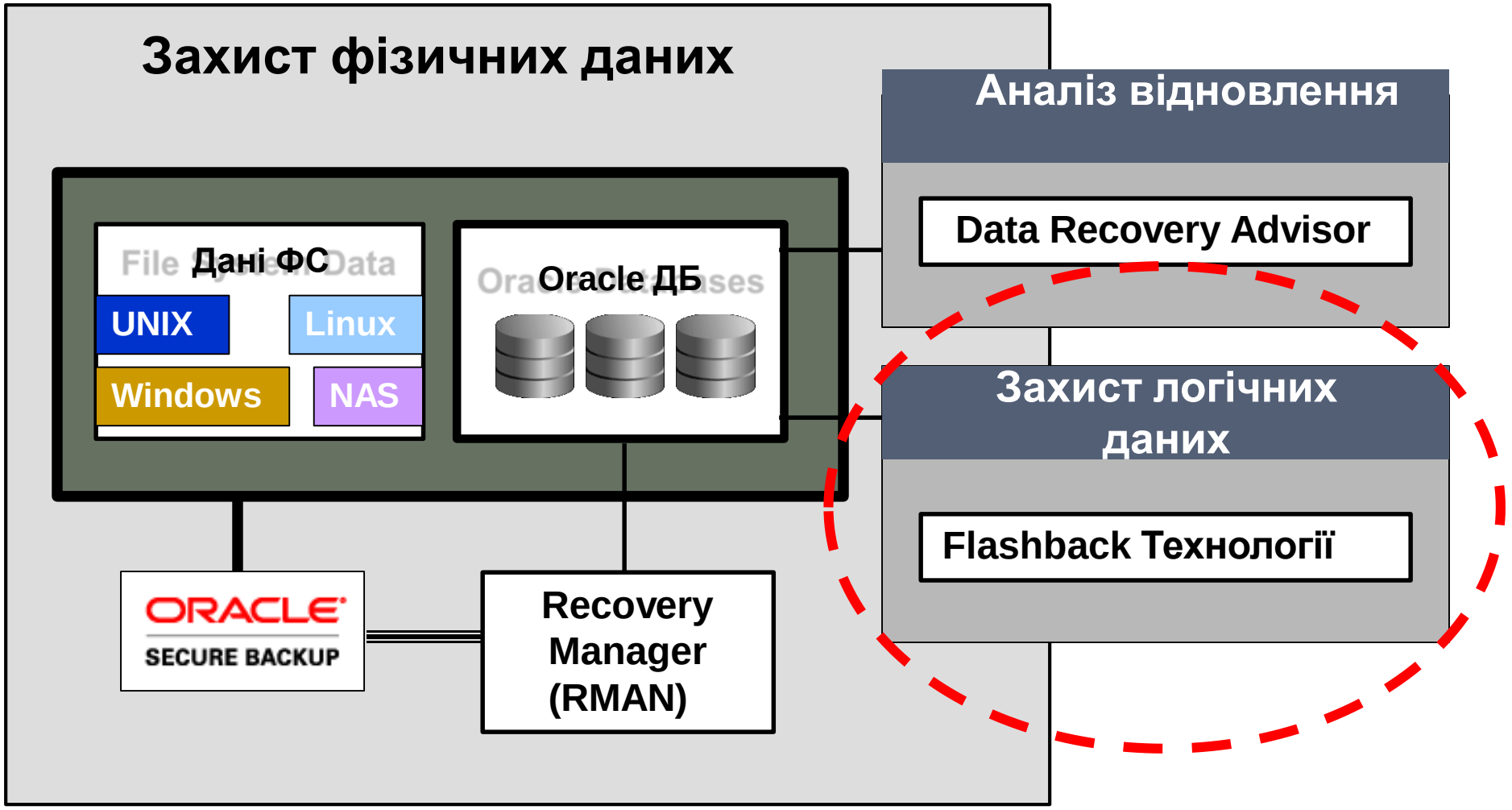
- Перевірка наборів резервних копій:

```
VALIDATE BACKUPSET 3;
```

- CROSSCHECK синхронізує логічні записи RMAN резервних копій з файлами на носії:

```
CROSSCHECK BACKUP;
CROSSCHECK COPY;
```

Захист логічних даних
Швидке «перемотування» логічних помилок



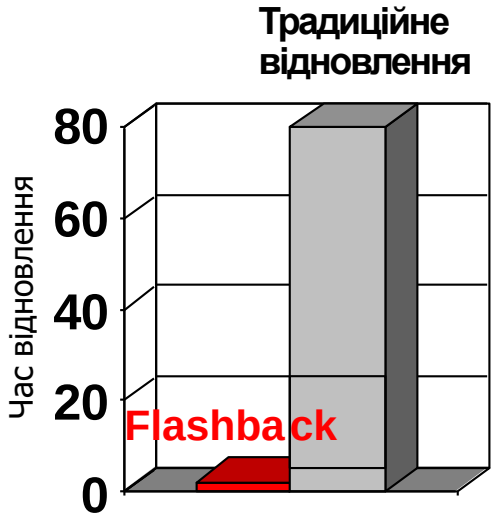
Технології Flashback
Виявлення та виправлення помилок

Flashback революціонізує підхід до відновлення помилок:

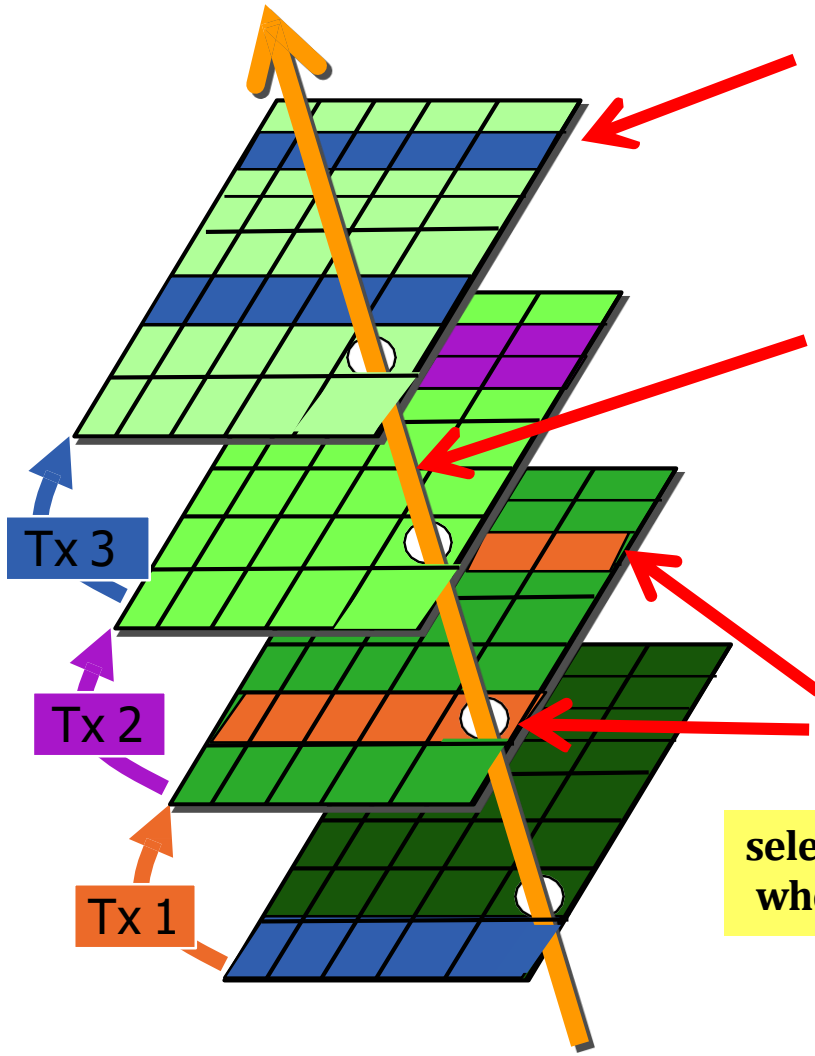
- ❑ Перегляд «хороших» даних за минулий момент часу
- ❑ Проста перемотка та зміна даних
- ❑ Час на виправлення помилки дорівнює часу на помилку

Correction Time = Error Time + ~~f(DB_SIZE)~~

- ❑ Низький вплив
- ❑ Чудовий інструмент для налаштування баз даних QA, Dev і Training
- ❑ Flashback простий – прості команди, жодних складних процедур



Дослідження помилок із Flashback



- **Запит Flashback**
  - Запит на всі дані в певний момент часу

```
select * from Salary AS OF ‘12:00 P.M.’ where ...
```

- **Запит версії Flashback**
  - Переглянути всі версії рядка за часовим діапазоном
  - Переглянути транзакції, які змінили рядок

```
select * from Salary
 ‘12:00 PM’ and ‘2:00 PM’ where ...
```

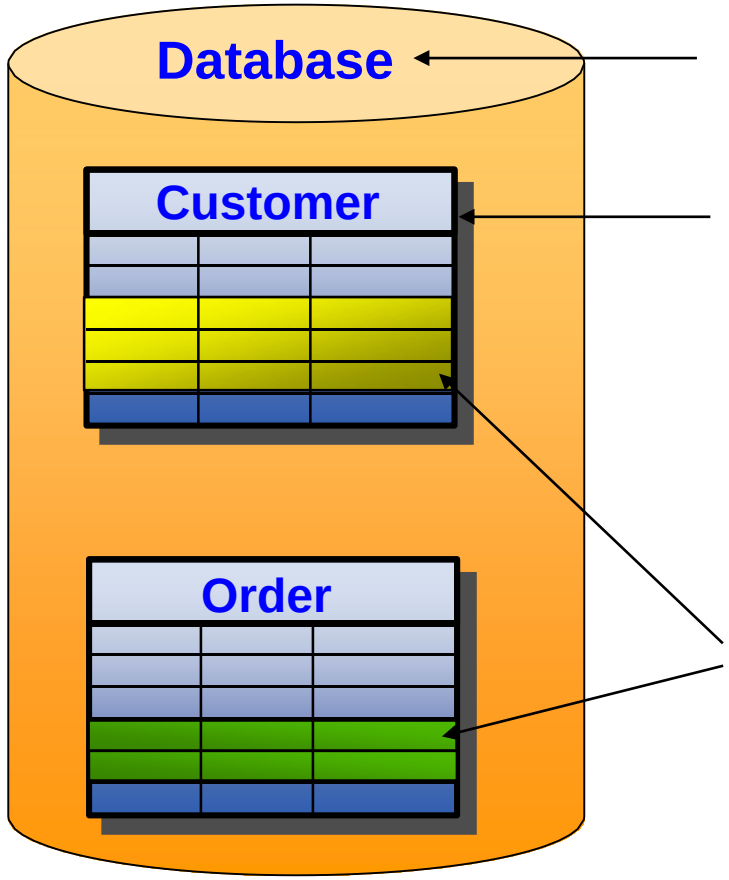
- **Запит транзакції Flashback**
  - Переглянути всі зміни, внесені трансакцією

```
select * from
 where xid = HEXTORAW(‘00020003000002D’);
```

Перемотування бази даних за допомогою Flashback Database

- Можна використовувати Oracle Flashback Database, щоб перемотати всю базу даних до минулого часу
- На відміну від відновлення носіїв, не потрібно відновлювати файли даних, щоб повернути базу даних до минулого стану
- Щоб використати команду RMAN FLASHBACK DATABASE, база даних має бути попередньо налаштована для створення журналів спогадів

Виправлення помилок за допомогою Flashback



- **Flashback Database** – відновлення бази даних до стану будь-якого моменту часу
- **Flashback Table** – відновлення вмісту таблиць до будь-якого моменту часу (на основі скасування)
- **Flashback Drop** – відновлення випадково видалених таблиць (на основі вільного місця в табличному просторі)
- **Flashback Transaction** – відкликання транзакції та всіх наступних конфліктних транзакцій (на основі redo)

Донецький національний технічний університет

Тема 8. Робота з SQL.
Лекція 12. Робота з SQL. Складні запити.

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Використання групових функцій
- Використання пропозицій об'єднання (JOIN)
- Використання підзапитів для вирішення запитів
- Використання операторів множин

Використання групових функцій

План розділу

- Групові функції:
  - Типи і синтаксис
  - Використання AVG, SUM, MIN, MAX, COUNT
  - Використання ключового слова DISTINCT у групових функціях
  - Нульові значення у функції групи
- Угруповання рядків:
  - Пропозиція GROUP BY
  - Пропозиція HAVING
- Групові функції вкладення

Що таке групові функції?

- Групові функції впливають на набори рядків з метою надання одного угрупованого результату.

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	90	24000
2	90	17000
3	90	17000
4	60	9000
5	60	6000
6	60	4200
7	50	5800
8	50	3500
9	50	3100
10	50	2600
...		
18	20	6000
19	110	12000
20	110	8300

Максимальна зарплата в таблиці EMPLOYEES

MAX(SALARY)
24000

Типи групи функцій

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

Групові функції: синтаксис

```
SELECT      group_function(column), ...
FROM        table
[WHERE      condition]
[ORDER BY   column];
```

Використання функцій AVG і SUM

- Можна використовувати AVG і SUM для числових даних.

```
SELECT  AVG(salary), MAX(salary),
        MIN(salary), SUM(salary)
FROM    employees
WHERE   job_id LIKE '%REP%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

Використання функцій MIN і MAX

- Можна використовувати MIN і MAX для числових, символьних типів даних та типу дати.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM       employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

Використання функції COUNT

- COUNT (*) повертає кількість рядків у таблиці:

1

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(*)
1	5

- COUNT(expr) повертає кількість рядків з ненульовими значеннями для expr:

2

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

	COUNT(COMMISSION_PCT)
1	3

Використання ключового слова DISTINCT

- COUNT(DISTINCT expr) повертає число відмінних ненульових значень expr.
- Вивести на екран число відмінного відділу оцінює в таблиці EMPLOYEES:

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

Групи функцій і нульові значення

- Групові функції ігнорують нульові значення у стовпці:

1

```
SELECT AVG(commission_pct)
FROM   employees;
```

	AVG(COMMISSION_PCT)
1	0.2125

- NVL застосовують у групових функціях з метою включення нульових значень:

2

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

Створення груп даних

EMPLOYEES

	DEPARTMENT_ID	SALARY	
1	10	4400	4400
2	20	13000	9500
3	20	6000	
4	50	5800	
5	50	2500	
6	50	2600	3500
7	50	3100	
8	50	3500	
9	60	4200	6400
10	60	6000	
11	60	9000	
12	80	11000	10033
13	80	10500	
14	80	8600	
...			
19	110	12000	
20	(null)	7000	

Середня зарплата в таблиці EMPLOYEES для кожного відділу

	DEPARTMENT_ID	AVG(SALARY)
1	10	4400
2	20	9500
3	50	3500
4	60	6400
5	80	10033.333333333333...
6	90	19333.333333333333...
7	110	10150
8	(null)	7000

Створення груп даних: синтаксис GROUP BY

```
SELECT     column, group_function(column)
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

- Можна розділити рядки таблиці на менші групи за допомогою пропозиції GROUP BY.

Застосування пропозиції GROUP BY

- Всі стовпці в списку SELECT, які не використовуються у групових функціях, повинні бути перераховані в пункті GROUP BY.

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id ;
```

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.33333333333...
3	20	9500
4	110	10150
5	50	3500
6	80	10033.33333333333...
7	60	6400
8	10	4400

Використання пункту GROUP BY

- Стовпець GROUP BY не повинен бути в списку SELECT.

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```

	AVG(SALARY)
1	7000
2	19333.333333333333333333...
3	9500
4	10150
5	3500
6	10033.333333333333333333...
7	6400
8	4400

Групування за декількома стовпцями

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_MAN	5800
5	50	ST_CLERK	2500
6	50	ST_CLERK	2600
7	50	ST_CLERK	3100
8	50	ST_CLERK	3500
9	60	IT_PROG	4200
10	60	IT_PROG	6000
11	60	IT_PROG	9000
12	80	SA_REP	11000
13	80	SA_MAN	10500
14	80	SA_REP	8600
...			
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

Зарплата в EMPLOYEES для кожного завдання, згрупованого за відділом.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

Використання пропозиції GROUP BY для декількох стовпців

```
SELECT   department_id dept_id, job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

Використання неприпустимих запитів груп функцій

- Будь-який стовпець або вираз у списку SELECT, який не є агрегатною функцією, повинні бути в пункті GROUP BY:

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"

Пункт GROUP BY повинен бути доданий, щоб опрацювати прізвища для кожного department_id.

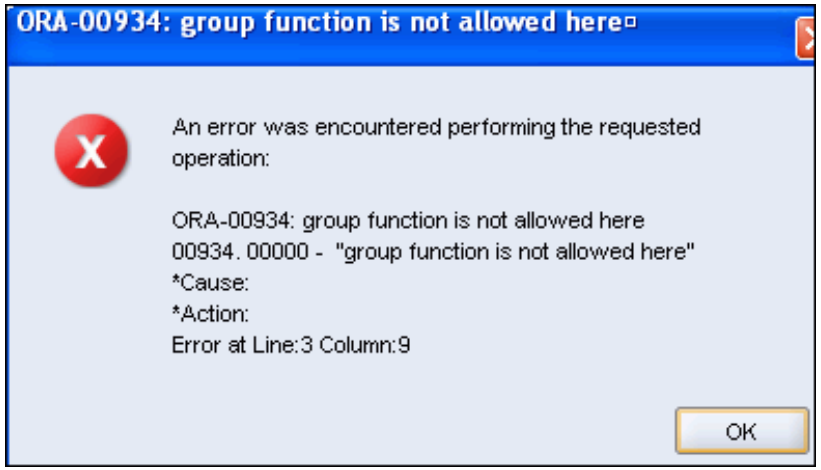
```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```

ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"

Або додати job_id в GROUP BY, або видалити стовпець job_id з списку SELECT.

Використання неприпустимих запитів групових функцій

- НЕ можна використовувати оператор WHERE для обмеження виводу групових функцій.
- Для обмеження виводу групових функцій слід використовувати пропозицію HAVING.
- НЕ можна використовувати групові функції в пропозиції WHERE.




```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
```

НЕ можна використовувати оператор WHERE для обмеження виведення групової функції

Обмеження результатів груп

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600

...

18	110	8300
19	110	12000
20	(null)	7000

Максимальна зарплата за відділ у разі, якщо вона більша за 10,000$

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	80	11000
3	90	24000
4	110	12000

Обмеження результатів груп з пунктом HAVING

- Коли використовується пропозиція HAVING, сервер Oracle обмежує групи наступним чином:
  1. Рядки групуються.
  2. Застосовується групова функція.
  3. Групи, відповідні пункту HAVING, виводяться на екран.

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

Використання пропозиції HAVING

```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY  department_id
HAVING    MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12000
4	80	11000

Використання пункту HAVING

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

Функції групи вкладення

- Вивести на екран максимальну середню зарплату:

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

	MAX(AVG(SALARY))
1	19333.333333333333333333333333333333333

Підсумки розділу

- В цьому розділі розглянуті наступні питання:
  - Використання групових функцій COUNT, MAX, MIN, SUM, і AVG
  - Запис запитів, які використовують пропозицію GROUP BY
  - Запис запитів, які використовують пропозицію HAVING

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

Використання пропозицій об'єднання (JOIN)

План розділу

- Типи об'єднань та їх синтаксис
- Звичайне об'єднання:
  - використовуючи оператор USING
  - використовуючи оператор ON
- Самообє’днання
- Об'єднання, які базуються на нерівності
- Зовнішні об'єднання:
  - Ліве зовнішнє об'єднання
  - Праве зовнішнє об'єднання
  - Повне зовнішнє об'єднання
- Декартовий добуток
  - Перехресне об'єднання

Отримання даних з декількох таблиць

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
...			
18	202	Fay	20
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
5	144	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

Типи об'єднань

- Об'єднання, які сумісні зі стандартом SQL: 1999 включають в себе:
  - Звичайні об'єднання:
    - запит `NATURAL JOIN`
    - використовуючи оператор `USING`
    - використовуючи оператор `ON`
  - Зовнішні об'єднання:
    - Ліве зовнішнє об'єднання
    - Праве зовнішнє об'єднання
    - Повне зовнішнє об'єднання
  - Перехресні об'єднання

Об'єднання таблиць за допомогою синтаксису SQL: 1999

- Використання об'єднання для отримання даних з більш, ніж однієї таблиці:

```
SELECT   table1.column, table2.column
FROM     table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

Кваліфікація неоднозначних імен стовпців

- Використовуйте префікси таблиць, щоб вибрати імена стовпців, які знаходяться в декількох таблицях.
- Використовуйте префікси таблиць для підвищення продуктивності.
- Замість повних імен таблиць, використовуйте псевдоніми таблиць.
- Псевдоніми таблиць дають таблиці коротке ім'я:
  - робить SQL код менше, використовує менше пам'яті
- Використовуйте псевдоніми стовпців для виділення стовпців, які мають однакові імена, але знаходяться в різних таблицях.

Створення звичайних об'єднань

- Запит NATURAL JOIN засновується на всіх колонках в двох таблицях, які мають однакове ім'я.
- Він вибирає рядки з двох таблиць, які мають однакові значення в усіх відповідних стовпцях.
- Якщо у колонок однакові імена, але різні типи даних, буде повернуто повідомлення про помилку.

Отримання записів, використовуючи звичайні об'єднання


```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

Створення об'єднань з оператором USING

- Якщо декілька стовпців мають однакові імена, але типи даних не збігаються, звичайне об'єднання можна застосувати з використанням оператору USING для вибору стовпців, які повинні використовуватися в об'єднанні, що базується на рівності.
- Оператор USING краще використовувати для встановлення відповідності одного стовпця, коли є більше одного відповідного стовпця.
- Звичайне об'єднання і оператор USING є взаємовиключними.

Об'єднання імен стовпців

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
100	90
101	90
102	90
103	60
104	60
107	60
124	50
141	50
142	50
143	50
144	50
149	80
174	80
176	80

...

Зовнішній ключ

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Первинний ключ

Отримання записів, використовуючи оператор USING

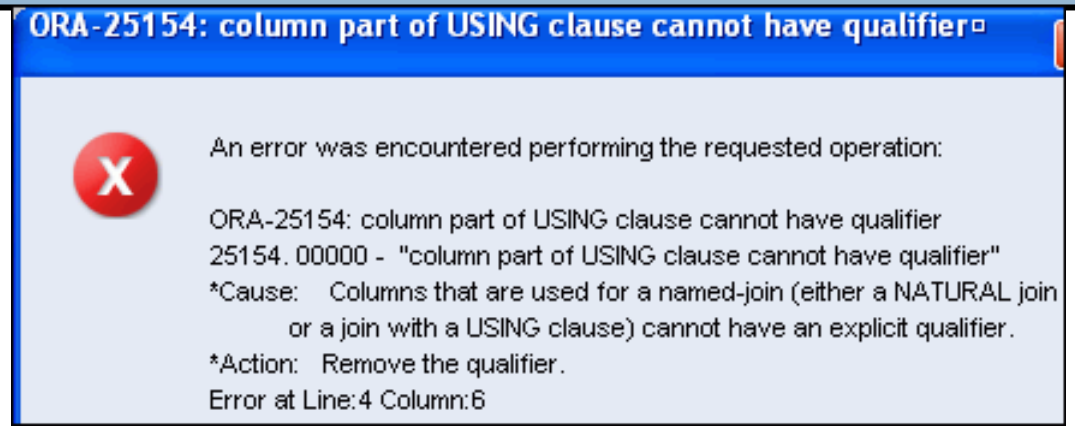
```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	124	Mourgos	1500	50
5	144	Vargas	1500	50
6	143	Matos	1500	50
7	142	Davies	1500	50
8	141	Rajs	1500	50
9	107	Lorentz	1400	60
10	104	Ernst	1400	60
...				
19	205	Higgins	1700	110

Використання псевдонімів таблиць з використанням оператора USING

- Не кваліфікуйте стовпець, який використовується USING.
- Якщо той же самий стовпець використовується ще де-небудь в виразі SQL, не давайте йому псевдонім.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE d.location_id = 1400;
```



Створення об'єднань за допомогою оператора ON

- Умовою для звичайного об'єднання в основному є об'єднання, що базується на рівності всіх стовпців з однаковими іменами.
- Оператор ON краще використовувати для вказівки довільних умов або для вказівки стовпців, які необхідно об'єднати.
- Умова об'єднання відокремлена від інших умов пошуку.
- Умова ON дозволяє зробити код більш зрозумілим.

Отримання даних за допомогою оператора ON

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400

...

Створення тристороннього об'єднання за допомогою оператора ON

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping

...

Применение дополнительных условий к объединению

- Використайте умову AND або WHERE для застосування додаткових умов:

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149 ;
```

Or

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
WHERE   e.manager_id = 149 ;
```

Об'єднання таблиці з самою собою

EMPLOYEES (WORKER)

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos
141	Rajs
142	Davies
143	Matos

...

MANAGER_ID в таблиці WORKER рівний стовпцю EMPLOYEE_ID в таблиці MANAGER.

Самооб'єднання з використанням оператора ON


```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON    (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King
9	Hartstein	King
10	De Haan	King

...

Об'єднання, які базуються на нерівності

EMPLOYEES

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hunold	9000
5	Ernst	6000
6	Lorentz	4200
7	Mourgos	5800
8	Rajs	3500
9	Davies	3100
10	Matos	2600
...		
19	Higgins	12000
20	Gietz	8300

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

Таблиця JOB_GRADES визначає LOWEST_SAL і HIGHEST_SAL – межі значень для кожного GRADE_LEVEL. Отже, стовпець GRADE_LEVEL може використовуватися для присвоєння класів кожному співробітнику.

Отримання записів за допомогою об'єднань, що базуються на нерівності

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

Отримання записів за допомогою зовнішнього об'єднання без відповідності

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	90	King
2	90	Kochhar
3	90	De Haan
4	60	Hunold
5	60	Ernst
6	60	Lorentz
7	50	Mourgos
8	50	Rajs
9	50	Davies
10	50	Matos
...		
19	110	Higgins
20	110	Gietz

В департаменті 190 немає робітників.

Внутрішні проти зовнішніх об'єднань

- В SQL:1999, об'єднання двох таблиць, яке повертає тільки відповідні рядки, називається внутрішнім об'єднанням.
- Об'єднання двох таблиць, яке повертає результат внутрішнього об'єднання, а також невідповідні рядки з лівої (або правої) таблиці називається лівим (правим) зовнішнім об'єднанням.
- Об'єднання двох таблиць, яке повертає результат внутрішнього об'єднання, а також результати лівого і правого зовнішніх об'єднань називається повним зовнішнім об'єднанням.

Ліве зовнішнє об'єднання

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

Праве зовнішнє об'єднання

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

Повне зовнішнє об'єднання


```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

Декартовий добуток

- Декартовий добуток формується коли:
  - Умова об'єднання відсутня
  - Умова об'єднання є невірною
  - Всі рядки першої таблиці приєднуються до всіх рядків у другій таблиці
- Для запобігання декартового добутку, необхідно використовувати правильну умову об'єднання.

Отримання декартового добутку

EMPLOYEES (20 рядків)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	60
...			
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS (8 рядків)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

Декартовий добуток: 20 x 8 = 160 рядків

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	100	90	1700
2	101	90	1700
3	102	90	1700
4	103	60	1700
...			
159	205	110	1700
160	206	110	1700

Створення перехресних об'єднань

- Оператор CROSS JOIN повертає векторний добуток двох таблиць.
- Він також називається декартовим добутком двох таблиць.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration

...

159	Whalen	Contracting
160	Zlotkey	Contracting

Висновок за розділом

- В цьому розділі було розглянуто як використовувати об'єднання для відображення даних з декількох таблиць, використовуючи:
  - Об'єднання, що базуються на рівності
  - Об'єднання, що базуються на нерівності
  - Зовнішні об'єднання
  - Самоб’єднання
  - Перехресні об'єднання
  - Звичайні об'єднання
  - Повні (або двохсторонні) об'єднання

Використання підзапитів для вирішення запитів

План розділу

- Підзапити: типи, синтаксис та керівництво користувача
- Однорядкові підзапити:
  - Функції групування в підзапиті
  - Оператор HAVING з підзапитами
- Багаторядкові підзапити
  - Використання оператора ALL або ANY
- Нульові значення в підзапиті

Використання підзапитів для вирішення проблем

- У кого зарплата більше ніж у Абеля?

Основний запит:

У яких працівників зарплата більше ніж у Абеля?

Підзапит:

Яка зарплата у Абеля?

Синтаксис підзапитів

```
SELECT    select_list
FROM      table
WHERE     expr operator
                    (SELECT       select_list
                    FROM          table);
```

- Підзапит (внутрішній запит) виконується до головного запиту (зовнішнього запиту).
- Результат підзапиту використовується головним запитом.

Використання підзапиту

```
SELECT last_name, salary
FROM   employees
WHERE  salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Abel');
```

11000

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000
5	Higgins	12000

Керівництво до використання підзапитів

- Заключайте підзапити в дужки.
- Розміщуйте підзапити на правильній стороні порівнюючої умови для читабельності. Тим не менш, підзапит може з'являтися на будь-якій стороні порівнюючого оператора.
- Необхідно використовувати однорядкові оператори з однорядковими підзапитами і багаторядкові оператори з багаторядковими підзапитами.

Типи підзапитів

- Однорядковий підзапит

- Багаторядковий підзапит

Однорядкові підзапити

- Повертає тільки один рядок
- Використовує однорядкові оператори порівняння

Operator	Meaning
=	Дорівнює
>	Більше
>=	Не менше
<	Менше
<=	Не більше
<>	Не дорівнює

Виконання однорядкових підзапитів


```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                  (SELECT job_id
                   FROM   employees
                   WHERE  last_name = 'Taylor')
AND    salary >
                  (SELECT salary
                   FROM   employees
                   WHERE  last_name = 'Taylor');
```

SA_REP

8600

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

Використання функцій групування в підзапиті

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

2500

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

Оператор HAVING з підзапитами

- Сервер Oracle спочатку виконує підзапити.
- Сервер Oracle повертає результати в оператор HAVING головного запиту.

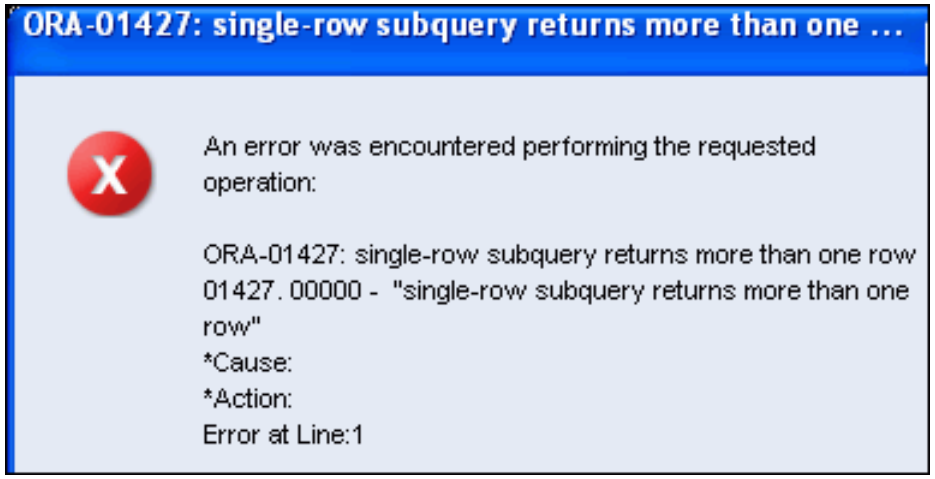
```
SELECT    department_id, MIN(salary)
FROM      employees
GROUP BY  department_id
HAVING    MIN(salary) >
                                  (SELECT MIN(salary)
                                   FROM   employees
                                   WHERE  department_id = 50);
```

2500

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	90	17000
3	20	6000
...		
7	10	4400

Що неправильно в даному виразі?


```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                (SELECT   MIN(salary)
                 FROM     employees
                 GROUP BY department_id);
```



Однорядковий оператор з багаторядковим підзапитом

Жоден з рядків не повертається внутрішнім підзапитом


```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');
```

```
0 rows selected
```

Підзапит не повертає рядків, так як немає робітника по імені “Haas.”

Багаторядкові підзапити

- Повертає більше, ніж один рядок
- Використовує багаторядкові оператори порівняння

Operator	Meaning
`IN`	Дорівнює будь-якому члену списку
`ANY`	Перед ним повинні бути =, !=, >, <, <=, >=. Порівнює значення з кожним значенням в списку або повернутим запитом. Дорівнює `FALSE` якщо запит не повертає рядків.
`ALL`	Перед ним повинні бути =, !=, >, <, <=, >=. Порівнює значення з кожним значенням в списку або повернутим запитом. Дорівнює `TRUE` якщо запит не повертає рядків.

Використання оператора ANY в багаторядкових підзапитах

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400

...

9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

Використання оператора ALL в багаторядкових підзапитах

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

Нульові значення в підзапиті


```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                            (SELECT mgr.manager_id
                             FROM   employees mgr);
```

```
0 rows selected
```

Висновок за розділом

- В даному розділі було розглянуто, як:
  - Визначати, коли підзапит може допомогти у вирішенні проблеми
  - Писати підзапити, коли запит засновується на невідомих значеннях

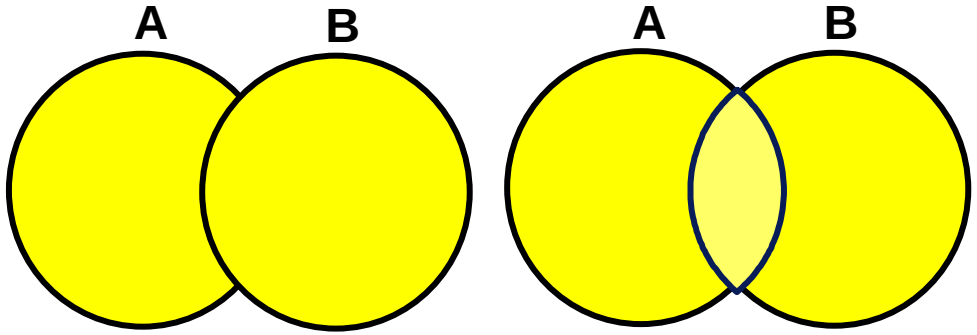
```
SELECT   select_list
FROM     table
WHERE    expr operator
                    (SELECT select_list
                    FROM    table);
```

Використання операторів множин

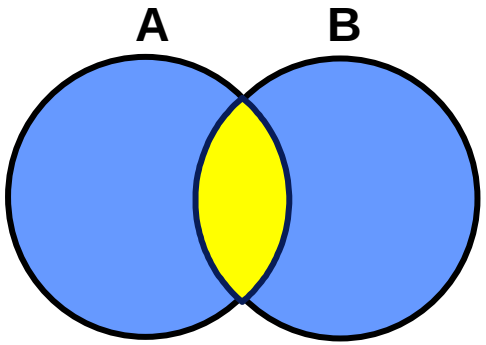
План розділу

- Оператори множин: типи та вказівки
- Таблиці, які використовуються на занятті
- UNION і UNION ALL оператори
- INTERSECT оператор
- MINUS оператор
- Узгодження операторів SELECT
- Використання оператора ORDER BY в операціях над множинами

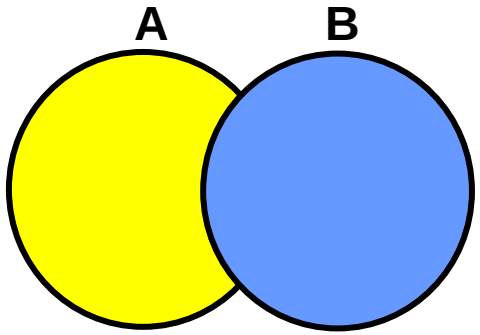
Оператори множин



UNION/UNION ALL



INTERSECT



MINUS

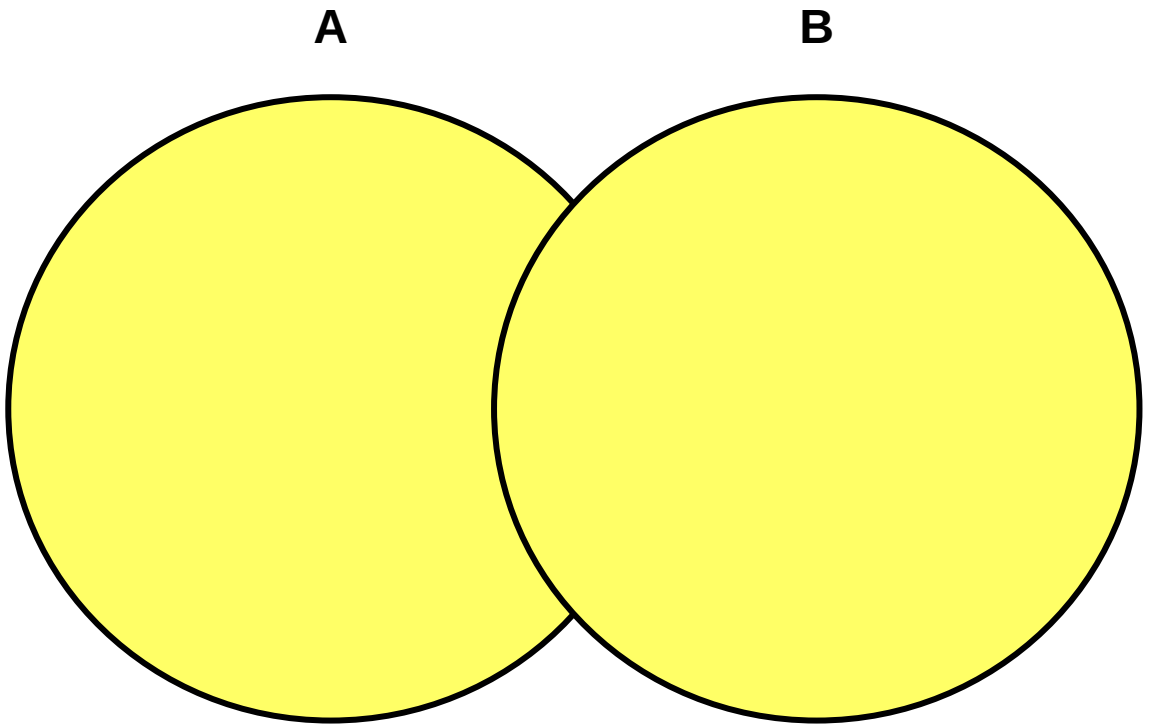
Оператори множин: Вказівки

- Вирази в списках SELECT повинні збігатися за числом.
- Тип даних кожного стовпця у другому запиті повинен збігатися з типом даних відповідного стовпця в першому запиті.
- Дужки можуть використовуватися, щоб змінити послідовність виконання.
- Оператор ORDER BY може з'явитися тільки в самому кінці виразу.

Oracle сервер і оператори множин

- Повторювані рядки автоматично видаляються, за винятком у UNION ALL.
- Імена стовпців з першого запиту з'являються в результаті.
- Вихідні дані відсортовуються в порядку зростання за замовчуванням, за винятком UNION ALL.

Оператор UNION



Оператор UNION повертає всі рядки, вибрані яким-небудь запитом, з усуненням повторюваних рядків

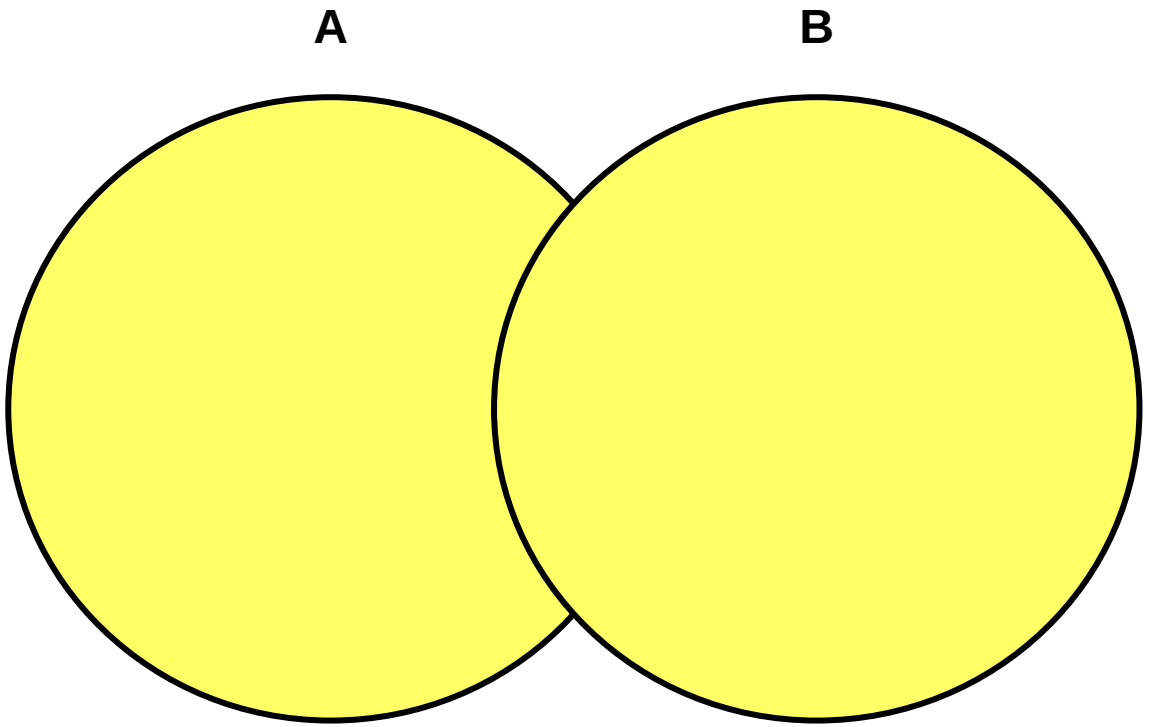
Використання оператору UNION

- Відобразити поточні і попередні дані про роботу всіх робітників. Відобразити інформацію про кожного робітника тільки один раз.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID
1	100	AD_PRES
2	101	AC_ACCOUNT
...		
22	200	AC_ACCOUNT
23	200	AD_ASST
24	201	MK_MAN
...		

Оператор UNION ALL



Оператор `UNION ALL` використовується для повернення всіх рядків з багатьох таблиць.

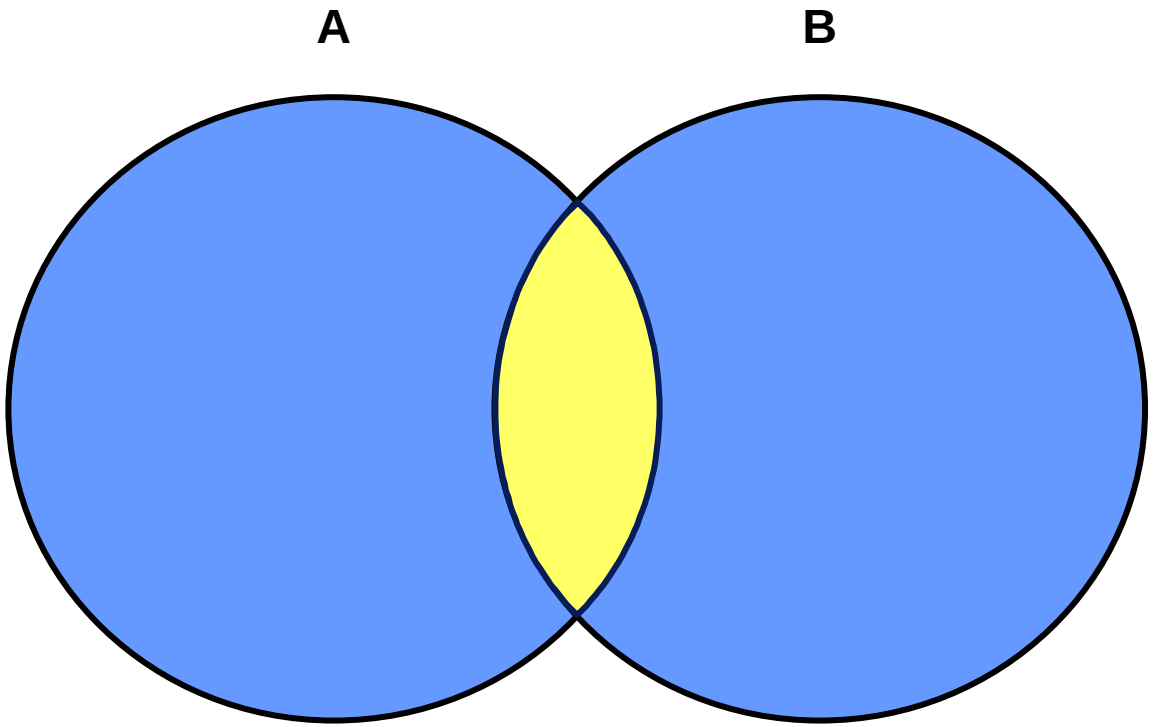
Використання оператору UNION ALL

- Відображення поточних і попередніх відділів усіх співробітників.

```
SELECT employee_id, job_id, department_id
FROM   employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM   job_history
ORDER BY  employee_id;
```

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100	AD_PRES	90
...			
16	144	ST_CLERK	50
17	149	SA_MAN	80
18	174	SA_REP	80
19	176	SA_REP	80
20	176	SA_MAN	80
21	176	SA_REP	80
22	178	SA_REP	(null)
...			
30	206	AC_ACCOUNT	110

Оператор INTERSECT



Оператор `INTERSECT` використовується для повернення всіх рядків, які є загальними для декількох запитів.

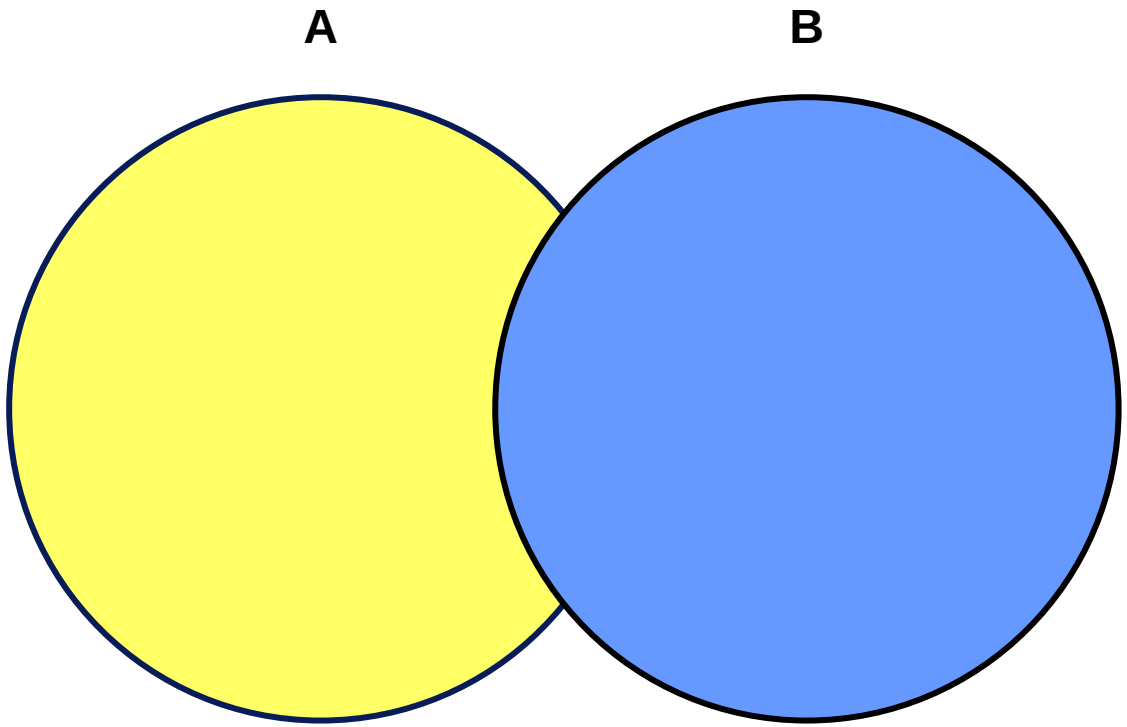
Використання оператору INTERSECT

- Відобразити employee IDs і job IDs тих робітників, які працюють на тій же самій посаді (тобто, вони міняли роботу, але повернулися на ту ж саму роботу, на якій вони працювали).

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

Оператор MINUS



Оператор MINUS використовується для повернення всіх рядків, які вибираються першим запитом, але не присутні у результатах другого запиту

Використання оператору MINUS

- Відобразити employee IDs тих робітників, які не змінили своєї роботи жодного разу.

```
SELECT employee_id
FROM   employees
MINUS
SELECT employee_id
FROM   job_history;
```

	EMPLOYEE_ID
1	100
2	103
3	104
4	107
5	124
...	
14	205
15	206

Узгодження операторів SELECT

- Використовуючи оператор UNION відобразити location ID, назву відділу, і штат де він знаходиться.
- Необхідно узгодити типи даних (використовуючи функцію TO_CHAR або будь-яку іншу функцію перетворення), коли стовпець не існує в одній або іншій таблиці.

```
SELECT location_id, department_name "Department",
   TO_CHAR(NULL) "Warehouse location"
FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "Department",
   state_province
FROM locations;
```

Узгодження операторів SELECT: Приклад

- Використовуючи оператор UNION, відобразити employee ID, job ID, і зарплату усіх робітників.

```
SELECT employee_id, job_id,salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID	SALARY
1	100	AD_PRES	24000
2	101	AC_ACCOUNT	0
3	101	AC_MGR	0
4	101	AD_VP	17000
5	102	AD_VP	17000

...

29	205	AC_MGR	12000
30	206	AC_ACCOUNT	8300

Використання оператора ORDER BY в операціях над множинами

- Оператор ORDER BY може з'являтися тільки один раз у кінці складного запиту.
- Компонентні запити не можуть мати індивідуальні оператори ORDER BY.
- Оператор ORDER BY розпізнає тільки стовпці першого запиту SELECT.
- За замовчуванням, перший стовпець першого запиту SELECT використовується для сортування вихідних даних у порядку зростання.

Висновок за розділом

- В даному розділі були розглянуті наступні питання:
  - Повернення усіх різних рядків за допомогою оператора UNION
  - Повернення усіх різних рядків, включаючи дублікати, за допомогою оператора UNION ALL
  - Повернення усіх рядків, які є загальними для обох запитів за допомогою оператора INTERSECT
  - Повернення усіх різних рядків, вибраних першим запитом, але не другим за допомогою оператора MINUS
  - Використання оператору ORDER BY в самому кінці виразу

Донецький національний технічний університет

Тема 9. Мова програмування PL/SQL. Лекція 13. Вступ до PL/SQL.

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Вступ до PL/SQL
- Оголошення змінних в PL/SQL
- Написання виконуваних операторів
- Взаємодія з сервером Oracle

Вступ в PL/SQL

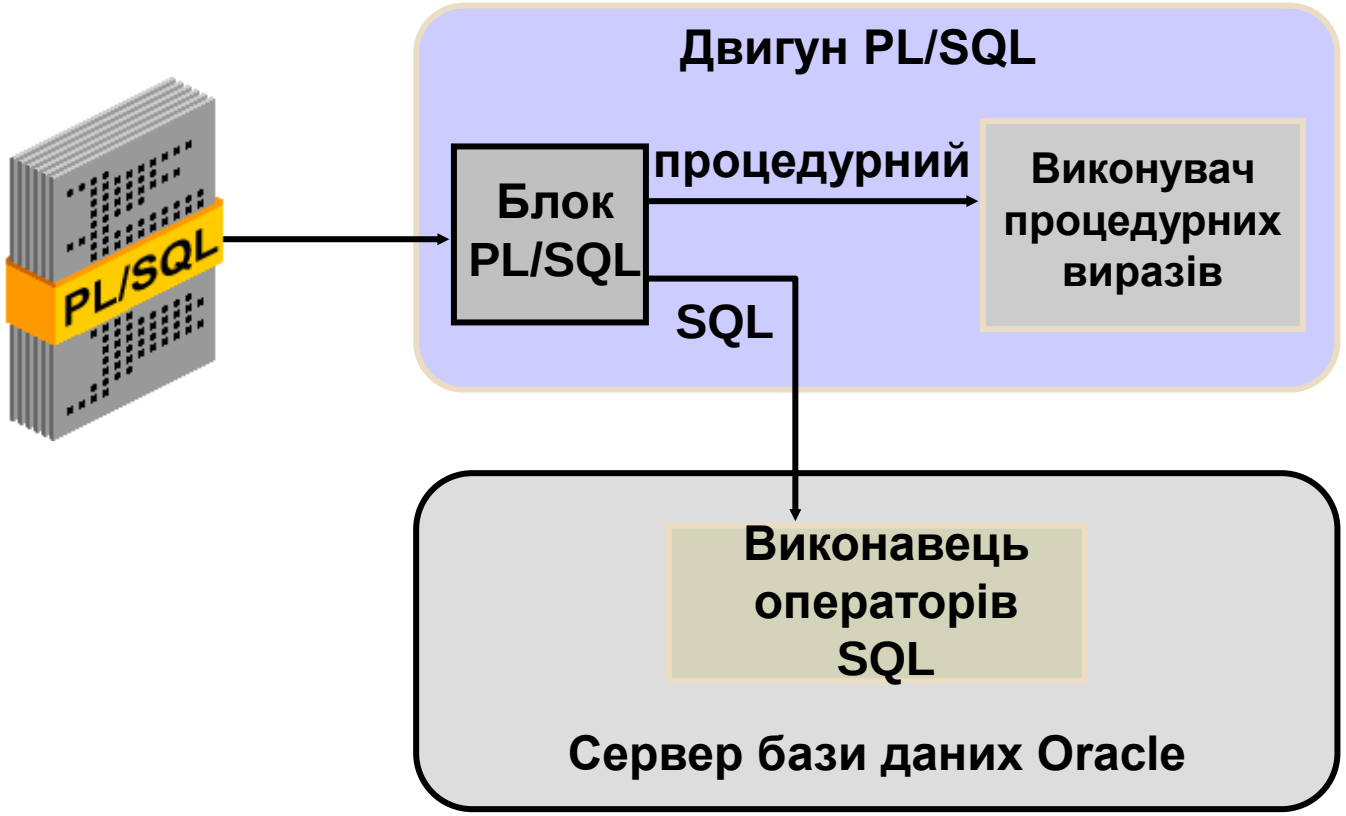
Про PL/SQL

- PL/SQL:
  - Розшифровується як «розширення процедурної мови до SQL»
  - Є стандартною мовою доступу до даних корпорації Oracle для реляційних баз даних
  - Безпроблемно інтегрує процедурні конструкції з SQL

Про PL/SQL

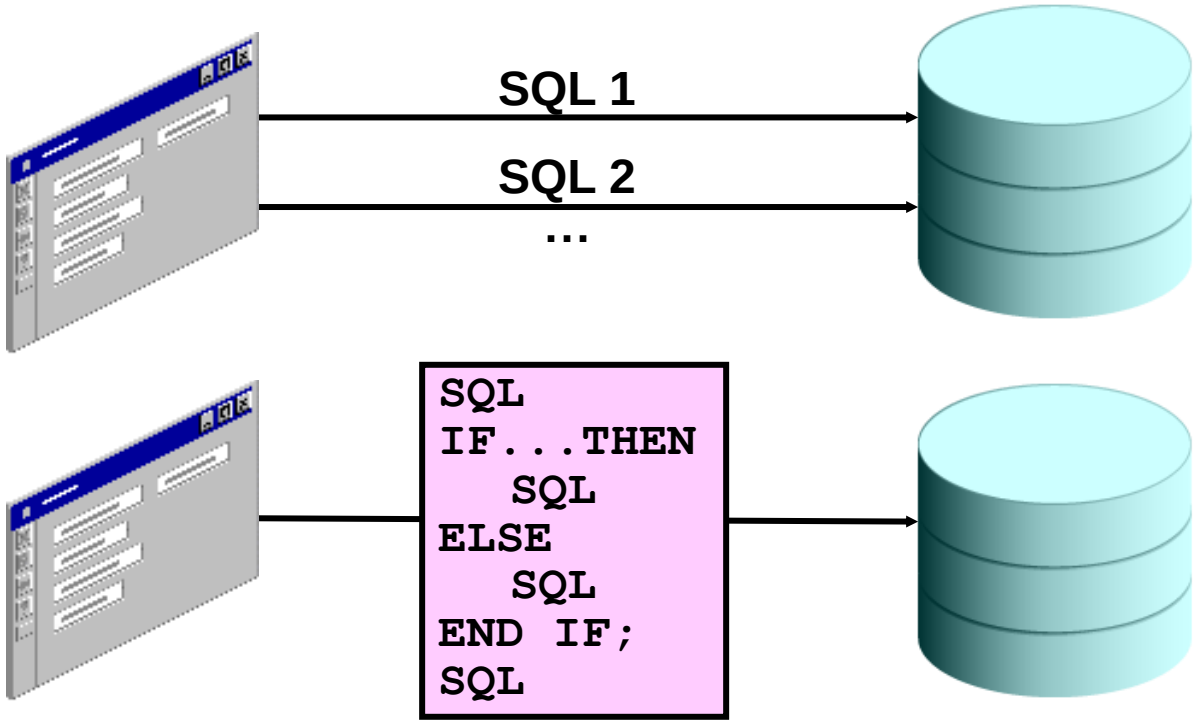
- PL/SQL:
  - Забезпечує блочну структуру для виконуваних одиниць коду. Обслуговування коду стає легшим завдяки такій чітко визначеній структурі.
  - Надає такі процедурні конструкції, як:
    - Змінні, константи та типи даних
    - Керуючі структури, такі як умовні оператори та цикли
    - Повторно використовувані програмні блоки, які пишуться один раз і виконуються багато разів

Середовище PL/SQL



Переваги PL/SQL

- Інтеграція процедурних конструкцій з SQL
- Покращена продуктивність

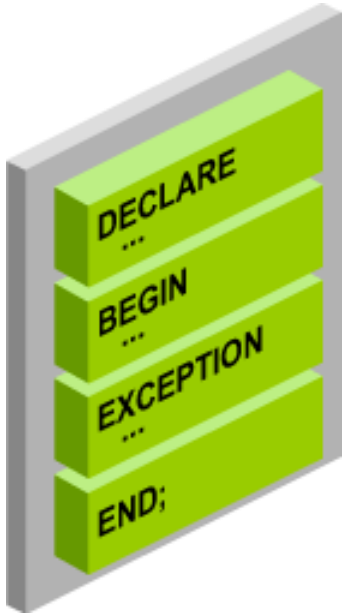


Переваги PL/SQL

- Розробка модульної програми
- Інтеграція з інструментами Oracle
- Портативність
- Обробка винятків

Блокова структура PL/SQL

- `DECLARE` (необов'язково)
  - Змінні, курсори, визначені користувачем винятки
- `BEGIN` (обов'язково)
  - Оператори SQL
  - Оператори PL/SQL
- `EXCEPTION` (необов'язково)
  - Дії, які потрібно виконати при виникненні помилок
- `END;` (обов'язковий)



Типи блоків

Анонімний

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

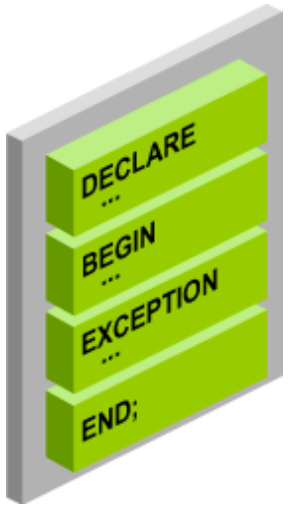
Процедура

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Функція

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

Програмні конструкції

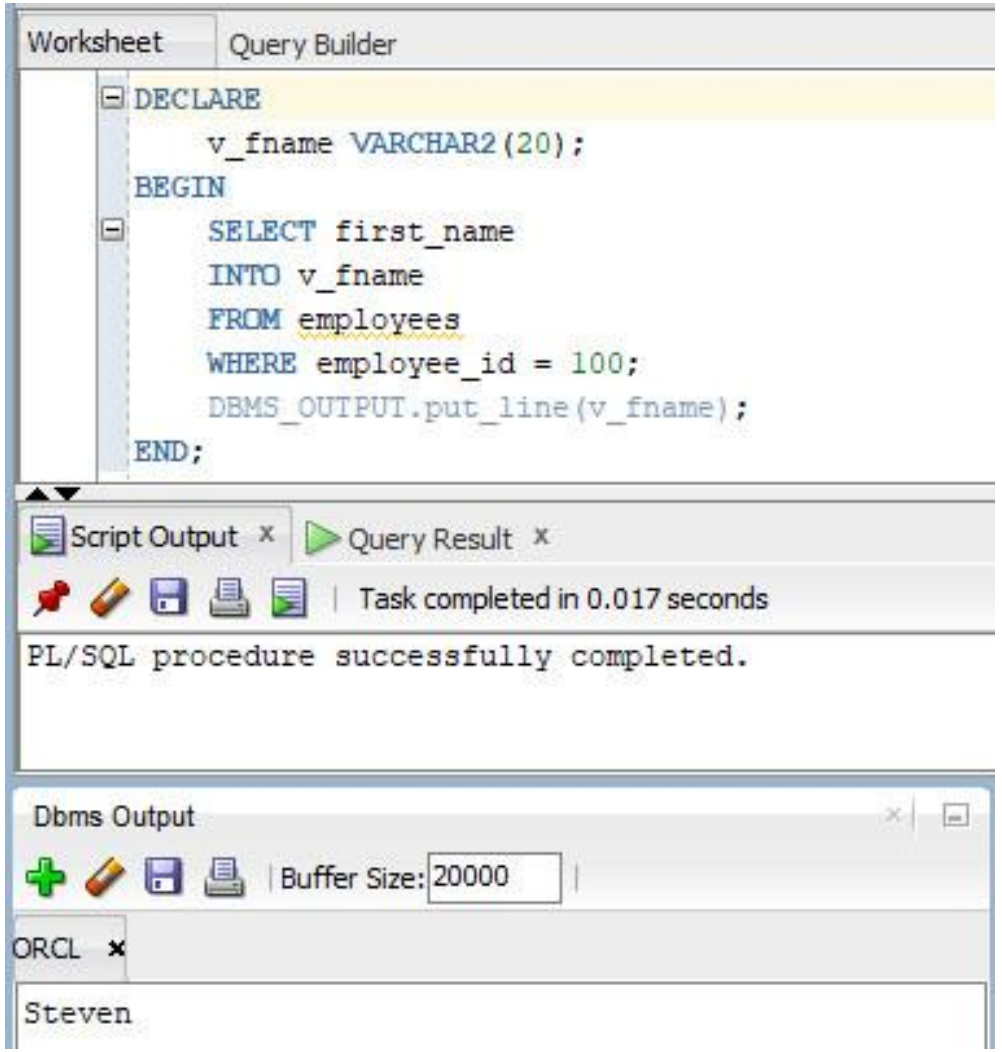


Інструменти Конструкції
Анонімні блоки
Процедури застосування або функції
Пакети застосунків
Тригери програми
Типи об'єктів

Конструкції серверу бази даних
Анонімні блоки
Збережені процедури або функції
Збережені пакети
Тригери бази даних
Типи об'єктів

Створіть анонімний блок

- Введіть анонімний блок у робочу область SQL Developer.
- Виконайте його.



Вікторина

- Блок PL/SQL *має* складатися з наступних трьох секцій:
  - Декларативний розділ, який починається ключовим словом `DECLARE` і закінчується, коли починається розділ виконуваного файлу.
  - Виконуваний розділ, який починається ключовим словом `BEGIN` і закінчується `END`.
  - Розділ обробки винятків, який починається ключовим словом `EXCEPTION` і вкладений у розділ виконуваного файлу.

1. правда
2. помилковий

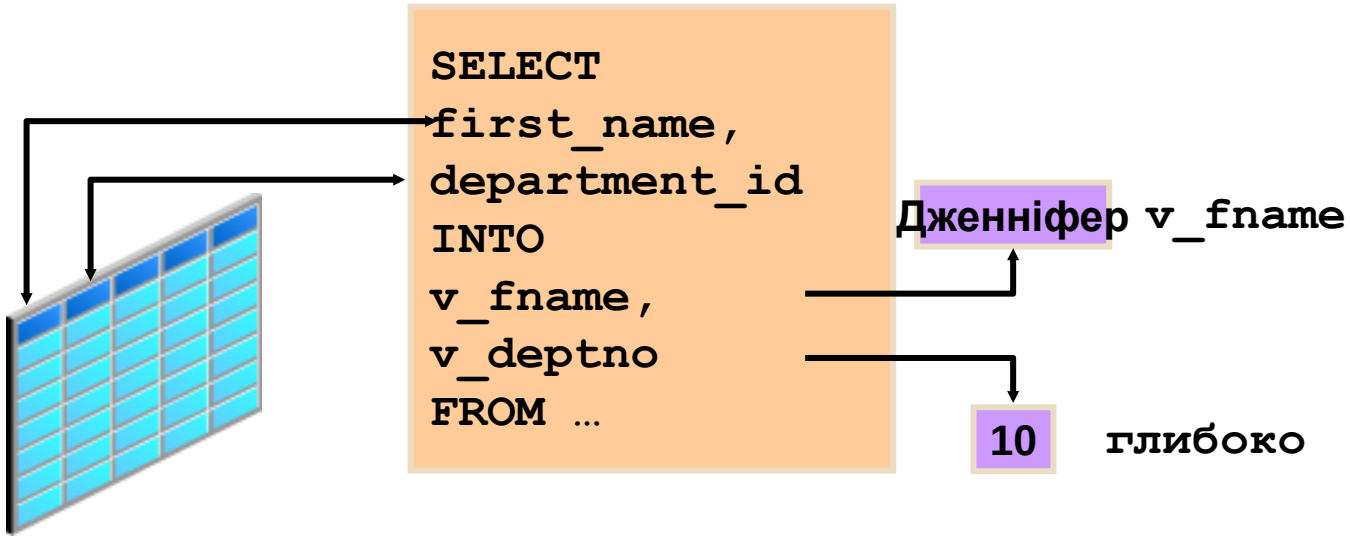
Висновок по розділу

- В цьому розділі Ви ознайомилися з:
  - Інтеграцією операторів SQL із конструкціями програм PL/SQL
  - Описом переваг PL/SQL
  - Типами блоків PL/SQL
  - Виведенням повідомлень на PL/SQL

Оголошення змінних PL/SQL

Використання змінних

- Змінні можна використовувати для:
  - Тимчасового зберігання даних
  - Маніпулювання збереженими значеннями
  - Багаторазового використання



Вимоги до імен змінних

- Ім'я змінної:
  - Має починатися з букви
  - Може містити літери або цифри
  - Може містити спеціальні символи (такі як `$` , `_` і `#`)
  - Має містити не більше 30 символів
  - Не має містити зарезервованих слів

Обробка змінних у PL/SQL

- Змінна:
  - Оголошується та ініціалізується в декларативному розділі
  - Використовується та змінюється у розділі виконуваних файлів
  - Передається як параметр до підпрограм PL/SQL
  - Використовується для зберігання виводу підпрограми PL/SQL

Оголошення та ініціалізація змінних PL/SQL

- Синтаксис:

```
ідентифікатор [CONSTANT] тип даних [NOT NULL]
[:= | DEFAULT expr ];
```

- Приклад:

```
DECLARE
  v_hiredate      DATE;
  v_deptno        NUMBER(2) NOT NULL := 10;
  v_location      VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

Оголошення та ініціалізація змінних PL/SQL

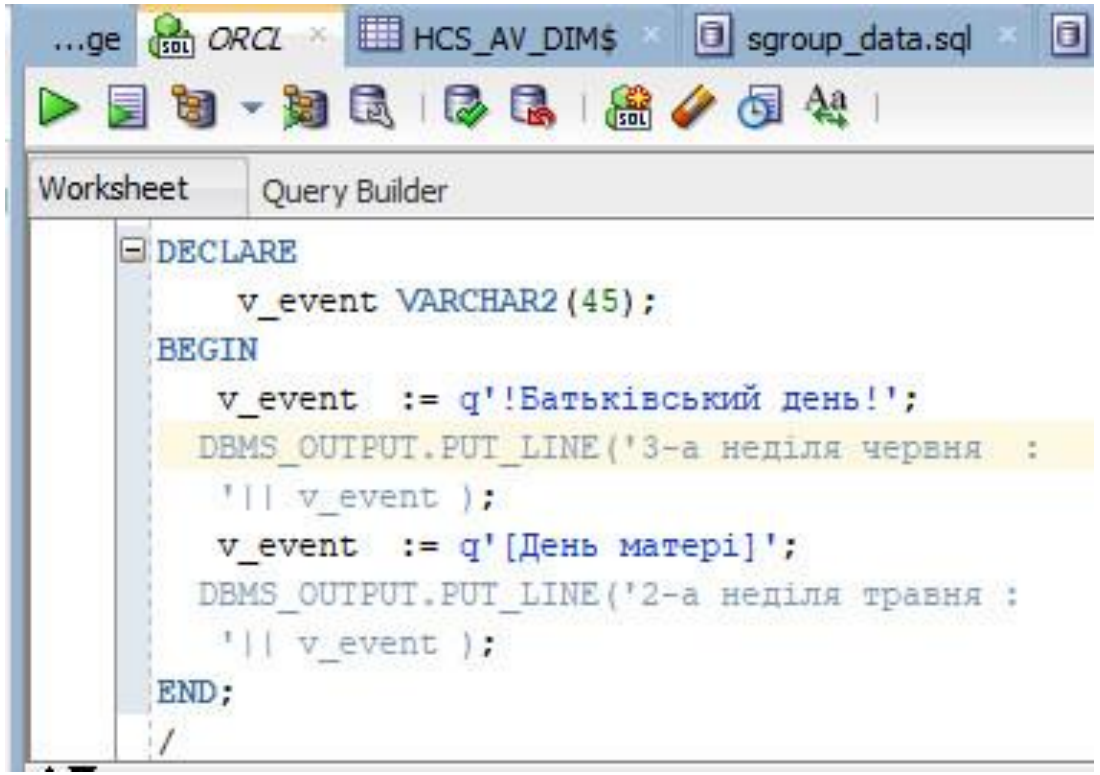
1

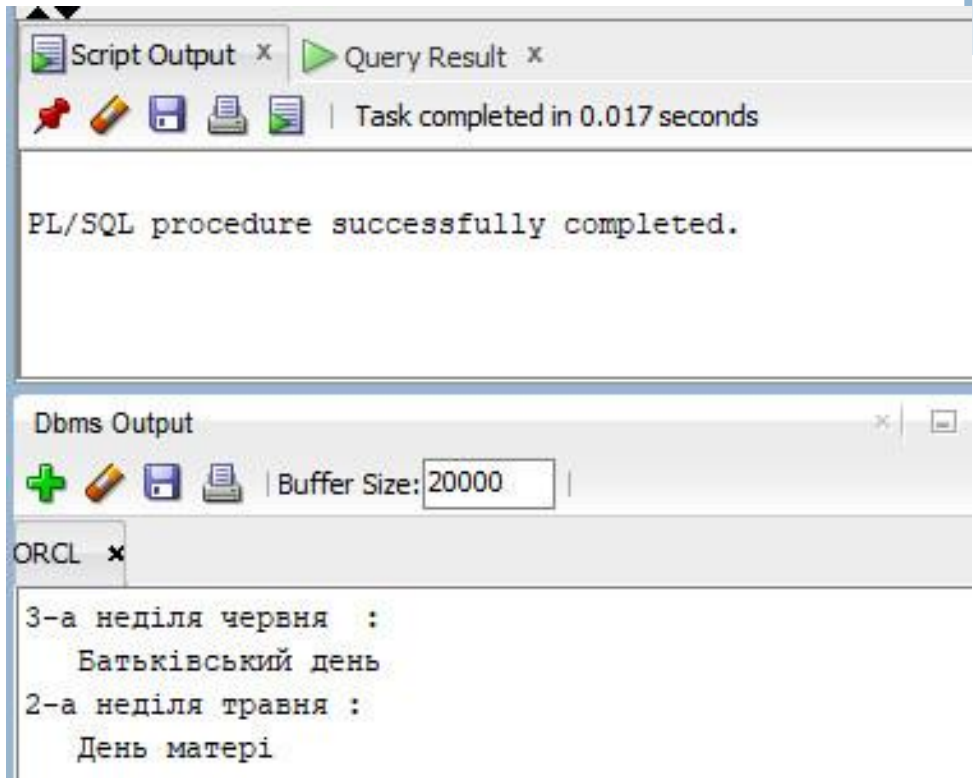
```
DECLARE
  v_myName VARCHAR2(20);
BEGIN
 DBMS_OUTPUT.PUT_LINE('Моє ім'я: '|| v_myName);
 v_myName := 'Ярослав';
 DBMS_OUTPUT.PUT_LINE('Моє ім'я: '|| v_myName);
END;
/
```

2

```
DECLARE
 v_myName VARCHAR2(20):= 'Ярослав';
BEGIN
 v_myName := 'Андрій';
 DBMS_OUTPUT.PUT_LINE('Моє ім'я: '|| v_myName);
END;
/
```

Роздільники в рядкових літералах





Типи змінних

- Типи змінних PL/SQL:
  - Скалярний
  - Композитний
  - Посилання
  - Великий об'єкт (LOB)
- Змінні, що не належать до PL/SQL: змінні прив’язки

Типи змінних

`True`

25-СІЧНЯ-01

Білосніжка

**Давним-давно
в далекій-далекій країні
жила принцеса на ім'я
Білосніжка. . .**

256120,08

Атланта

Рекомендації щодо оголошення та ініціалізації змінних PL/SQL

- Дотримуйтеся правил іменування.
- Використовуйте значущі ідентифікатори для змінних.
- Ініціалізувати змінні, позначені як `NOT NULL` і `CONSTANT`.
- Ініціалізуйте змінні оператором присвоювання (:=) або ключовим словом `DEFAULT`:

```
v_myName VARCHAR2(20):='Іван';
```

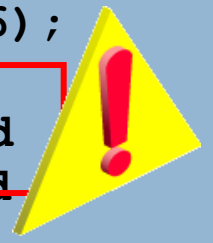
```
v_myName VARCHAR2(20) DEFAULT 'Джон';
```

- Оголошуйте один ідентифікатор на рядок для кращої читабельності та підтримки коду.

Вказівки щодо оголошення змінних PL/SQL

- Уникайте використання імен стовпців як ідентифікаторів.

```
DECLARE
  employee_id  NUMBER(6);
BEGIN
  SELECT     employee_id
  INTO       employee_id
  FROM       employees
  WHERE      last_name = 'Kochhar';
END;
/
```



- Використовуйте обмеження `NOT NULL`, коли змінна має містити значення.

Скалярні типи даних

- Зберігає одне значення
- Не має внутрішніх компонентів

`True`

25-СІЧНЯ-01

Душа лінивого бажає, та не має нічого; а душа старанного збагатиться.

256120,08

Атланта

Базові скалярні типи даних

- CHAR [(maximum_length)]
- VARCHAR2 (maximum_length)
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- BINARY_FLOAT
- BINARY_DOUBLE

Базові скалярні типи даних

- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

Оголошення скалярних змінних

- Приклади:

```
DECLARE
  v_emp_job          VARCHAR2(9);
  v_count_loop       BINARY_INTEGER := 0;
  v_dept_total_sal   NUMBER(9,2) := 0;
  v_orderdate        DATE := SYSDATE + 7;
  c_tax_rate         CONSTANT NUMBER(3,2) := 8.25;
  v_valid            BOOLEAN NOT NULL := TRUE;
  ...
```

Атрибут %TYPE

- Використовується для оголошення змінної відповідно до:
  - Визначення стовпця бази даних
  - Іншої оголошеної змінної
- Має префікс:
  - Імена таблиць і стовпців бази даних
  - Ім'я оголошеної змінної

Оголошення змінних з атрибутом %TYPE

- Синтаксис

```
ідентифікатор table.column_name%TYPE;
```

- Приклади

```
...
emp_lname members.last_name%TYPE;
...
```

```
...
  balance          NUMBER(7,2);
  min_balance      balance%TYPE := 1000;
...
```

Оголошення булевих змінних

- Булевій змінній можна присвоїти лише значення `TRUE`, `FALSE` та `NULL`.
- Умовні вирази використовують логічні оператори `AND` та `OR` та унарний оператор `NOT` для перевірки значень змінних.
- Змінні завжди дають значення `TRUE`, `FALSE` або `NULL`.
- Для повернення логічного значення можна використовувати арифметичні, символьні вирази та вирази дати.

Змінні прив'язки

- Змінні прив’язки:
  - Створюються в середовищі
  - Також називаються змінними *хоста*
  - Створюються за допомогою ключового слова `VARIABLE`
  - Використовуються в операторах SQL і блоках PL/SQL
  - Доступ до них навіть після виконання блоку PL/SQL
  - Посилаються на них з попередньою двокрапкою

Друк змінних прив'язки

- Приклад:

Worksheet | Query Builder

```
VARIABLE b_emp_salary NUMBER
BEGIN
    SELECT salary INTO :b_emp_salary
    FROM  employees WHERE employee_id = 178;
END;
/
PRINT b_emp_salary
SELECT first_name, last_name FROM employees
WHERE salary = :b_emp_salary;
```

Script Output | Query Result

Task completed in 0.133 seconds

```
Oliver          Tuvault
Sarath          Sewall
Kimberely       Grant
```

Друк змінних прив'язки

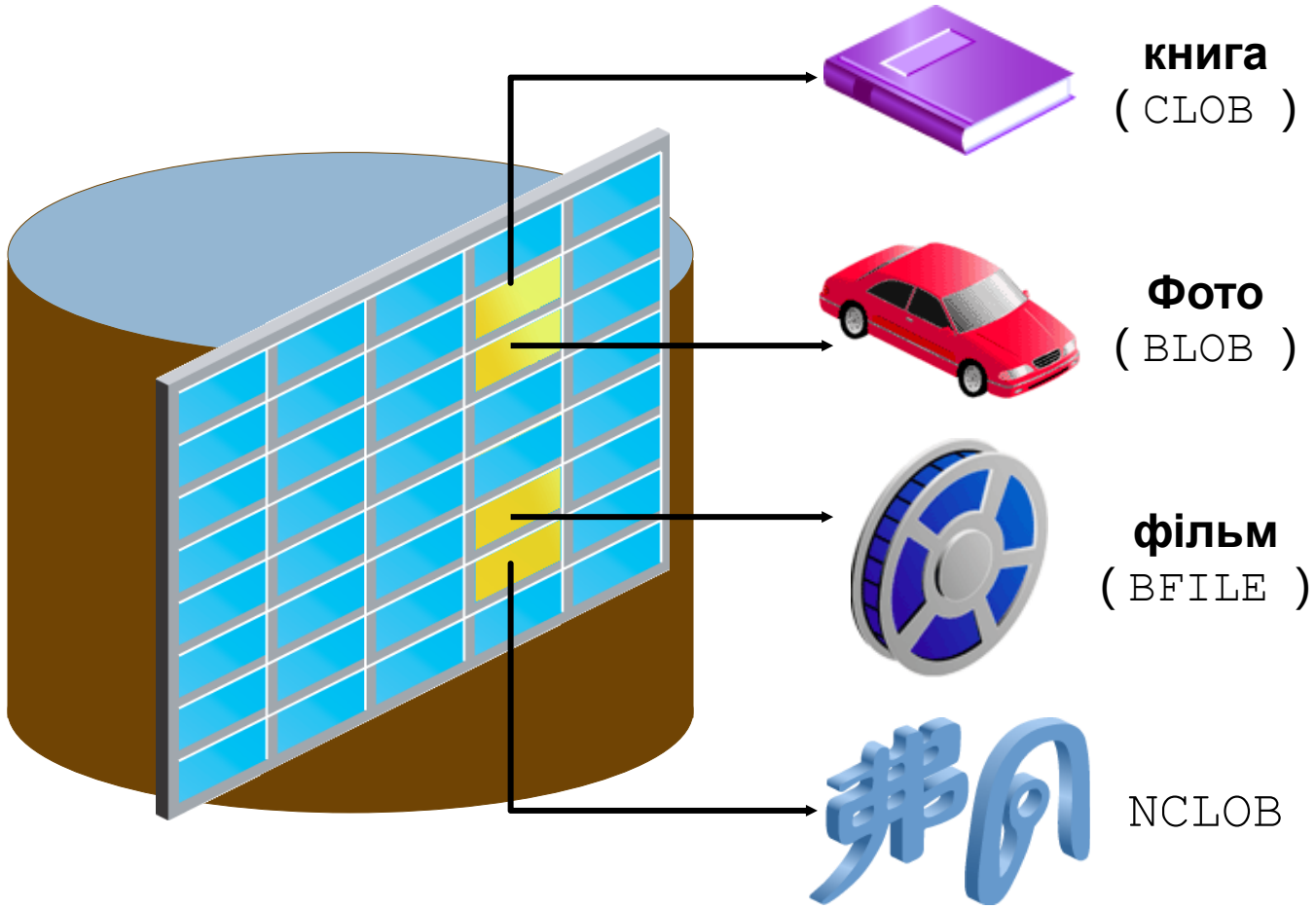
- Приклад:

```
VARIABLE b_emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
  v_empno NUMBER(6):=&empno;
BEGIN
   SELECT salary  INTO :b_emp_salary
   FROM  employees WHERE employee_id = v_empno;
END;
```

Вихід:

```
7000
```

Змінні типу даних LOB



Складені типи даних

TRUE	23-DEC-23	ATLANTA	

PL/SQL table structure

1	SMITH
2	JONES
3	NANCY
4	TIM

VARCHAR2

PLS_INTEGER

PL/SQL table structure

1	5000
2	2345
3	12
4	3456

NUMBER

PLS_INTEGER

Вікторина

Атрибут `%TYPE`:

1. Використовується для оголошення змінної відповідно до визначення стовпця бази даних.
2. Використовується для оголошення змінної відповідно до набору стовпців у таблиці бази даних або поданні.
3. Використовується для оголошення змінної відповідно до визначення іншої оголошеної змінної.
4. Має префікс таблиці бази даних і імен стовпців або ім'я оголошеної змінної.

Висновки за розділом

- В цьому розділі Ви мали навчитися:
  - Розпізнавати дійсні та недійсні ідентифікатори.
  - Оголошувати змінні в декларативному розділі блоку PL/SQL.
  - Ініціалізувати змінні та використовувати їх у розділі виконуваних файлів.
  - Розрізняти скалярні та складені типи даних.
  - Використовувати атрибут `%TYPE`.
  - Використовувати змінні прив'язки.

Написання виконуваних операторів

Лексичні одиниці в блоці PL/SQL

Лексичні одиниці:

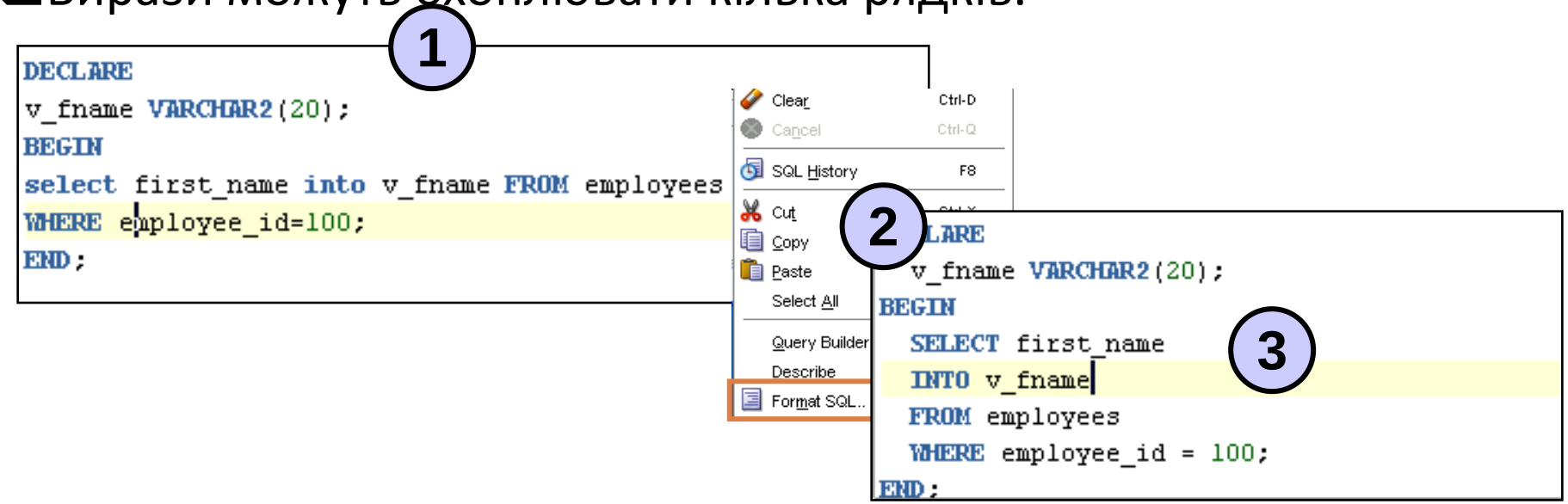
- Є будівельними блоками будь-якого блоку PL/SQL.
- Це послідовності символів, включаючи літери, цифри, знаки табуляції, пробіли, повернення та символи.
- Можна класифікувати як:
  - Ідентифікатори: `v_fname, c_percent`
  - Роздільники: `; , +, -`
  - Літерали: `Іван, 428, Правда`
  - Коментарі: `--, /* */`

Синтаксис блоку PL/SQL і вказівки

- Літерали
  - Літерали символів і дат повинні бути взяті в одинарні лапки.
  - Числа можуть бути простими значеннями або в науковій нотації.

```
name := 'Henderson';
```

- Вирази можуть охоплювати кілька рядків.



Код коментування

- До однорядкових коментарів додайте два дефіси (--).
- Розмістіть багаторядкові коментарі між символами /* і */.

Приклад:

```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Розрахувати річну зарплату на основі
введення місячної зарплати від користувача */
v_annual_sal := monthly_sal * 12;
--У наступному рядку відображається річна зарплата
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
КІНЕЦЬ;
/
```

Функції SQL у PL/SQL

- Доступні в процедурних виразах:
  - Однорядкові функції
- Недоступні в процедурних виразах:
  - `DECODE`
  - Групові функції

Функції SQL у PL/SQL: приклади

- Отримайте довжину рядка:

```
v_desc_size INTEGER(5);
v_prod_description VARCHAR2(70):='Ви можете
використовувати цей продукт з вашими радіоприймачами для
високої частоти';

-- отримати довжину рядка в prod_description
v_desc_size:= LENGTH(v_prod_description);
```

- Отримайте кількість місяців роботи працівника:

```
v_tenure:= MONTHS_BETWEEN (CURRENT_DATE, v_hiredate);
```

Використання послідовностей у виразах PL/SQL

- Починаючи з 11g:

```
DECLARE
  v_new_id NUMBER;
BEGIN
  v_new_id := my_seq.NEXTVAL;
END;
/
```

- До 11g:

```
DECLARE
  v_new_id NUMBER;
BEGIN
  SELECT my_seq.NEXTVAL INTO v_new_id FROM Dual;
END;
/
```

Перетворення типу даних

- Перетворює дані на порівнювані типи даних
- Буває двох видів:
  - Неявне перетворення
  - Явне перетворення
- Функції:
  - `TO_CHAR`
  - `TO_DATE`
  - `TO_NUMBER`
  - `TO_TIMESTAMP`

Перетворення типу даних

1
```
date_of_joining DATE:= '02-Feb-2023';
```

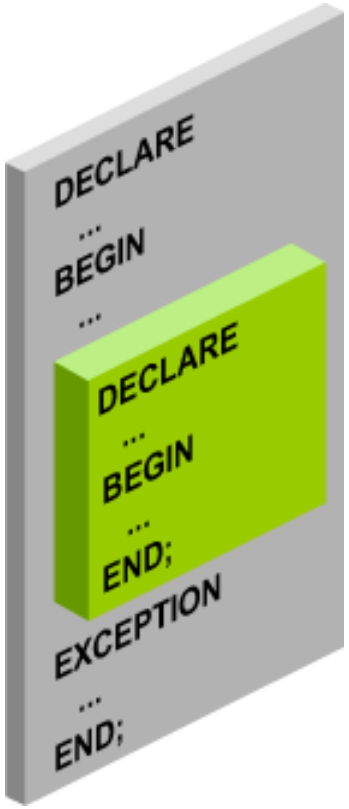
2
```
date_of_joining DATE:= 'February 02,2023';
```

3
```
date_of_joining DATE:= TO_DATE('February
02,2023','Month DD, YYYY');
```

Вкладені блоки

Блоки PL/SQL можуть бути вкладеними.

- Виконуваний розділ BEGIN … END) може містити вкладені блоки.
- Розділ винятків може містити вкладені блоки.



Вкладені блоки: приклад

Worksheet | Query Builder

```
DECLARE
 v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
   v_inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
   DBMS_OUTPUT.PUT_LINE(v_inner_variable);
   DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
 DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

Dbms Output

Buffer Size

ORCL ×

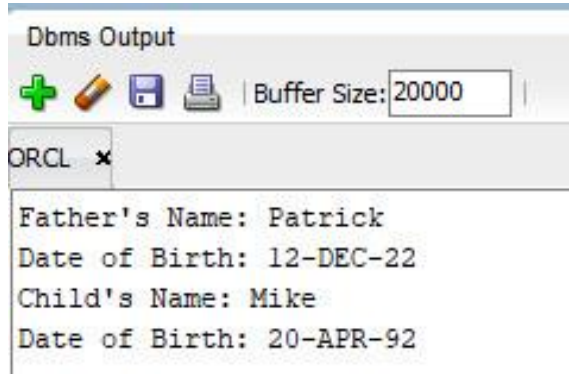
```
LOCAL VARIABLE
GLOBAL VARIABLE
GLOBAL VARIABLE
```

Область змінної та видимість

```
DECLARE
 v_father_name VARCHAR2(20):='Patrick';
 v_date_of_birth DATE:='20-Apr-1992';
BEGIN
  DECLARE
   v_child_name VARCHAR2(20):='Mike';
   v_date_of_birth DATE:='12-Dec-2022';
  BEGIN
   DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
   DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
  END;
 DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
/
```

1

2



Кваліфікуйте ідентифікатор

0.024 seconds

Worksheet | Query Builder

```
BEGIN <<outer>>
DECLARE
 v_father_name VARCHAR2(20):='Patrick';
 v_date_of_birth DATE:='20-Apr-1992';
BEGIN
  DECLARE
   v_child_name VARCHAR2(20):='Mike';
   v_date_of_birth DATE:='12-Dec-2022';
  BEGIN
   DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: '
                        ||outer.v_date_of_birth);
   DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
  END;
END;
END outer;
```

Dbms Output

Buffer Size: 20000

ORCL

```
Father's Name: Patrick
Date of Birth: 20-APR-92
Child's Name: Mike
Date of Birth: 12-DEC-22
```

Визначення області видимості змінної: приклад

```
BEGIN <<outer>>
DECLARE
  v_sal       NUMBER(7,2) := 60000;
  v_comm      NUMBER(7,2) := v_sal * 0.20;
  v_message   VARCHAR2(255) := ' eligible for commission';
BEGIN
  DECLARE
        v_sal       NUMBER(7,2) := 50000;
        v_comm      NUMBER(7,2) := 0;
        v_total_comp  NUMBER(7,2) := v_sal + v_comm;
  BEGIN
         v_message := 'CLERK not'||v_message;
        outer.v_comm := v_sal * 0.30;
  END;
 v_message := 'SALESMAN'||v_message;
END;
END outer;
/
```

1 → `outer.v_comm := v_sal * 0.30;`

2 → `END;`

Оператори в PL/SQL: приклади

- Збільшити лічильник для циклу.

```
loop count := loop_count + 1;
```

- Встановити значення логічного прапора.

```
good_sal := sal BETWEEN 50000 AND 150000;
```

- Перевірити, чи номер працівника містить значення.

```
valid  := (empno IS NOT NULL);
```

Інструкції з програмування

Спростіть обслуговування коду:

- Документування коду з коментарями
- Розробка угоди про регістр для коду
- Розробка домовленостей про іменування ідентифікаторів та інших об’єктів
- Покращення читабельності шляхом відступу

Відступи

Для ясності зробіть відступи для кожного рівня коду.

```
BEGIN
  IF x=0 THEN
     y:=1;
  END IF;
END;
/
```

```
DECLARE
  deptno        NUMBER(4);
  location_id   NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    deptno,
          location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';
...
END;
/
```

Вікторина

У виразах PL/SQL можна використовувати більшість однорядкових функцій SQL, таких як однорядкові функції для чисел, символів, перетворень та роботи з датами.

1. True
2. False

Резюме

В цьому розділі ви мали навчитися:

- Визначати лексичні одиниці в блоці PL/SQL
- Використовувати вбудовані функції SQL у PL/SQL
- Писати вкладені блоки, щоб зламати логічно пов’язану функціональність
- Вирішувати, коли виконувати явні перетворення
- Кваліфікувати змінні у вкладених блоках
- Використовувати послідовності у виразах PL/SQL

Взаємодія з сервером Oracle

Команди SQL у PL/SQL

- Вибір рядка даних із бази даних за допомогою команди `SELECT`. Повинна бути повернений лише один рядок.
- Внесення змін до рядків таблиць бази даних за допомогою команд DML.
- Управління транзакціями за допомогою команд `COMMIT`, `ROLLBACK` та `SAVEPOINT`.

Команди SELECT у PL/SQL

- Вибір даних з бази даних здійснюється за допомогою команди SELECT
- Синтаксис:

```
SELECT список_вибірки
INTO { ім'я_змінної [, ім'я_змінної ] ...
| ім'я_запису }
FROM таблиця
[WHERE умова ];
```

Команди SELECT у PL/SQL

- Пропозиція INTO є обов'язковою.
- Запити повинні повертати один і лише один рядок.
- Приклад:

Worksheet | Query Builder

```
SET SERVEROUTPUT ON

DECLARE
    fname VARCHAR2(25);
BEGIN
    SELECT
        first_name
    INTO fname
    FROM
        employees
    WHERE
        employee_id = 200;

    dbms_output.put_line(' First Name is : ' || fname);
END;
/
```

Dbms Output

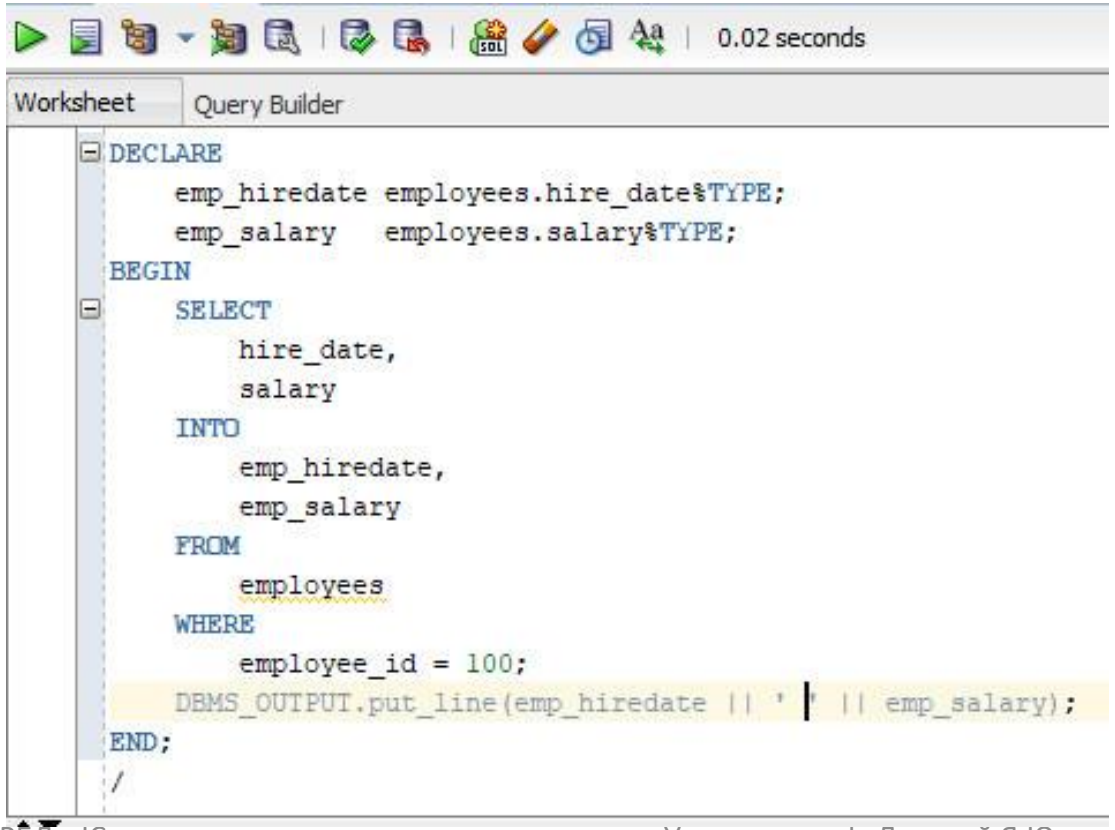
Buffer Size: 20000

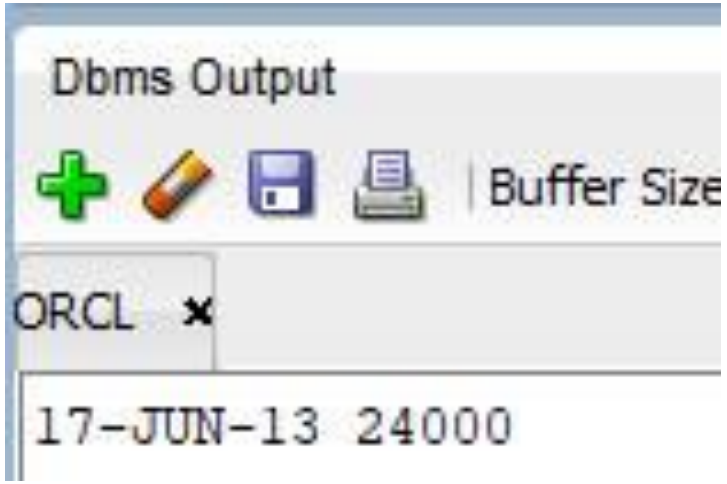
ORCL ×

```
First Name is : Jennifer
```

Вибірка даних у PL/SQL

- Вибірка дати найму та окладу для певного службовця.
- Приклад:





Вибірка даних у PL/SQL

- Знайти суму окладів всіх службовців для заданого відділу.
- Приклад:

Worksheet | Query Builder

```
SET SERVEROUTPUT ON
DECLARE
    sum_sal NUMBER(10, 2);
    deptno  NUMBER NOT NULL := 60;
BEGIN
    SELECT SUM(salary) -- group function
    INTO sum_sal FROM employees
    WHERE department_id = deptno;
    DBMS_OUTPUT.PUT_LINE ('sum of salary is ' || sum_sal );
END;
/
```

Dbms Output

Buffer Size: 20000

ORCL

```
sum of salary is 28800
```

Правила присвоєння імен

Worksheet | Query Builder

```
DECLARE
    hire_date   employees.hire_date%TYPE;
    sysdate     hire_date%TYPE;
    employee_id employees.employee_id%TYPE := 176;
BEGIN
    SELECT
        hire_date,
        sysdate
    INTO
        hire_date,
        sysdate
    FROM
        employees
    WHERE
        employee_id = employee_id;

END;
/
```

Script Output | Query Result

Task completed in 0.028 seconds

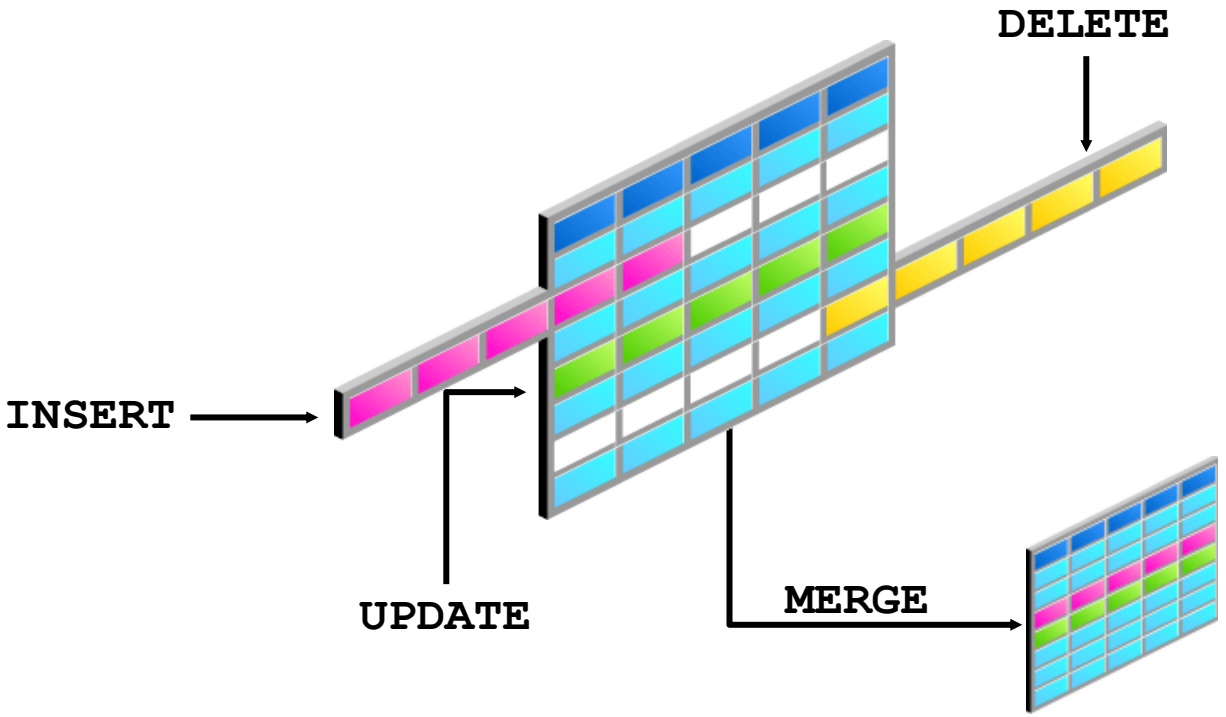
```
        employee_id = employee_id;

END;
Error report -
ORA-01422: exact fetch returned more than the requested number of rows
ORA-06512: at line 6
01422. 00000 -  "exact fetch returns more than requested number of rows"
*Cause:    The number specified in exact fetch is less than the rows returned.
*Action:   Rewrite the query or change number of rows requested
```

Правила присвоєння імен

- Щоб уникнути неоднозначності пропозиції `WHERE`, дотримуйтесь виробленої системи присвоєння імен.
- Уникайте використання імен стовпців бази як ідентифікаторів.
- Синтаксичні помилки можуть виникнути через те, що PL/SQL спочатку шукає стовпець з цим ім'ям у таблиці.
- Імена локальних змінних та формальних параметрів мають пріоритет над іменами *таблиць* бази даних.
- Імена *стовпців* мають пріоритет над іменами локальних змінних.

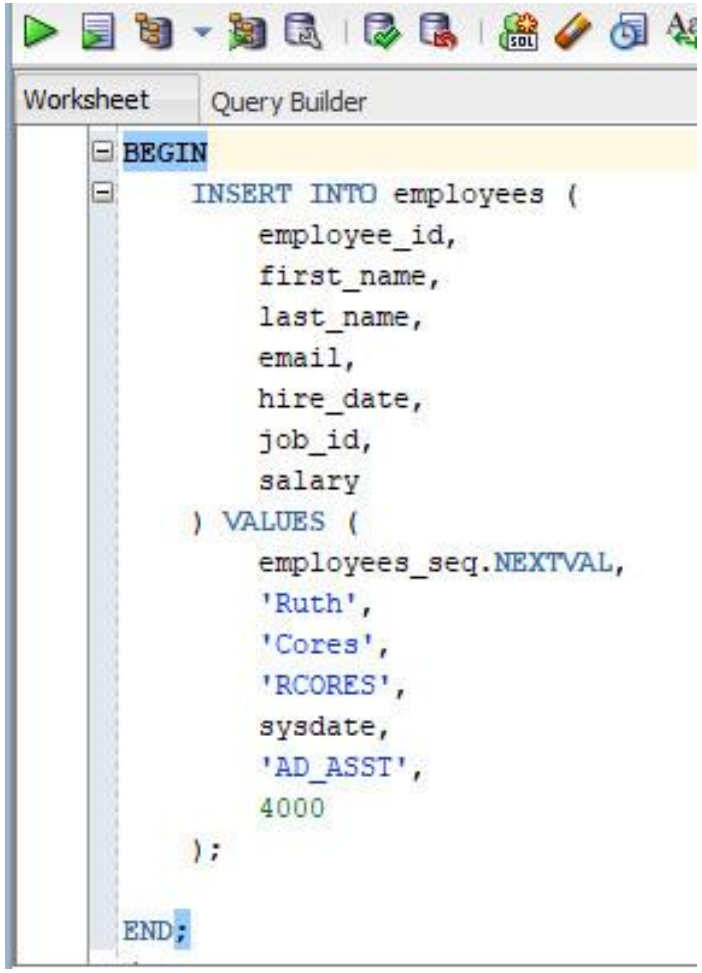
Маніпулювання даними в PL/SQL

- Внесення змін до таблиць бази даних за допомогою команд DML :
  - INSERT
  - UPDATE
  - DELETE
  - MERGE



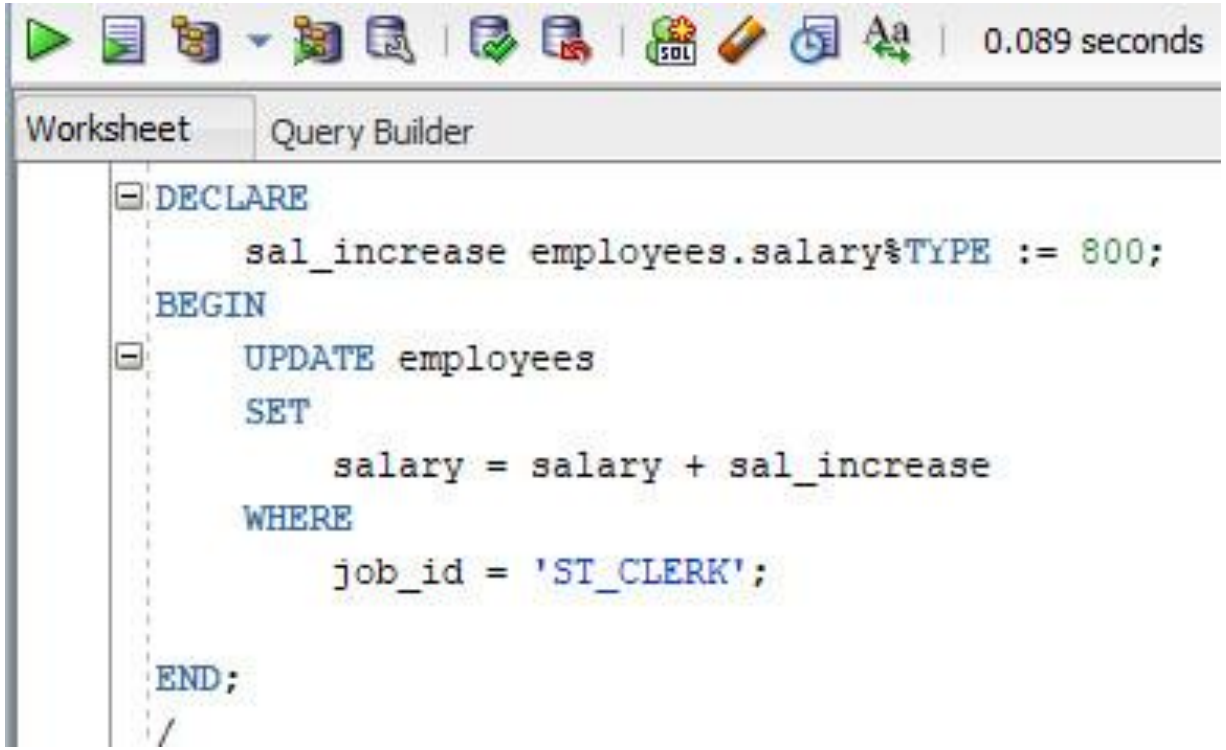
Вставка даних

- Приклад додавання інформації про нового службовця до таблиці `EMPLOYEES`:



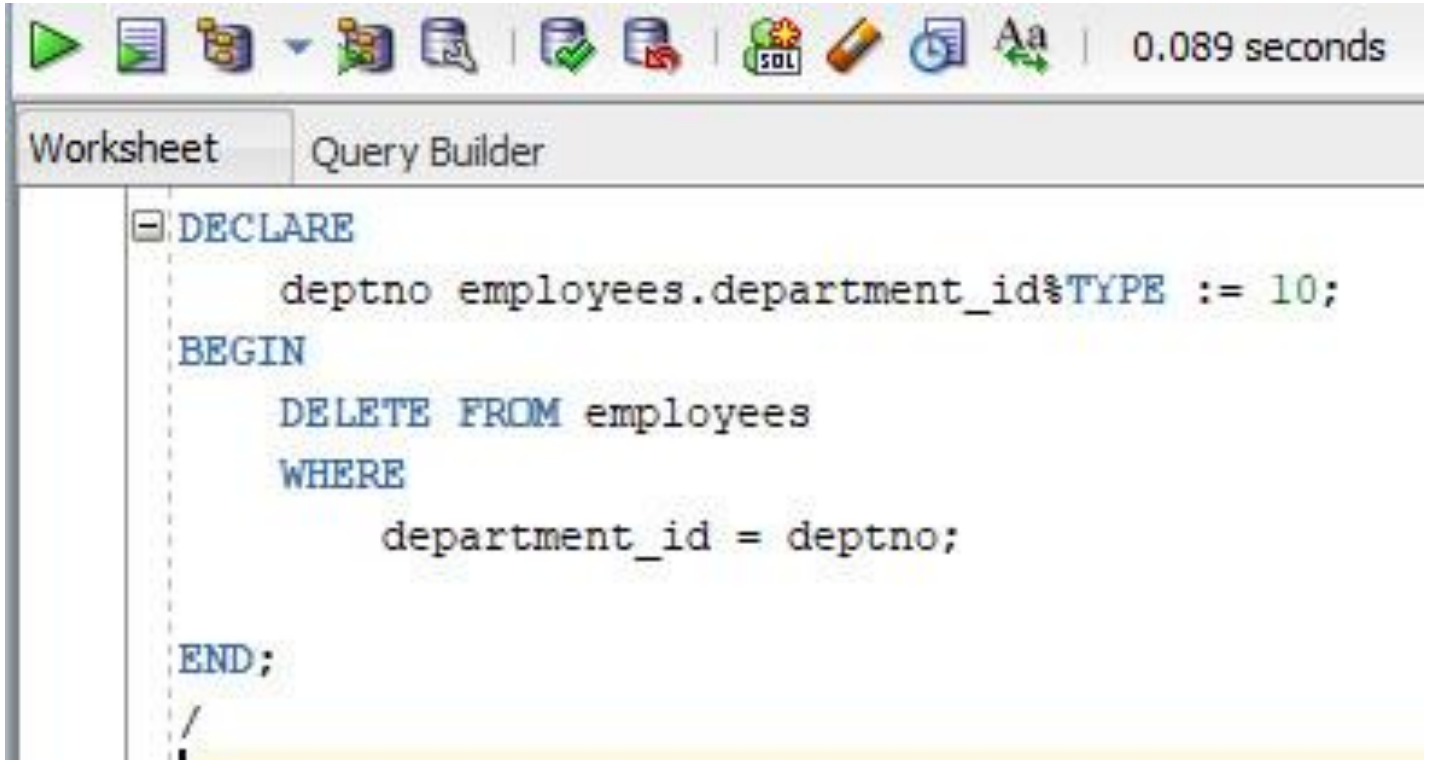
Оновлення даних

- Приклад збільшення окладу всіх співробітників таблиці EMPLOYEES, посада яких – клерк на складі (ST_CLERK):



Видалення даних

- Приклад видалення інформації про службовців 10 відділу таблиці EMPLOYEES:



Злиття рядків

- Вставлення або зміна рядків у таблиці COPY_EMP для встановлення відповідності до таблиці EMPLOYEES.

```
DECLARE
      empno employees.employee_id%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
     USING employees e
     ON (e.employee_id = empno)
   WHEN MATCHED THEN
     UPDATE SET
       c.first_name     = e.first_name,
       c.last_name      = e.last_name,
       c.email          = e.email,
       . . .
   WHEN NOT MATCHED THEN
     INSERT VALUES(e.employee_id, e.first_name, e.last_name,
           . . .,e.department_id);
END;
/
```

Курсор SQL

- Курсор – це покажчик на приватну область пам'яті, виділену сервером Oracle.
- Є два типи курсорів:
  - Неявні курсори: створюються та внутрішньо супроводжуються сервером Oracle для обробки команд SQL
  - Явні курсори: явно оголошуються програмістом

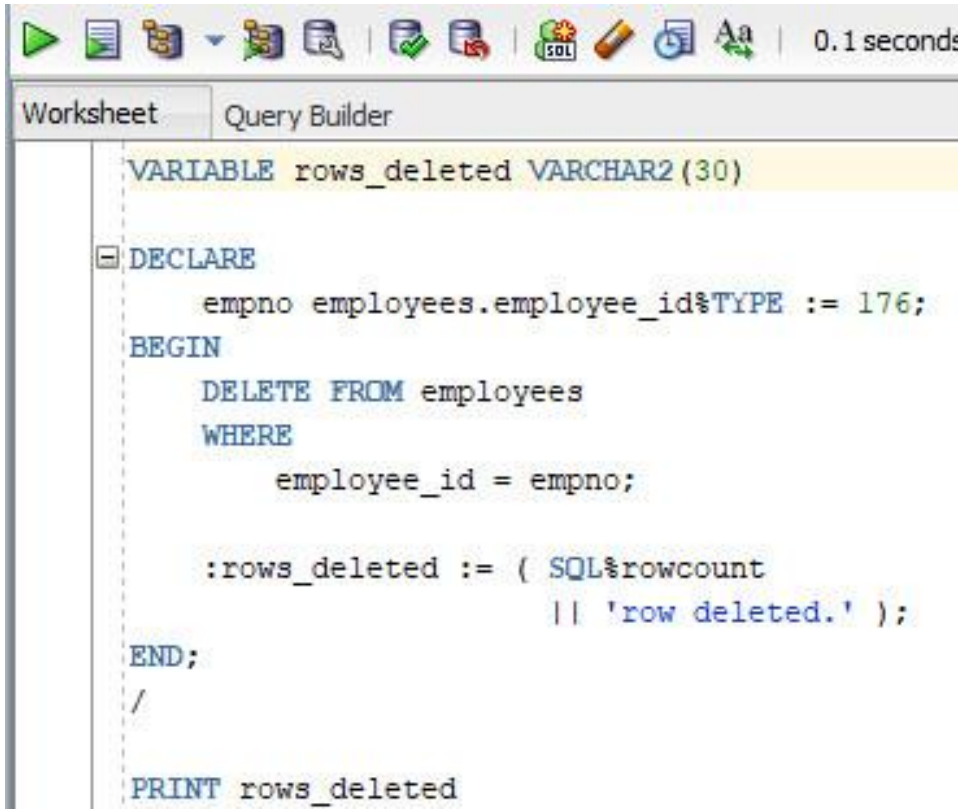
Атрибути курсору SQL, які використовуються неявними курсорами$_1$

- Атрибути курсору SQL дозволяють користувачеві перевірити результат виконання своєї команди SQL.

SQL%FOUND	Логічний атрибут, що повертає значення "істина" (TRUE), якщо остання команда SQL обробила один або кілька рядків.
SQL%NOTFOUND	Логічний атрибут, що повертає значення "істина" (TRUE), якщо остання команда SQL не обробила жодного рядка.
SQL%ROWCOUNT	Кількість рядків, оброблених останньою командою SQL (ціле число).

Атрибути курсору SQL, які використовуються неявними курсорами$_2$

- Приклад видалення рядків із конкретним номером службовця з таблиці `EMPLOYEES` та виведення кількості віддалених рядків.



Висновки за розділом

- Використання в блоці PL/SQL команд DML.
- Використання в командах `SELECT` пропозиції `INTO`, що обов'язково вказується в PL/SQL.
- Відмінності між явними та неявними курсорами.
- Використання атрибутів курсору SQL для визначення результатів виконання команди SQL.

Донецький національний технічний університет

Тема 9. Мова програмування PL/SQL. Лекція 14. PL/SQL. Управляючі структури. Складені типи.

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

- Запис управляючих структур
- Робота з складеними типами даних

Запис управляючих структур

Управління процесом виконання

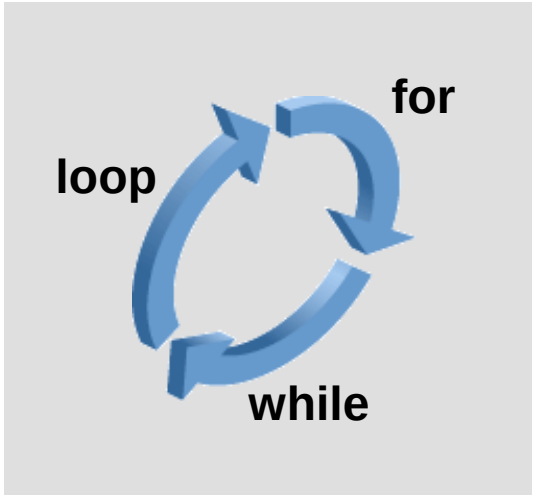
```
IF...
  THEN...
END IF;
```

```
IF...
  THEN...
  ELSE...
END IF;
```

```
IF...
  THEN...
  ELSIF...
  THEN...
END IF;
```

```
IF...
  THEN...
  ELSIF...
  THEN...
  ELSE...
END IF;
```

```
CASE
  WHEN... THEN..
  WHEN... THEN..
  WHEN... THEN..
  ELSE
END CASE;
```



Оператор IF

- Приклад:

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

Простий оператор IF

- Приклад:

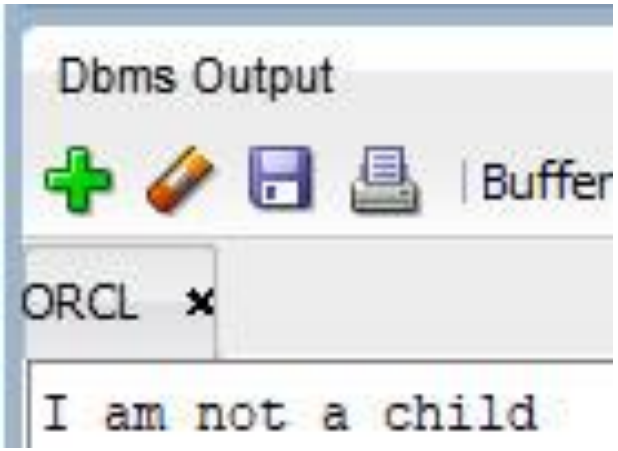
Worksheet | Query Builder

```
DECLARE
  v_myage  number:=31;
BEGIN
  IF v_myage  < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  END IF;
END;
```

Оператори IF THEN ELSE

- Приклад:

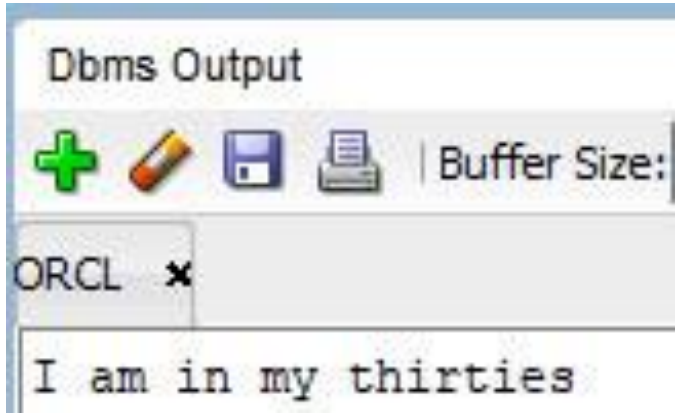
```
Worksheet  Query Builder
DECLARE
v_myage  number:=31;
BEGIN
IF v_myage  < 11
 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
 ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
END IF;
END;
/
```



IF ELSIF ELSE

Worksheet | Query Builder

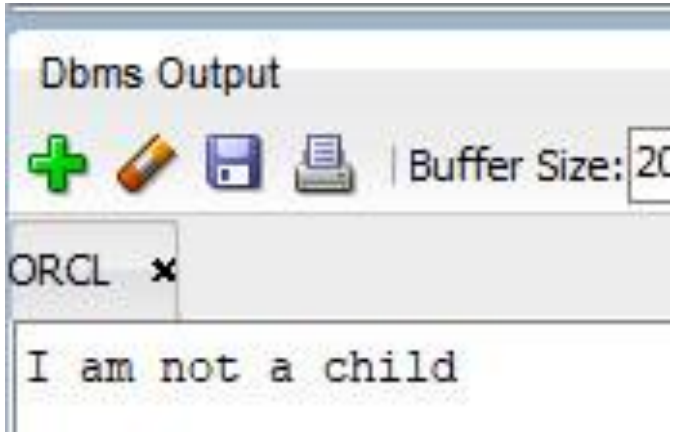
```
DECLARE
  v_myage number:=31;
BEGIN
  IF v_myage  < 11 THEN
       DBMS_OUTPUT.PUT_LINE(' I am a child ');
    ELSIF v_myage  < 20 THEN
       DBMS_OUTPUT.PUT_LINE(' I am young ');
    ELSIF v_myage  < 30 THEN
       DBMS_OUTPUT.PUT_LINE(' I am in my twenties');
    ELSIF v_myage  < 40 THEN
       DBMS_OUTPUT.PUT_LINE(' I am in my thirties');
    ELSE
       DBMS_OUTPUT.PUT_LINE(' I am always young ');
  END IF;
END;
```



NULL значення в операторі IF

Worksheet | Query Builder

```
DECLARE
  v_myage  number;
BEGIN
  IF v_myage  < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
/
```



Вираз CASE

- Вираз CASE вибирає результат і повертає його.
- Для вибору результатів CASE використовує вирази. Значення, повернене за допомогою цих виразів використовується для вибору однієї з декількох варіантів.

```
CASE selector
   WHEN expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN
  [ELSE resultN+1]
END;
/
```

Вираз CASE: Приклад

Worksheet | Query Builder

```
SET VERIFY OFF
DECLARE
   v_grade  CHAR(1) := UPPER('&grade');
   appraisal VARCHAR2(20);
BEGIN
   appraisal := CASE v_grade
         WHEN 'A' THEN 'Excellent'
         WHEN 'B' THEN 'Very Good'
         WHEN 'C' THEN 'Good'
         ELSE 'No such grade'
      END;
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade  || ' Appraisal ' || appraisal);
END;
/
```

Dbms Output

Buffer Size: 20000

ORCL ×

```
Grade: A Appraisal Excellent
```

Пошук виразів CASE

Worksheet | Query Builder

```
DECLARE
   v_grade  CHAR(1) := UPPER('&grade');
   appraisal VARCHAR2(20);
BEGIN
    appraisal := CASE
         WHEN v_grade  = 'A' THEN 'Excellent'
         WHEN v_grade  IN ('B','C') THEN 'Good'
         ELSE 'No such grade'
     END;
   DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade  || ' Appraisal ' || appraisal);
END;
/
```

Dbms Output

Buffer Size: 20000

ORCL ×

```
Grade: B Appraisal Good
```

Оператор CASE


```
DECLARE
   v_deptid NUMBER;
   v_deptname VARCHAR2(20);
   v_emps NUMBER;
   v_mngid NUMBER:= 108;
BEGIN
  CASE  v_mngid
   WHEN  108 THEN
    SELECT department_id, department_name
     INTO v_deptid, v_deptname FROM departments
     WHERE manager_id=108;
    SELECT count(*) INTO v_emps FROM employees
     WHERE department_id=v_deptid;
   WHEN  200 THEN
    ...
 END CASE;
DBMS_OUTPUT.PUT_LINE ('You are working in the '|| deptname||
' department. There are '||v_emps ||' employees in this
department');
END;
/
```

Обробка NULL

- При роботі з NULL, можна уникнути деяких поширених помилок, маючи на увазі наступні правила:
  - Прості порівняння, що включають NULL завжди дають значення NULL.
  - Якщо умова повертає NULL, в умовних операторах управління, пов'язаних з ним, послідовність операторів не виконується.

Логічні таблиці

- Створення простої логічної умови з оператором порівняння.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Логічні умови

- Що таке значення `flag` в кожному конкретному випадку?

```
flag := reorder_flag AND available_flag;
```

REORDER_FLAG	AVAILABLE_FLAG	FLAG
TRUE	TRUE	? (1)
TRUE	FALSE	? (2)
NULL	TRUE	? (3)
NULL	FALSE	? (4)

Ітераційне управління: оператор LOOP

- Є три типи циклів:
  - основний цикл
  - FOR цикл
  - WHILE цикл

Основний цикл

- Синтаксис:

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```

Основний цикл

- Приклад:

```
DECLARE
  v_countryid    locations.country_id%TYPE := 'CA';
  v_loc_id       locations.location_id%TYPE;
  v_counter      NUMBER(2) := 1;
  v_new_city     locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
/
```

Цикл WHILE$_1$

- Синтаксис:

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- Використовуйте цикл з умовою продовження (WHILE) для повторення операторів, поки умова має значення TRUE.

Цикл WHILE$_2$

- Приклад:

```
DECLARE
  v_countryid   locations.country_id%TYPE := 'CA';
  v_loc_id      locations.location_id%TYPE;
  v_new_city    locations.city%TYPE := 'Montreal';
  v_counter     NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
  END LOOP;
END;
/
```

Цикл FOR$_1$

- Цикл FOR використовується для виконання певної кількості ітерацій.
- Лічильник не оголошується; він оголошений неявно.

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Цикл FOR$_2$

- Приклад:

```
DECLARE
  v_countryid   locations.country_id%TYPE := 'CA';
  v_loc_id      locations.location_id%TYPE;
  v_new_city    locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id
    FROM locations
    WHERE country_id = v_countryid;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + i), v_new_city, v_countryid );
  END LOOP;
END;
/
```

Цикл FOR$_3$

- Інструкції:
  - Лічильник всередині циклу є невизначеним поза циклом.
  - Не посилайтеся на лічильник в якості цільового призначення.
  - Ніякий пов’язаний цикл не повинен мати значення NULL.

Інструкції для циклів

- Основний цикл використовується, коли оператори всередині циклу повинні виконатися принаймні один раз.
- WHILE цикл використовується, якщо умова має бути провірена на початку кожної ітерації.
- FOR цикл використовується, якщо кількість ітерацій відома.

Вкладені цикли і мітки

- Можна вкладати цикли для декількох рівнів.
- Мітки використовуються, щоб розрізняти блоки і цикли.
- Вихід з зовнішнього циклу здійснюється за допомогою оператора EXIT, яке посилається на мітку.

Вкладені цикли і мітки


```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
/
```

Оператор CONTINUE

- Визначення:
  - Додає функціональність для початку наступної ітерації циклу
  - Забезпечує програмістам можливість передачі управління на наступну ітерацію циклу
  - Використовуються паралельні структури до оператора EXIT

- Переваги:
  - Полегшує процес програмування
  - Може з'явитися невелике поліпшення продуктивності в порівнянні з попередніми обхідними шляхами програмування, для імітації оператора CONTINUE

Оператор CONTINUE: Приклад$_1$

```
DECLARE
  v_total SIMPLE_INTEGER := 0;
BEGIN
  FOR i IN 1..10 LOOP
1   v_total := v_total + i;
    dbms_output.put_line
     ('Total is: '|| v_total);
    CONTINUE WHEN i > 5;
2   v_total := v_total + i;
    dbms_output.put_line
     ('Out of Loop Total is:
      '|| v_total);
  END LOOP;
END;
/
```

```
Total is: 1
Out of Loop Total is: 2
Total is: 4
Out of Loop Total is: 6
Total is: 9
Out of Loop Total is: 12
Total is: 16
Out of Loop Total is: 20
Tot
al is: 25
Out of Loop Total is: 30
Total is: 36
Total is: 43
Total is: 51
Total is: 60
Total is: 70
```

Оператор CONTINUE: Приклад$_2$


```
DECLARE
 v_total NUMBER := 0;
BEGIN
 <<BeforeTopLoop>>
 FOR i IN 1..10 LOOP
   v_total := v_total + 1;
   dbms_output.put_line
     ('Total is: ' || v_total);
   FOR j IN 1..10 LOOP
     CONTINUE BeforeTopLoop WHEN i + j > 5;
     v_total := v_total + 1;
   END LOOP;
 END LOOP;
END two_loop;
```

```
Total is: 1
Total is: 6
Total is: 10
Total is: 13
Total is: 15
Total is: 16
Total is: 17
Total is: 18
Total is: 19
Total is: 20
```

Висновок за розділом

- У цьому розділі можна було дізнатися, як змінити логічний потік операторів за допомогою наступних структур управління:
  - Умовний (оператор `IF`)
  - Оператор `CASE`
- Цикли:
  - основний цикл
  - `FOR` цикл
  - `WHILE` цикл
  - оператори `EXIT` та `CONTINUE`

Робота з складеними типами даних

Складені типи даних$_1$

- Можуть містити кілька значень (на відміну від скалярних типів)
- Мають два типи:
  - Записи PL/SQL
  - PL/SQL набори
    - INDEX BY таблиць
    - асоціативні масиви
    - вкладені таблиці
    - VARRAY

Складені типи даних$_2$

- Використання записів PL/SQL, коли треба зберігати значення різних типів, які логічно пов'язані.
- Використання PL/SQL наборів, коли треба для зберігання значень одного і того ж типу даних.

Записи PL/SQL

- Повинен містити один або кілька компонентів (званих полями) будь-якого скаляра, `RECORD` або `INDEX BY`
- Схожі на структури в більшості мов третього покоління (включаючи C і C++)
- Визначаються користувачем і може бути підмножиною рядків у таблиці
- Обробка набору полів в якості логічної одиниці
- Зручні для витягання рядків даних з таблиці з метою оброблення

Створення записів$_1$

- Синтаксис:

1
```
TYPE type_name IS RECORD
     (field_declaration[, field_declaration]…);
```

2
```
identifier    type_name;
```

field_declaration:

```
field_name {field_type | variable%TYPE
           | table.column%TYPE | table%ROWTYPE}
           [[NOT NULL] {:= | DEFAULT} expr]
```

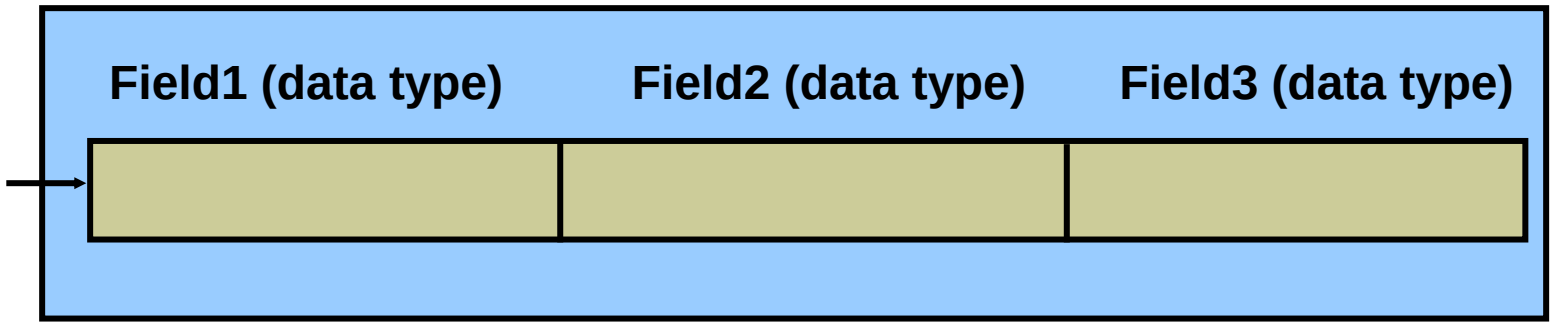
Створення записів$_2$

- Оголошено, що змінні запису зберігають дані про імена, роботу і зарплату та нового співробітника.

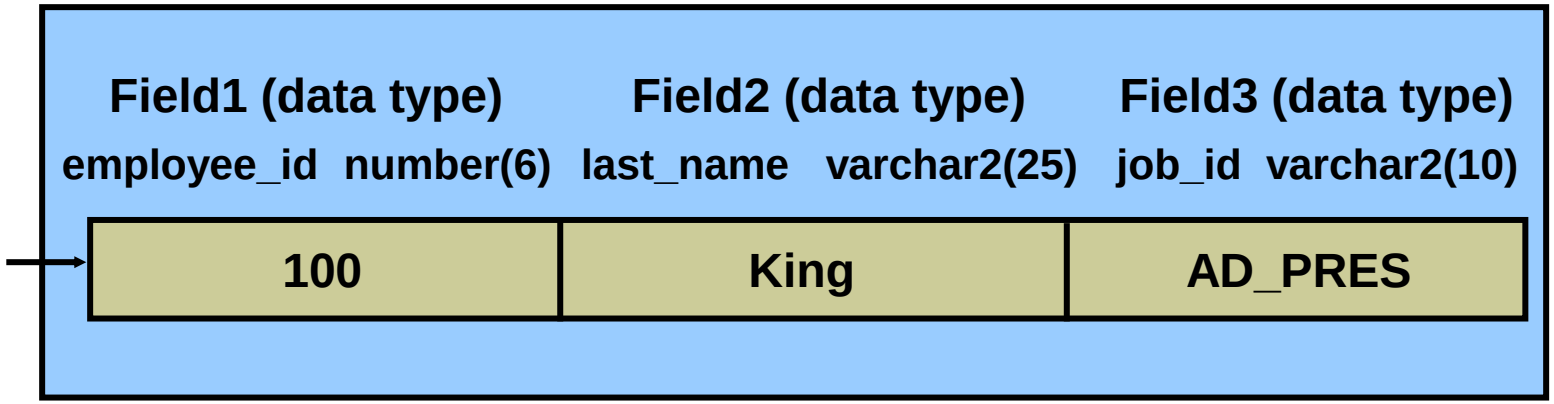
Приклад:

```
DECLARE
  type t_rec is record
    (v_sal number(8),
     v_minsal number(8) default 1000,
     v_hire_date employees.hire_date%type,
     v_rec1 employees%rowtype);
  v_myrec t_rec;
BEGIN
  v_myrec.v_sal := v_myrec.v_minsal + 500;
  v_myrec.v_hire_date := sysdate;
  SELECT * INTO v_myrec.v_rec1
      FROM employees WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE(v_myrec.v_rec1.last_name ||' '||
  to_char(v_myrec.v_hire_date) ||' '|| to_char(v_myrec.v_sal));
END;
```

Структура запису PL/SQL



Приклад:



Атрибут %ROWTYPE

- Оголошено змінну відповідно з набору стовпців таблиці або подання бази даних.
- Поля у записі беруть імена і типи даних з стовпців таблиці.
- Синтаксис:

```
DECLARE
   identifier  reference%ROWTYPE;
```

Переваги використання %ROWTYPE

- Число і типи даних основних стовпців бази даних не повинні бути відомі і, по суті, можуть змінитися під час виконання.
- Атрибут %ROWTYPE корисний при отриманні рядка за допомогою оператора SELECT *

Атрибут %ROWTYPE: Приклад


```
DECLARE
  v_employee_number number:= 124;
  v_emp_rec   employees%ROWTYPE;
BEGIN
 SELECT * INTO v_emp_rec FROM employees
 WHERE  employee_id = v_employee_number;
 INSERT INTO retired_emps(empno, ename, job, mgr,
                 hiredate, leavedate, sal, comm, deptno)
   VALUES (v_emp_rec.employee_id, v_emp_rec.last_name,
           v_emp_rec.job_id, v_emp_rec.manager_id,
           v_emp_rec.hire_date, SYSDATE,
           v_emp_rec.salary,  v_emp_rec.commission_pct,
           v_emp_rec.department_id);
END;
/
```

Вставка запису за допомогою %ROWTYPE


```
...
DECLARE
  v_employee_number number:= 124;
  v_emp_rec retired_emps%ROWTYPE;
BEGIN
 SELECT employee_id, last_name, job_id, manager_id,
 hire_date, hire_date, salary, commission_pct,
 department_id INTO v_emp_rec FROM employees
 WHERE  employee_id = v_employee_number;
 INSERT INTO retired_emps VALUES v_emp_rec;
END;
/
SELECT * FROM retired_emps;
```

Оновлення рядка в таблиці за допомогою запису


```
SET VERIFY OFF
DECLARE
  v_employee_number number:= 124;
  v_emp_rec retired_emps%ROWTYPE;
BEGIN
 SELECT * INTO v_emp_rec FROM retired_emps;
 v_emp_rec.leavedate:=CURRENT_DATE;
 UPDATE retired_emps SET ROW = v_emp_rec WHERE
  empno=v_employee_number;
END;
/
SELECT * FROM retired_emps;
```

INDEX BY таблиці або асоціативні масиви

- PL/SQL структури з двома колонками:
  - Первинний ключ як ціле число або строковий тип даних
  - Колонка скаляру або тип запису даних
- Необмежені в розмірах. Однак, розмір залежить від значень, що може містити ключовий тип даних.

Створення таблиць INDEX BY

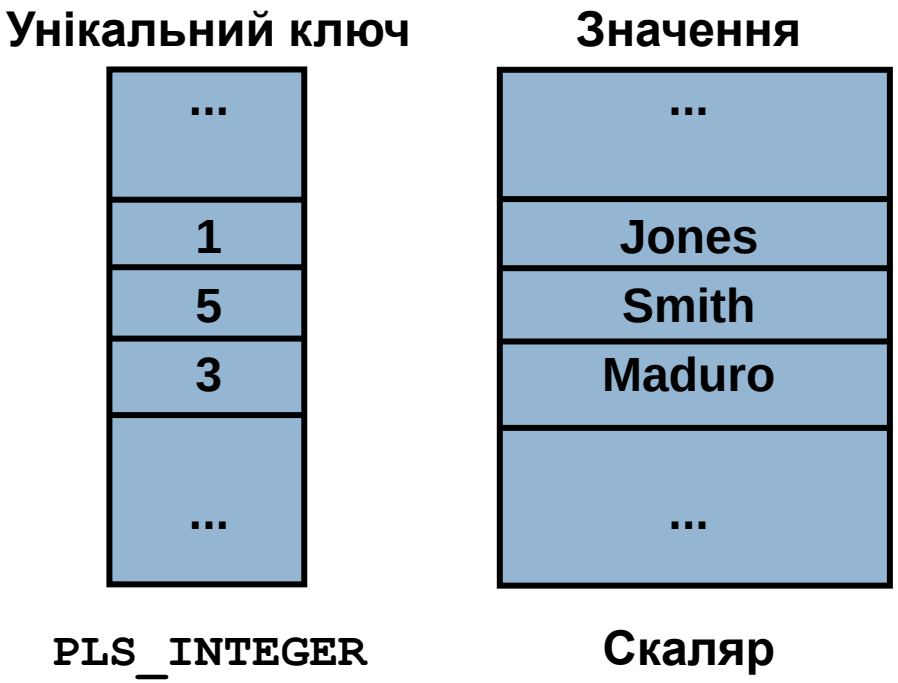
- Синтаксис:

```
TYPE type_name IS TABLE OF
     {column_type | variable%TYPE
     | table.column%TYPE} [NOT NULL]
     | table%ROWTYPE
     [INDEX BY PLS_INTEGER | BINARY_INTEGER
      | VARCHAR2(<size>)];
identifier   type_name;
```

- Оголосимо, що таблиця INDEX BY зберігає прізвища співробітників:

```
...
TYPE ename_table_type IS TABLE OF
 employees.last_name%TYPE
 INDEX BY PLS_INTEGER;
 ...
ename_table ename_table_type;
```

Таблична структура INDEX BY



Створення таблиці INDEX BY


```
DECLARE
   TYPE ename_table_type IS
      TABLE OF employees.last_name%TYPE INDEX BY PLS_INTEGER;
   TYPE hiredate_table_type IS
      TABLE OF DATE INDEX BY PLS_INTEGER;
   ename_table    ename_table_type;
   hiredate_table hiredate_table_type;
BEGIN
  ename_table(1)    := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
   IF ename_table.EXISTS(1) THEN
   INSERT
   into.....end;
/
```

Використання INDEX BY в табличних методах

- Наступні методи роблять INDEX BY таблиці простіше у використанні:
- EXISTS
  - COUNT
  - FIRST
  - LAST
  - PRIOR
  - NEXT
  - DELETE

Таблиця запису INDEX BY

- Визначимо змінну таблиці INDEX BY для зберігання всього рядка таблиці.
- Приклад:

```
DECLARE
  TYPE dept_table_type IS TABLE OF
       departments%ROWTYPE
       INDEX BY VARCHAR2(20);
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```

Таблиця запису INDEX BY: Приклад

Worksheet | Query Builder

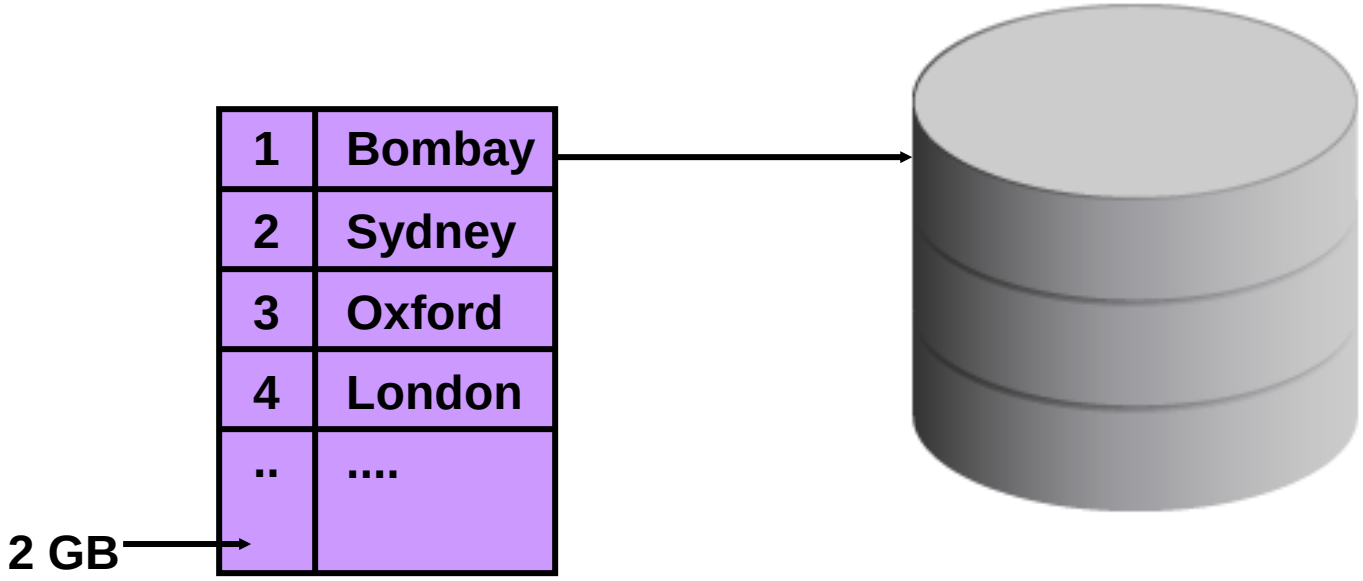
```
DECLARE
   TYPE emp_table_type IS TABLE OF
      employees%ROWTYPE INDEX BY PLS_INTEGER;
   my_emp_table  emp_table_type;
   max_count         NUMBER(3):= 104;
BEGIN
  FOR i IN 100..max_count
  LOOP
   SELECT * INTO my_emp_table(i) FROM employees
   WHERE employee_id = i;
  END LOOP;
  FOR i IN my_emp_table.FIRST..my_emp_table.LAST
  LOOP
     DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
  END LOOP;
END;
/
```

Dbms Output

ORCL

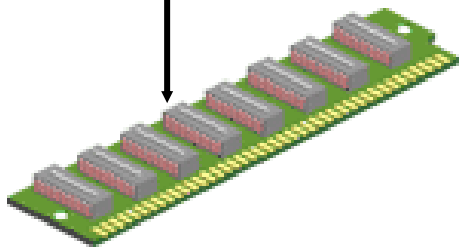
```
King
Yang
Garcia
James
Miller
```

Вкладені таблиці



VARRAY (масив змінного розміру)

1	Bombay
2	Sydney
3	Oxford
4	London
..	....
10	Tokyo



Висновок за розділом

- У цьому розділі розглянуті наступні питання:
  - Визначення і створення записів PL/SQL
  - Робота з:
    - записами
    - INDEX BY таблицями
    - таблицями записів INDEX BY
  - Визначення PL/SQL записів за допомогою атрибута %ROWTYPE

Донецький національний технічний університет

Тема 9. Мова програмування PL/SQL. Лекція 15. PL/SQL. Використання явних курсорів.

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

Лекційні питання

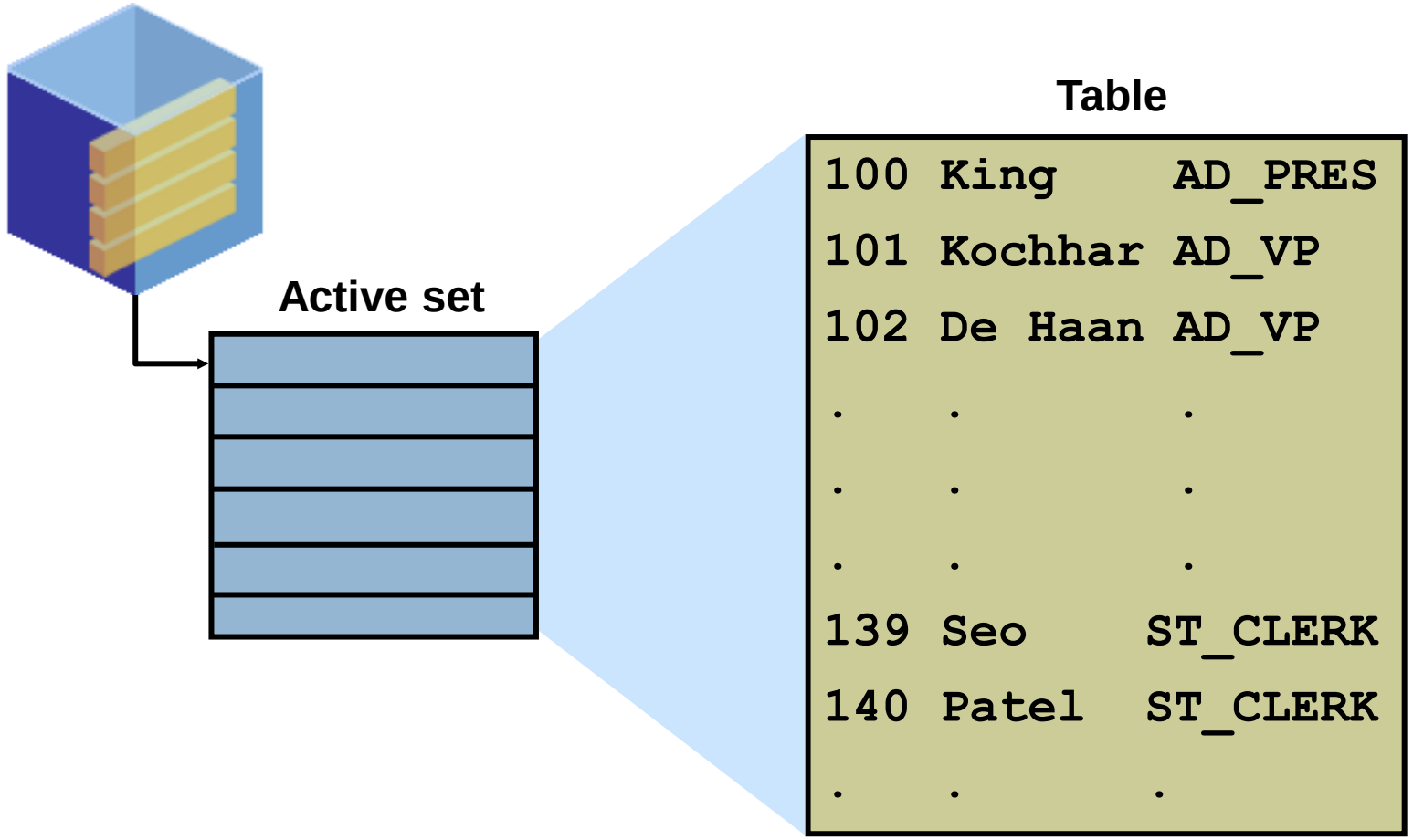
- Використання явних курсорів
- Обробка винятків
- Створення збережених процедур і функцій
- Створення тригерів

Використання явних курсорів

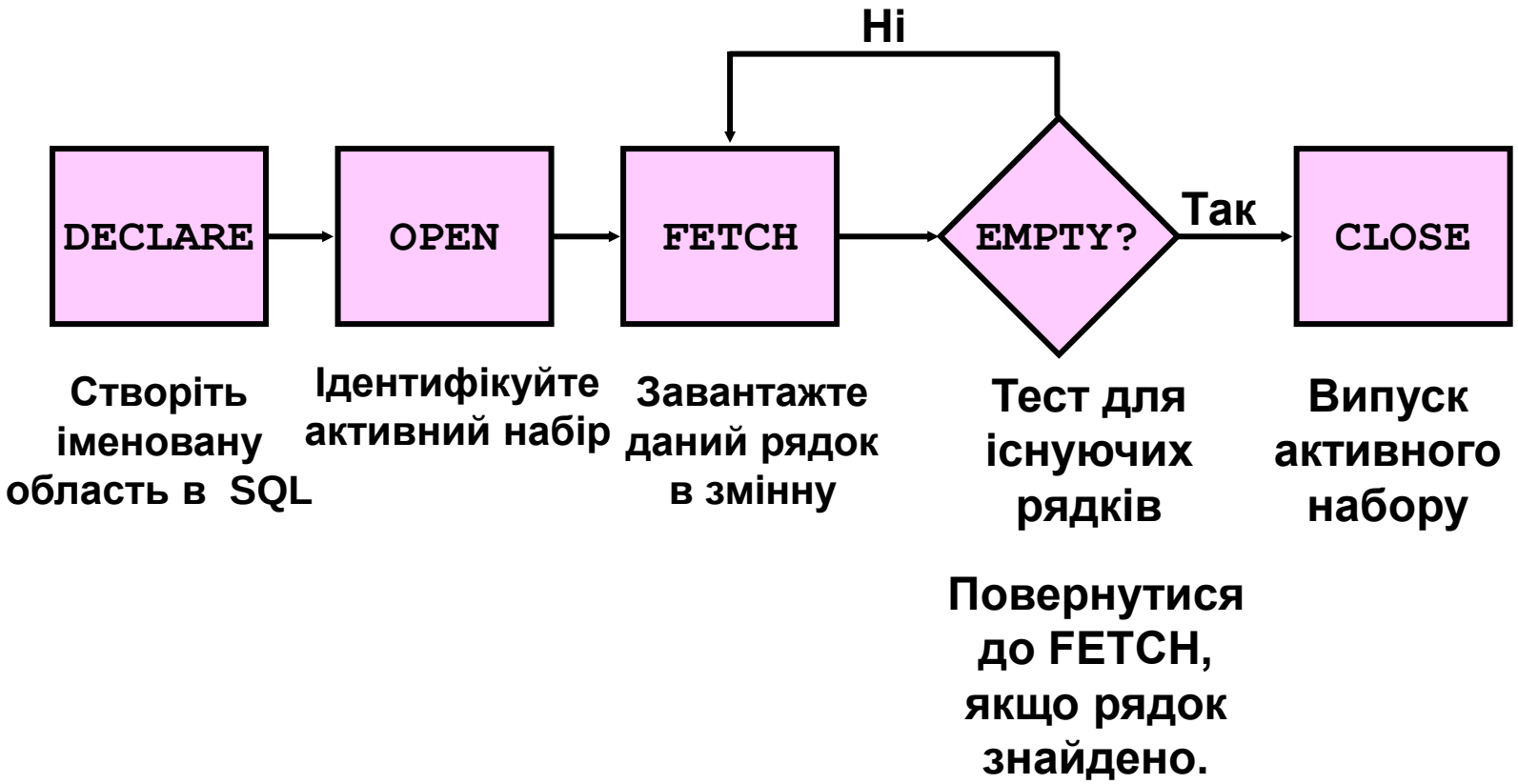
Курсори

- Кожний SQL–запит виконується на сервері Oracle та має відповідний індивідуальний курсор:
  - Неявні курсори: Оголошуються і управляються PL/SQL для всіх операторів DML і PL/SQL Select
  - Явні курсори: Оголошуються і управляються програмістом

Явні операції з курсором

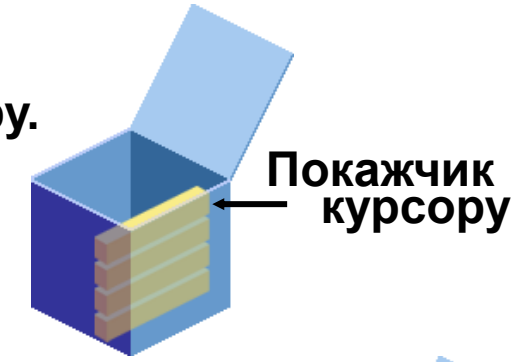


Управління явними курсорами$_1$



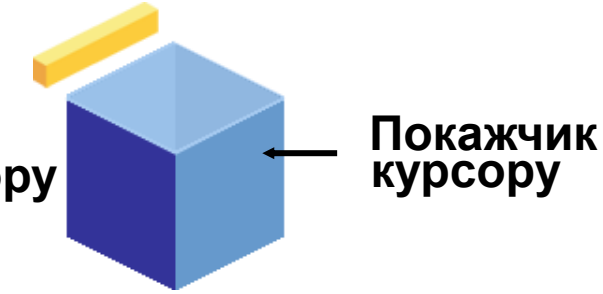
Управління явними курсорами$_2$

1. Відкриття курсору.



2. Вибір рядка

3. Закриття курсору



Оголошення курсору

- Синтаксис:

```
CURSOR cursor_name IS
     select_statement;
```

- Приклади:

```
DECLARE
  CURSOR c_emp_cursor IS
  SELECT employee_id, last_name FROM employees
  WHERE department_id =30;
```

```
DECLARE
  v_locid NUMBER:= 1700;
  CURSOR c_dept_cursor IS
  SELECT * FROM departments
  WHERE location_id = v_locid;
...
```

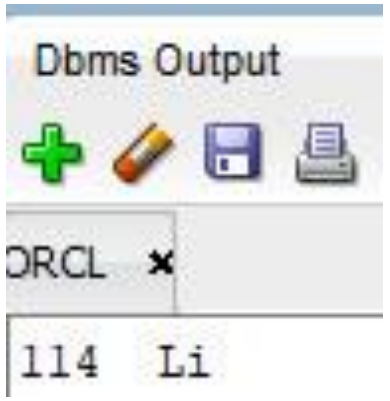
Відкриття курсору

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE department_id =30;
...
BEGIN
  OPEN c_emp_cursor;
```

Отримання даних з курсору$_1$

Worksheet | Query Builder

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE department_id =30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN c_emp_cursor;
  FETCH c_emp_cursor INTO v_empno, v_lname;
  DBMS_OUTPUT.PUT_LINE( v_empno ||'  '||v_lname);
END;

/
```



Отримання даних з курсору$_2$

Worksheet | Query Builder

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE department_id =30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_empno, v_lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno ||'  '||v_lname);
  END LOOP;
END;
/
```

Dbms Output

Buffer Size: 20000

ORCL

```
114  Li
115  Khoo
116  Baida
117  Tobias
118  Himuro
119  Colmenares
```

Закриття курсору


```
...
  LOOP
    FETCH c_emp_cursor INTO empno, lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  END LOOP;
 CLOSE c_emp_cursor;
END;
/
```

Курсори і записи

- Обробка рядків активного набору шляхом вибірки значень в записах PL/SQL.

Worksheet | Query Builder

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE  department_id =30;
   v_emp_record c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_emp_record;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                        ||' '||v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END;
```

Dbms Output

Buffer Size: 20000

ORCL

```
114 Li
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

Курсори для циклів$_1$

- Синтаксис:

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- Курсор циклу - це ярлик для обробки явних курсорів.
- Неявне відкриття, вибірка, вихід.
- Записи неявно оголошуються.

Курсори для циклів$_2$

Worksheet | Query Builder

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE department_id =30;
BEGIN
   FOR emp_record IN c_emp_cursor
    LOOP
     DBMS_OUTPUT.PUT_LINE( emp_record.employee_id  || ' ' ||emp_record.last_name);
    END LOOP;
END;
/
```

Dbms Output

Buffer Size: 20000

ORCL ×

```
114 Li
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

Явні атрибути курсору

- Використовуються явні атрибути курсору для отримання інформацію про статус курсору.

Атрибути	Тип	Опис
%ISOPEN	Boolean	Значення TRUE, якщо курсор відкритий
%NOTFOUND	Boolean	Значення TRUE, якщо нова вибірка не повертає рядок
%FOUND	Boolean	Значення TRUE, якщо нова вибірка повертає рядок, з доповненням %NOTFOUND
%ROWCOUNT	Number	Оцінює загальну кількість рядків

Атрибут %ISOPEN

- Вибирає рядки тільки тоді, коли курсор відкритий.
- Атрибут %ISOPEN використовується для перевірки, чи є курсор відкритим.
- Приклад:

```
IF NOT c_emp_cursor%ISOPEN THEN
    OPEN c_emp_cursor;
END IF;
LOOP
  FETCH c_emp_cursor...
```

%ROWCOUNT і %NOTFOUND: Приклад

Worksheet | Query Builder

```
DECLARE
  CURSOR c_emp_cursor IS SELECT employee_id,
    last_name FROM employees;
  v_emp_record  c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
   FETCH c_emp_cursor INTO v_emp_record;
   EXIT WHEN c_emp_cursor%ROWCOUNT > 10 OR
                      c_emp_cursor%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                ||' '||v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END ;
/
```

Dbms Output

Buffer Size

ORCL ×

```
174 Abel
166 Ande
130 Atkinson
116 Baida
167 Banda
172 Bates
192 Bell
151 Bernstein
129 Bissot
169 Bloom
```

Курсор для циклів, використання підзапитів

- Немає необхідності оголошувати курсор.
- Приклад:

Worksheet | Query Builder

```
BEGIN
  FOR emp_record IN (SELECT employee_id, last_name    FROM employees WHERE department_id =30)
  LOOP
   DBMS_OUTPUT.PUT_LINE(emp_record.employee_id
   ||' '||emp_record.last_name);
  END LOOP;
END;
/
```

Dbms Output | Buffer Siz

ORCL ×

```
114 Li
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

Курсори з параметрами$_1$

- Синтаксис:

```
CURSOR cursor_name
  [(parameter_name datatype, ...)]
IS
  select_statement;
```

- Передача параметрів курсора відбувається при відкритому курсорі, саме тоді запит виконується.
- Відкриємо явний курсор кілька разів з іншим активним набором кожен раз.

```
OPEN  cursor_name(parameter_value,.....) ;
```

Курсори з параметрами$_2$


```
DECLARE
  CURSOR   c_emp_cursor (deptno NUMBER) IS
   SELECT  employee_id, last_name
   FROM    employees
   WHERE   department_id = deptno;
   ...
BEGIN
  OPEN c_emp_cursor (10);
  ...
  CLOSE c_emp_cursor;
  OPEN c_emp_cursor (20);
  ...
```

FOR UPDATE

- Синтаксис:

```
SELECT ...
FROM           ...
FOR UPDATE [OF column_reference][NOWAIT | WAIT n];
```

- Явне блокування використовується для заборони доступу до інших сеансів на час транзакції.
- До оновлення краще заблокувати або видалити рядки.

WHERE CURRENT OF

- Синтаксис:

```
WHERE CURRENT OF cursor ;
```

- Використовуйте даний тип курсора для оновлення або видалення поточного рядка.
- Додайте FOR UPDATE у запит курсору, щоб зафіксувати перші рядки.
- Використовуйте WHERE CURRENT для посилання на поточний рядок з явного курсору.

```
UPDATE employees
   SET    salary = ...
   WHERE CURRENT OF c_emp_cursor;
```

Курсори з підзапитами

- Приклад:

```
DECLARE
  CURSOR my_cursor IS
    SELECT t1.department_id, t1.department_name,
           t2.staff
    FROM   departments t1, (SELECT department_id,
                                   COUNT(*) AS staff
                            FROM employees
                            GROUP BY department_id) t2
    WHERE t1.department_id = t2.department_id
    AND   t2.staff >= 3;
...
```

Висновок за розділом

- У цьому розділі розглянуті наступні питання:
  - Типи курсорів:
    - Неявні курсори використовуються для всіх DML та однорядкових запитів.
    - Явні курсори використовуються для запитів з нулем, одним або кількома рядками.
  - Створення та обробка явних курсорів
    - Використання простих циклів і циклів курсора для роботи з декількома рядками.
    - Оцінювання стану курсору за допомогою атрибутів курсору
  - Використання FOR UPDATE та WHERE CURRENT для оновлення або видалення поточного рядка

Обробка винятків

Приклад виняткової ситуації


```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John''s last name is :'
                        ||v_lname);
END;
```

```
Error report -
ORA-01422: exact fetch returned more than the requested number of rows
ORA-06512: at line 4
01422. 00000 -  "exact fetch returns more than requested number of rows"
*Cause:    The number specified in exact fetch is less than the rows returned.
*Action:   Rewrite the query or change number of rows requested
```

Обробка винятків$_1$

Worksheet | Query Builder

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John''s last name is :'
                        ||v_lname);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
  DBMS_OUTPUT.PUT_LINE (' Your select statement retrieved multiple rows.
  Consider using a cursor.');
END;
/
```

Dbms Output

Buffer Size: 20000

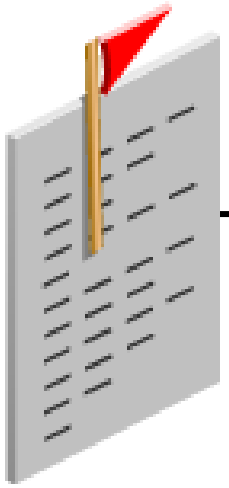
ORCL

```
Your select statement retrieved multiple rows.
  Consider using a cursor.
```

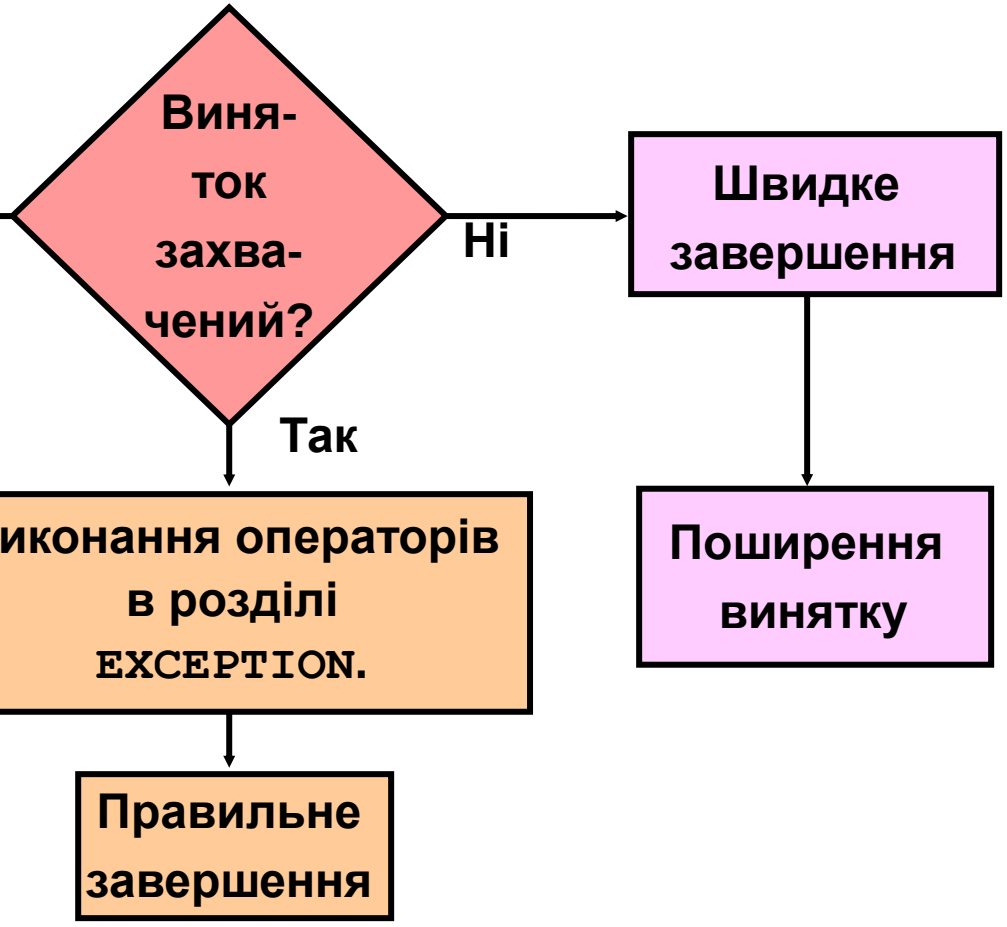
Обробка винятків$_2$

- Винятком є PL/SQL помилка, яка виникає в процесі виконання програми.
- Виняток може бути викликаний:
  - Неявно: сервером Oracle
  - Явно: програмою
- Виняток може бути оброблений:
  - Шляхом захоплення його в обробник
  - Шляхом поширення його в середовище виклику

Обробка винятків$_3$



Виняток піднято



Типи винятків

- Зумовлені сервером Oracle
- Незумовлені сервером Oracle

} **Неявно підвищені**

- Зумовлені користувачем

Явно підвищені

Перехоплення винятків

- Синтаксис:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

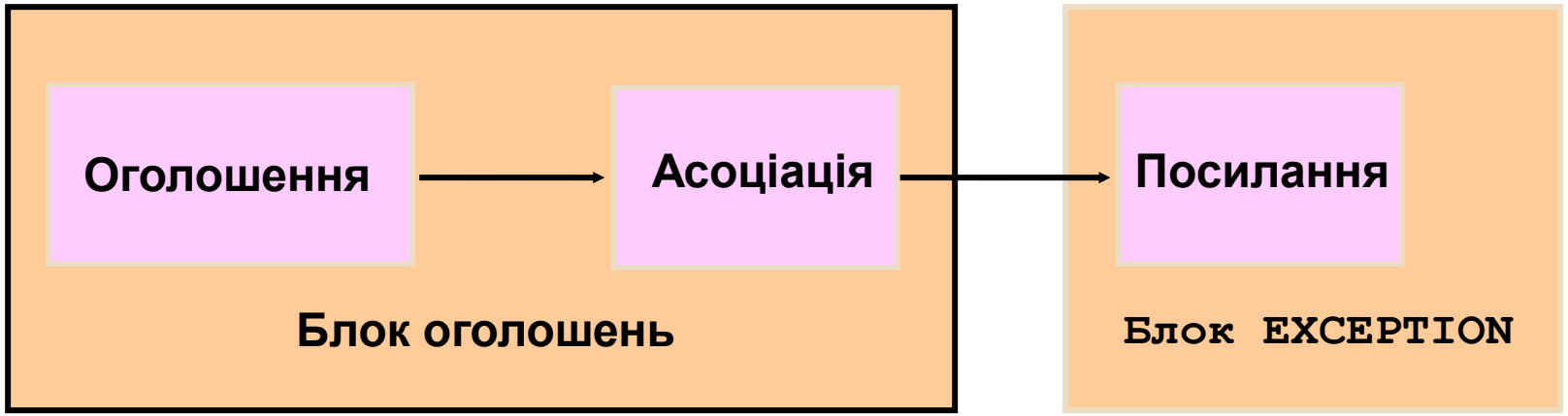
Інструкція для перехоплення винятків

- Ключове слово EXCEPTION починає обробку винятків.
- Допускається кілька обробників винятків.
- WHEN OTHERS є останнім пунктом.

Перехоплення зумовлених помилок сервера Oracle

- Варіанти зумовлених винятків:
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_ON_INDEX

Захоплення незумовлених помилок сервера Oracle



Незумовлена помилка

- Перехоплення помилки сервером Oracle `-01400` (“cannot insert `NULL`”):

```
DECLARE
 e_insert_excep EXCEPTION;                          ← 1
 PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);     ← 2
BEGIN
 INSERT INTO departments
 (department_id, department_name) VALUES (280, NULL);
EXCEPTION                                           ← 3
 WHEN e_insert_excep THEN
   DBMS_OUTPUT.PUT_LINE('INSERT OPERATION FAILED');
   DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

Функції для захоплення винятків$_1$

- SQLCODE: Повертає числове значення для коду помилки
- SQLERRM: Повертає повідомлення, пов'язане з номером помилки

Функції для захоплення винятків$_2$

- Приклад:

```
DECLARE
  error_code       NUMBER;
  error_message    VARCHAR2(255);
BEGIN
...
EXCEPTION
...
  WHEN OTHERS THEN
    ROLLBACK;
    error_code := SQLCODE ;
    error_message := SQLERRM ;
   INSERT INTO errors (e_user, e_date, error_code,
   error_message) VALUES(USER,SYSDATE,error_code,
   error_message);
END;
/
```

Перехоплення винятків,які зумовлені користувачем$_1$

Називаємо виняток

Підвищення за допомогою оператора RAISE

Обробка винятків

Перехоплення винятків,які зумовлені користувачем$_2$

```
DECLARE
  v_deptno NUMBER := 500;
  v_name VARCHAR2(20) := 'Testing';
  e_invalid_department EXCEPTION;
BEGIN
  UPDATE departments
  SET department_name = v_name
  WHERE department_id = v_deptno;
  IF SQL % NOTFOUND THEN
    RAISE e_invalid_department;
  END IF;
  COMMIT;
EXCEPTION
WHEN e_invalid_department THEN
  DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
/
```

Поширення винятків у підблоках

Підблоки можуть обробляти винятки, або передавати їх в зовнішній блок.

```
DECLARE
  . . .
  e_no_rows       exception;
  e_integrity     exception;
  PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
     SELECT ...
     UPDATE ...
     IF SQL%NOTFOUND THEN
       RAISE e_no_rows;
     END IF;
    END;
  END LOOP;
EXCEPTION
  WHEN e_integrity THEN ...
  WHEN e_no_rows THEN ...
END;
/
```

Процедура RAISE_APPLICATION_ERROR[1]

- Приклад:

```
raise_application_error (error_number,
                         message[, {TRUE | FALSE}]);
```

- Можна використовувати дану процедуру для виведення повідомлення про помилки з збережених підпрограм.
- Можна повідомити про помилки в застосунках і уникнути повернення необроблених винятків.

Процедура RAISE_APPLICATION_ERROR$_2$

- Процедура використовується в двох різних місцях:
  - в виконавчому розділі
  - в розділі винятків
- Повертає помилку для користувача у відповідності з іншими помилками сервера Oracle

Процедура RAISE_APPLICATION_ERROR$_3$

- Виконавчий розділ:

```
BEGIN
...
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
       'This is not a valid manager');
  END IF;
   ...
```

- Розділ винятків:

```
...
EXCEPTION
   WHEN NO_DATA_FOUND THEN
     RAISE_APPLICATION_ERROR (-20201,
        'Manager is not a valid employee.');
END;
/
```

Висновок за розділом

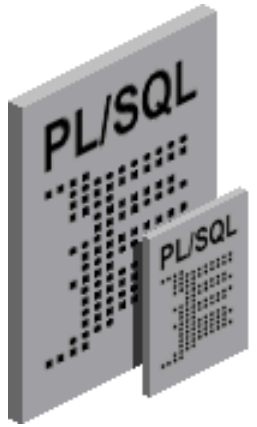
У цьому розділі розглянуті наступні питання:

- ❑Визначення PL/SQL винятків
- ❑Додавання розділу EXCEPTION в PL/SQL блоки
- ❑Обробка різних типів винятків:
  - Номери для зумовлених винятків
  - Винятки, які використовуються користувачем
  - Передавання винятків у вкладені блоки і виклики застосунків

Створення збережених процедур і функцій

Процедури і функції

- Можуть викликатись PL/SQL підпрограмами
- Мають блочні структури, подібні до анонімних блоків:
  - Існують додаткові секції оголошень (без ключового слова DECLARE)
  - Обов'язковий виконавчий розділ
  - Додатковий розділ для обробки виключень



Відмінності між анонімними блоками і підпрограмами

Анонімні блоки	Підпрограми
Безіменні PL / SQL блоки	Іменовані PL / SQL блоки
Компілюється кожен раз	Компілюється лише раз
Не зберігається в базі даних	Зберігається в базі даних
Не можуть бути викликані іншими застосунками	Іменовані і, отже, можуть бути викликані іншими застосунками
Не повертає значення	Підпрограми визиваються функціями, які повинні повертати значення.
Не можуть приймати параметри	Можуть приймати параметри

Процедура: Синтаксис


```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(argument1 [mode1] datatype1,
  argument2 [mode2] datatype2,
  . . .)]
IS|AS
procedure_body;
```

Процедура: Приклад

Worksheet | Query Builder

```
CREATE TABLE dept AS SELECT * FROM departments;
CREATE PROCEDURE add_dept IS
 v_dept_id dept.department_id%TYPE;
 v_dept_name dept.department_name%TYPE;
BEGIN
 v_dept_id:=280;
 v_dept_name:='ST-Curriculum';
 INSERT INTO dept(department_id,department_name)
 VALUES(v_dept_id,v_dept_name);
 DBMS_OUTPUT.PUT_LINE(' Inserted '|| SQL%ROWCOUNT ||' row ');
END;
```

Script Output | Query Result

Task completed in 3.139 seconds

```
Table DEPT created.


Procedure ADD_DEPT compiled
```

28	280	ST-Curriculum	(null)	(null)

Виклик процедури

```
BEGIN
 add_dept;
END;
/
SELECT department_id, department_name FROM dept
WHERE department_id=280;
```

```
anonymous block completed
 Inserted 1 row

DEPARTMENT_ID          DEPARTMENT_NAME
---------------------- ------------------------------
280                    ST-Curriculum

1 rows selected
```

Функція: Синтаксис

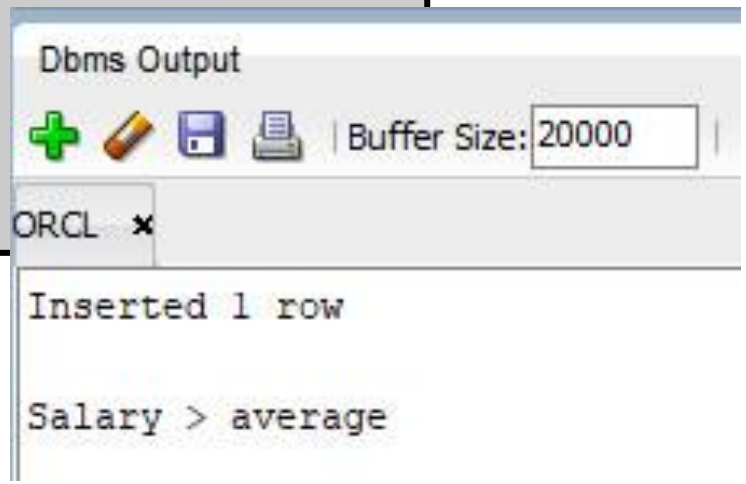
```
CREATE [OR REPLACE] FUNCTION function_name
 [(argument1 [mode1] datatype1,
  argument2 [mode2] datatype2,
  . . .)]
RETURN datatype
IS|AS
function_body;
```

Функція: Приклад

```
CREATE FUNCTION check_sal RETURN Boolean IS
v_dept_id employees.department_id%TYPE;
 v_empno   employees.employee_id%TYPE;
 v_sal     employees.salary%TYPE;
 v_avg_sal employees.salary%TYPE;
BEGIN
 v_empno:=205;
 SELECT salary,department_id INTO v_sal,v_dept_id FROM
employees
 WHERE employee_id= v_empno;
 SELECT avg(salary) INTO v_avg_sal FROM employees WHERE
department_id=v_dept_id;
 IF v_sal > v_avg_sal THEN
  RETURN TRUE;
 ELSE
  RETURN FALSE;
 END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   RETURN NULL;
END;
```

Виклик функції

```
BEGIN
 IF (check_sal IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
  NULL due to exception');
 ELSIF (check_sal) THEN
 DBMS_OUTPUT.PUT_LINE('Salary > average');
 ELSE
 DBMS_OUTPUT.PUT_LINE('Salary < average');
 END IF;
END;
/
```



Передача параметра функції


```
DROP FUNCTION check_sal;
CREATE FUNCTION check_sal(p_empno employees.employee_id%TYPE)
RETURN Boolean IS
 v_dept_id employees.department_id%TYPE;
 v_sal     employees.salary%TYPE;
 v_avg_sal employees.salary%TYPE;
BEGIN
 SELECT salary,department_id INTO v_sal,v_dept_id FROM employees
   WHERE employee_id=p_empno;
 SELECT avg(salary) INTO v_avg_sal FROM employees
   WHERE department_id=v_dept_id;
 IF v_sal > v_avg_sal THEN
  RETURN TRUE;
 ELSE
  RETURN FALSE;
 END IF;
EXCEPTION
  ...
```

Виклик функції з параметром

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Checking for employee with id 205');
 IF (check_sal(205) IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
  NULL due to exception');
 ELSIF (check_sal(205)) THEN
 DBMS_OUTPUT.PUT_LINE('Salary > average');
 ELSE
 DBMS_OUTPUT.PUT_LINE('Salary < average');
 END IF;
DBMS_OUTPUT.PUT_LINE('Checking for employee with id 70');
 IF (check_sal(70) IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
  NULL due to exception');
 ELSIF (check_sal(70)) THEN
 ...
 END IF;
END;
/
```

Висновок за розділом

- У цьому розділі розглянуті наступні питання:
  - Створення простих процедур
  - Виклик процедур з анонімного блоку
  - Створення простих функцій
  - Створення простих функцій, яка приймає параметри
  - Виклик функції з анонімного блоку

Створення тригерів

Типи тригерів

- Тригер:
  - Це блок PL/SQL або процедура PL/SQL, пов’язана з таблицею, поданням, схемою або базою даних
  - Виконується неявно щоразу, коли відбувається певна подія
  - Може бути одним із наступних:
    - Тригер програми: спрацьовує щоразу, коли відбувається подія з певною програмою
    - Тригер бази даних: спрацьовує кожного разу, коли в схемі або базі даних відбувається подія даних (наприклад, DML) або системна подія (наприклад, вхід або завершення роботи).

Інструкції з розробки тригерів

- Можна створювати тригери для:
  - Виконання пов’язаних дій
  - Централізації глобальних операцій
- Не потрібно проектувати тригери:
  - Де функціональні можливості вже вбудовані в сервер Oracle
  - Якщо дублюються інші тригери
- Можна створювати збережені процедури та викликати їх у тригері, якщо код PL/SQL дуже довгий.
- Надмірне використання тригерів може призвести до складних взаємозалежностей, які може бути важко підтримувати у великих програмах.

Створення тригерів DML

- Створення виразу DML або рядкового тригеру за допомогою:

```
CREATE [OR REPLACE] TRIGGER trigger_name
 timing
 event1 [OR event2 OR event3]
ON object_name
[[REFERENCING OLD AS old | NEW AS new]
 FOR EACH ROW
 [WHEN (condition)]]
trigger_body
```

- Тригер спрацьовує один раз для оператора DML.
- Рядковий тригер спрацьовує один раз для кожного задіяного рядка.

Примітка. Імена тригерів мають бути унікальними щодо інших тригерів у тій же схемі.

Типи тригерів DML

- Тип тригера визначає, чи виконується тіло для кожного рядка чи лише один раз для оператора ініціювання.
- Тригер виразу:
  - Виконується один раз для події, що запускає
  - Тип тригера за замовчуванням
  - Спрацьовує один раз, навіть якщо жоден рядок не зачеплено
- Рядковий тригер:
  - Виконується один раз для кожного рядка, на який впливає подія запуску
  - Не виконується, якщо подія запуску не впливає на жодні рядки
  - Позначається за допомогою пропозиції FOR EACH ROW

Час запуску

- Коли має спрацювати тригер?
  - BEFORE: Виконання тіла тригера перед ініціюючою подією DML у таблиці.
  - AFTER: Виконання тіла тригера після тригерної події DML у таблиці.
  - INSTEAD OF: Виконання тіла тригера замість оператора ініціювання. Використовується для переглядів, які не можна змінювати іншим чином.

Примітка. Якщо для одного об’єкта визначено кілька тригерів, то порядок запуску тригерів є довільним.

Послідовність спрацьовування тригера$_1$

- Використовується наступна послідовність запуску для тригера в таблиці при маніпуляції одним рядком

Запит DML

```
INSERT INTO departments
   (department_id,department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```

Triggering action → **BEFORE statement trigger**

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
...		
400	CONSULTING	2400

→ **BEFORE row trigger**

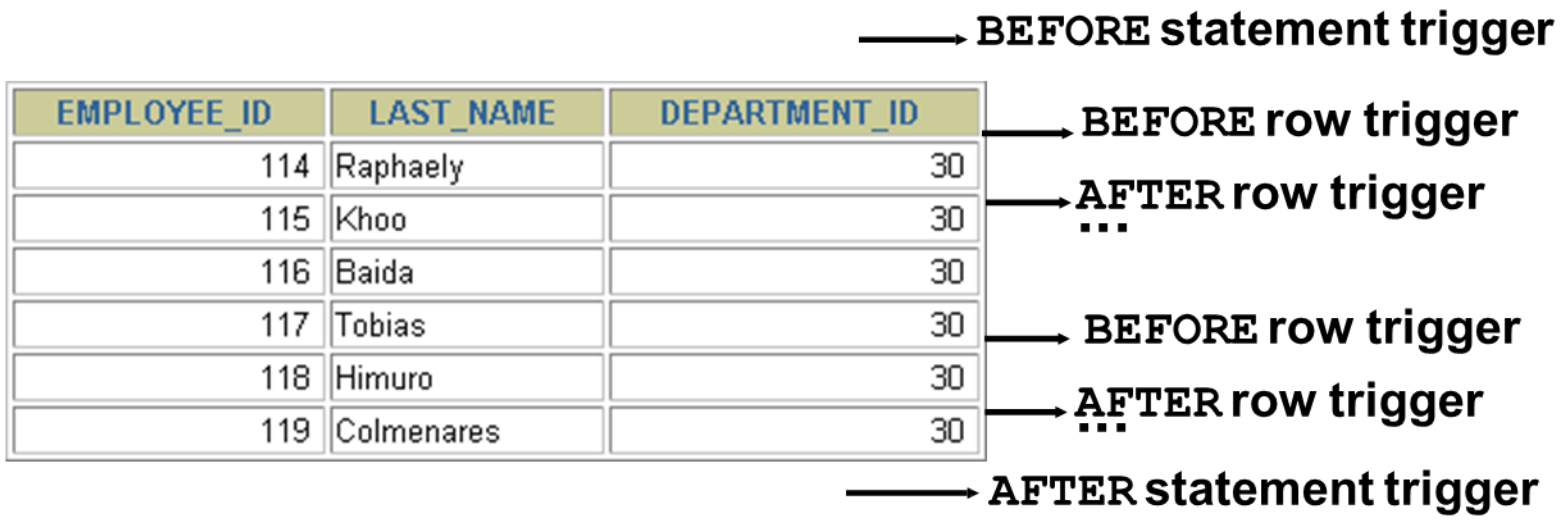
→ **AFTER row trigger**

→ **AFTER statement trigger**

Послідовність спрацьовування тригера$_2$

- Використовується наступна послідовність запуску для тригера в таблиці при маніпуляції багатьма рядками:

```
UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 30;
```



Типи і тіло тригерної події

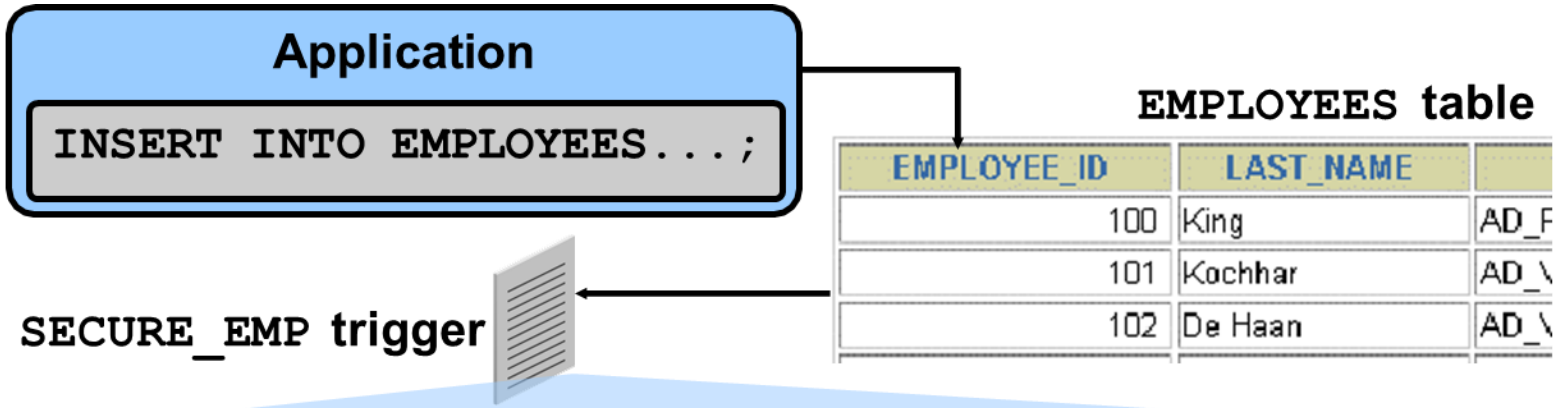
Тригерна подія:

- Визначає, який оператор DML викликає виконання тригера
- Типи:
  - INSERT
  - UPDATE [OF column]
  - DELETE

Тіло тригера:

- Визначає, яка дія виконується
- Є блоком PL/SQL або викликом процедури

Створення тригеру за виразом DML




```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON employees BEGIN
 IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
     (TO_CHAR(SYSDATE,'HH24:MI')
           NOT BETWEEN '08:00' AND '18:00') THEN
  RAISE_APPLICATION_ERROR(-20500, 'You may insert'
    ||' into EMPLOYEES table only during '
    ||' business hours.');
  END IF;
END;
```

Тестування SECURE_EMP

```
INSERT INTO employees (employee_id, last_name,
        first_name, email, hire_date,
        job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
        'IT_PROG', 4500, 60);
```

```
INSERT INTO employees (employee_id, last_name, first_name, email,
       *
ERROR at line 1:
ORA-20500: You may insert into EMPLOYEES table only during business hours.
ORA-06512: at "PLSQL.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'PLSQL.SECURE_EMP'
```

Використання умовних предикатів


```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees BEGIN
IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
   (TO_CHAR(SYSDATE,'HH24')
      NOT BETWEEN '08' AND '18') THEN
   IF DELETING THEN RAISE_APPLICATION_ERROR(
    -20502,'You may delete from EMPLOYEES table'||
           'only during business hours.');
   ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
    -20500,'You may insert into EMPLOYEES table'||
           'only during business hours.');
   ELSIF UPDATING('SALARY') THEN
    RAISE_APPLICATION_ERROR(-20503, 'You may '||
     'update SALARY only during business hours.');
   ELSE RAISE_APPLICATION_ERROR(-20504,'You may'||
     ' update EMPLOYEES table only during'||
     ' normal hours.');
   END IF;
END IF;
END;
```

Створення рядкового DML тригера


```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
  IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
    AND :NEW.salary > 15000 THEN
    RAISE_APPLICATION_ERROR (-20202,
      'Employee cannot earn more than $15,000.');
  END IF;
END;
/
```

Використання OLD і NEW кваліфікаторів


```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO audit_emp(user_name, time_stamp, id,
    old_last_name, new_last_name, old_title,
    new_title, old_salary, new_salary)
  VALUES (USER, SYSDATE, :OLD.employee_id,
    :OLD.last_name, :NEW.last_name, :OLD.job_id,
    :NEW.job_id, :OLD.salary, :NEW.salary);
END;
/
```

Використання OLD і NEW кваліфікаторів: приклад використання audit_emp


```
INSERT INTO employees
 (employee_id, last_name, job_id, salary, ...)
VALUES (999, 'Temp emp', 'SA_REP', 6000,...);

UPDATE employees
 SET salary = 7000, last_name = 'Smith'
 WHERE employee_id = 999;
```

```
SELECT user_name, timestamp, ...
FROM audit_emp;
```

USER_NAME	TIMESTAMP	ID	OLD_LAST_N	NEW_LAST_N	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
PLSQL	28-SEP-01			Temp emp		SA_REP		1000
PLSQL	28-SEP-01	999	Temp emp	Smith	SA_REP	SA_REP	1000	2000

Обмеження рядкового тригера: приклад


```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
 IF INSERTING THEN
   :NEW.commission_pct := 0;
 ELSIF :OLD.commission_pct IS NULL THEN
   :NEW.commission_pct := 0;
 ELSE
   :NEW.commission_pct := :OLD.commission_pct+0.05;
 END IF;
END;
/
```

Резюме щодо моделі виконання тригерів

- Виконання всіх тригерів BEFORE STATEMENT.
- Цикл для кожного задіяного рядка:
  - Виконання всіх тригерів BEFORE ROW.
  - Виконання операторів DML і перевірка обмеження цілісності.
  - Виконання всіх тригери AFTER ROW.
- Виконання всіх тригерів AFTER STATEMENT.

Примітка. Перевірку цілісності можна відкласти до виконання операції COMMIT.

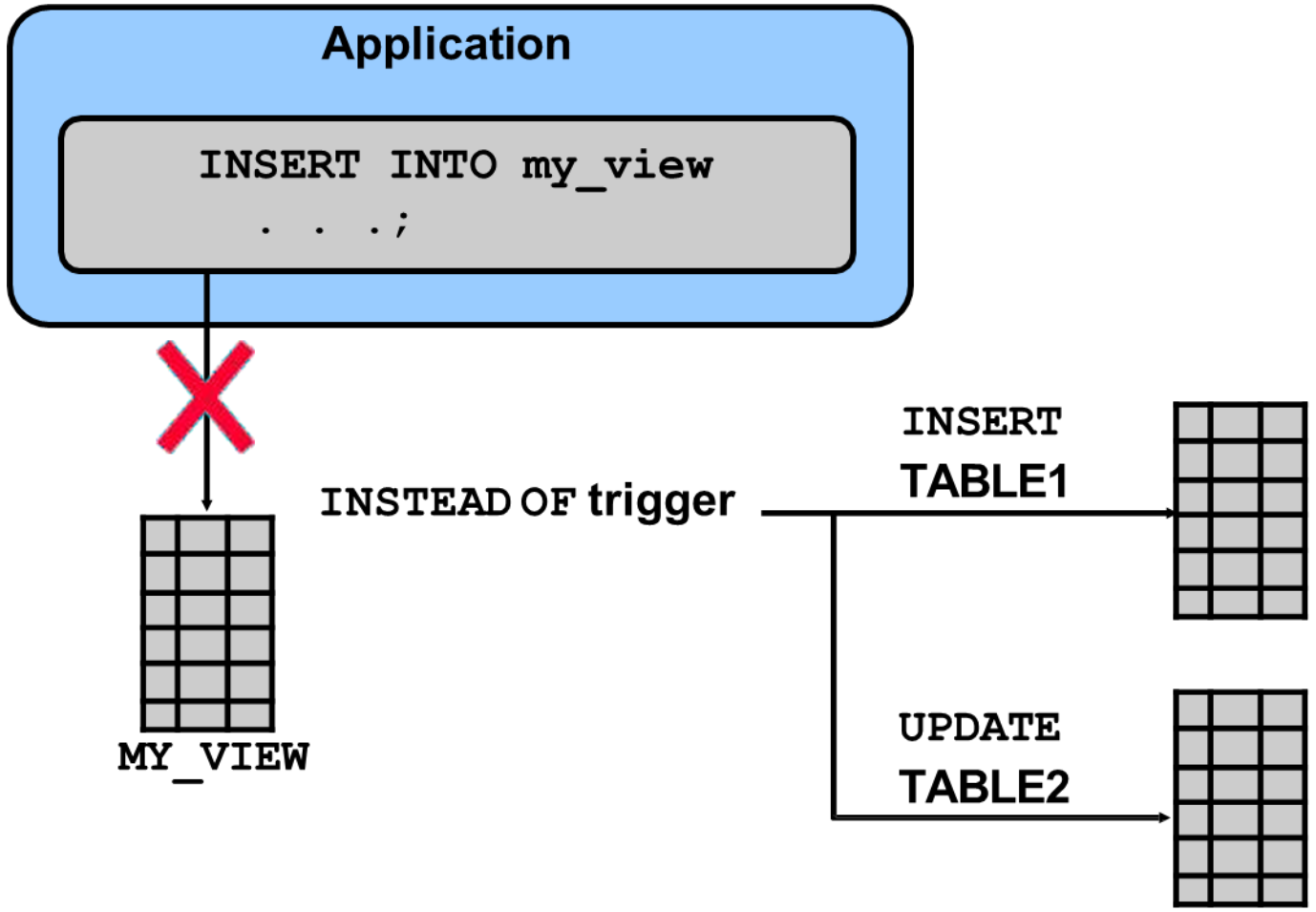
Реалізація обмеження цілісності за допомогою тригера

```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Integrity constraint violation error
```

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg
AFTER UPDATE OF department_id
ON employees FOR EACH ROW
BEGIN
 INSERT INTO departments VALUES(:new.department_id,
           'Dept '||:new.department_id, NULL, NULL);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
    NULL; -- mask exception if department exists
END;
/
```

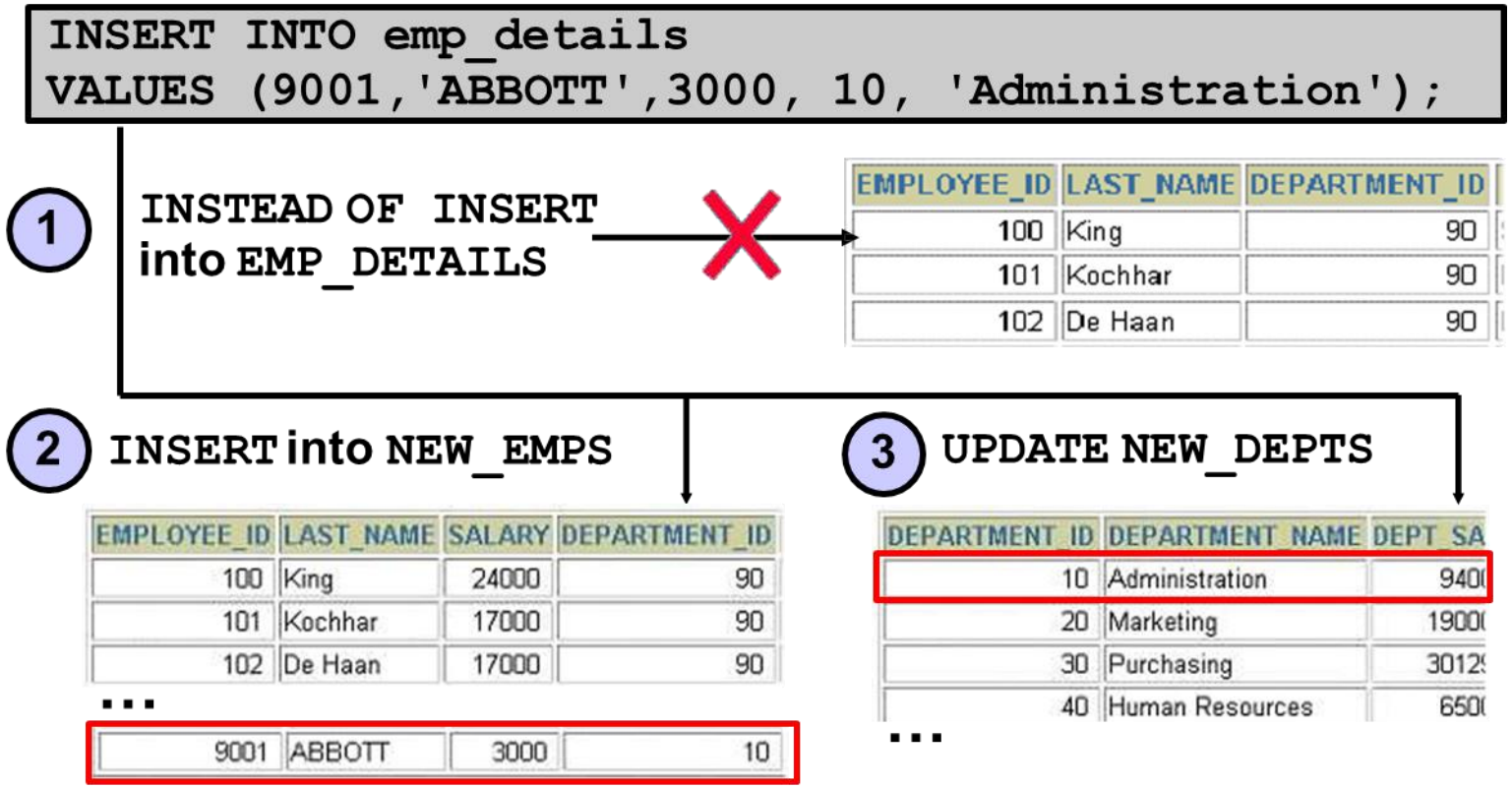
```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Successful after trigger is fired
```

Тригери INSTEAD OF



Створення тригера INSTEAD OF$_1$

- Виконання `INSERT` в `EMP_DETAILS` на основі таблиць `EMPLOYEES` і `DEPARTMENTS`:



Створення тригера INSTEAD OF$_2$

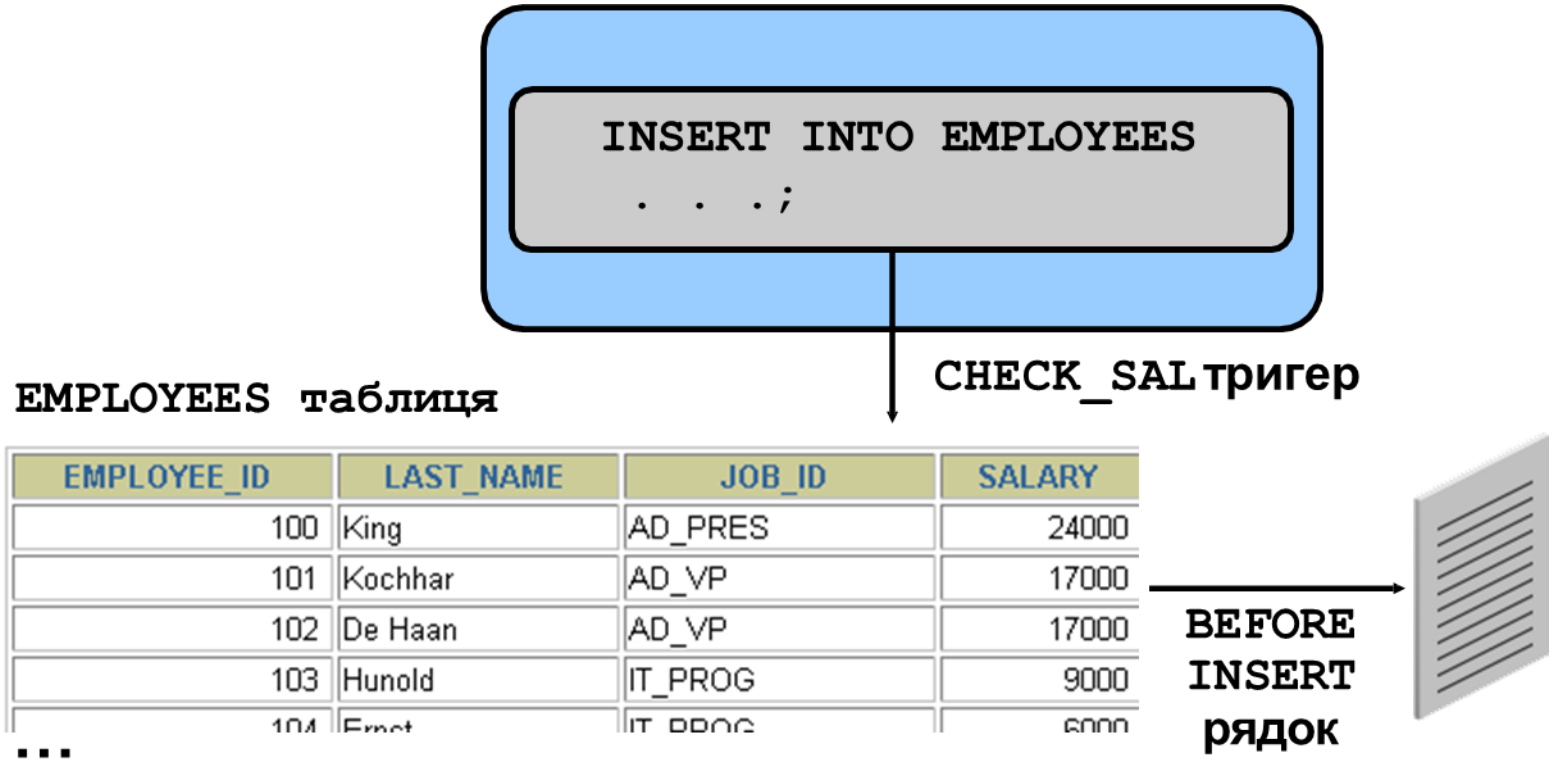
- Використовуйте `INSTEAD OF` для виконання DML на складних представленнях:

```
CREATE TABLE new_emps AS
 SELECT employee_id,last_name,salary,department_id
 FROM employees;

CREATE TABLE new_depts AS
 SELECT d.department_id,d.department_name,
        sum(e.salary) dept_sal
 FROM employees e, departments d
 WHERE e.department_id = d.department_id;

CREATE VIEW emp_details AS
 SELECT e.employee_id, e.last_name, e.salary,
        e.department_id, d.department_name
 FROM employees e, departments d
 WHERE e.department_id = d.department_id
GROUP BY d.department_id,d.department_name;
```

Порівняння тригерів бази даних і збережених процедур

Тригери	Процедури
Визначені CREATE TRIGGER	Визначені CREATE PROCEDURE
Словник даних містить код в USER_TRIGGERS.	Словник даних містить код в USER_SOURCE.
Неявно викликається DML	Викликається явно
COMMIT, SAVEPOINT, та ROLLBACK не дозволені.	COMMIT, SAVEPOINT, та ROLLBACK дозволені.

Порівняння тригерів бази даних і тригерів Oracle Forms



Керування тригерами

- Вимкнення або повторне ввімкнення тригеру бази даних:

```
ALTER TRIGGER trigger_name DISABLE | ENABLE
```

- Вимкнення або повторне ввімкнення усіх тригерів для таблиці :

```
ALTER TABLE table_name DISABLE | ENABLE
  ALL TRIGGERS
```

- Перекомпіляція тригеру для таблиці:

```
ALTER TRIGGER trigger_name COMPILE
```

Видалення тригерів

- Щоб видалити тригер із бази даних слід використати оператор `DROP TRIGGER`:

```
DROP TRIGGER trigger_name;
```

- Приклад:

```
DROP TRIGGER secure_emp;
```

Примітка: під час видалення таблиці всі тригери таблиці видаляються.

Тестування тригерів

- Перевірка кожної активної операцію з даними, а також неактивних операцій з даними.
- Перевірка кожного регістр виразу WHEN.
- Активація тригера безпосередньо з базової операції даних, а також опосередковано з процедури.
- Перевірка впливу тригера на інші тригери.
- Перевірка дії інших тригерів на тригер.

Застосунки для тригерів

Створення тригерів бази даних

- Запуск події користувача:
  - CREATE, ALTER або DROP
  - Вхід або вихід
- Запуск бази даних або системної події:
  - Вимкнення або запуск бази даних
  - Порушення конкретної помилки (або будь-якої помилки).

Створення тригерів на операторах DDL

- Синтаксис:

```
CREATE [OR REPLACE] TRIGGER trigger_name
Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE|SCHEMA}
trigger_body
```

Створення тригерів системних подій

- Синтаксис:

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE|SCHEMA}
trigger_body
```

Тригери LOGON і LOGOFF: приклад


```
CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON  ON  SCHEMA
BEGIN
 INSERT INTO log_trig_table(user_id,log_date,action)
 VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF  ON  SCHEMA
BEGIN
 INSERT INTO log_trig_table(user_id,log_date,action)
 VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

Вираз CALL

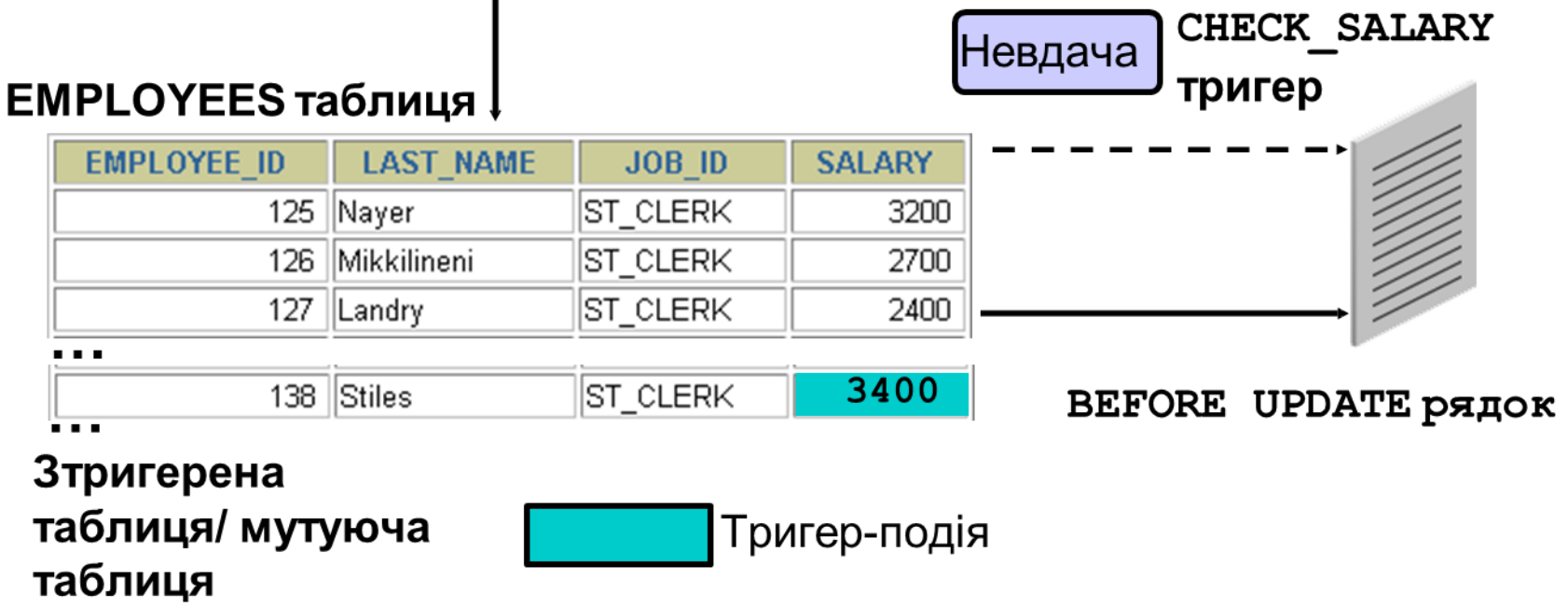
```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
event1 [OR event2 OR event3]
ON table_name
[REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
[WHEN condition]

/
```

```
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES

/
```

Примітка. Крапка з комою в кінці CALL відсутня.

Читання даних із змінюваної таблиці

```
UPDATE employees
  SET salary = 3400
  WHERE last_name = 'Stiles';
```



Змінювана таблиця: приклад$_1$


```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF salary, job_id
  ON employees
  FOR EACH ROW
  WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
  minsalary employees.salary%TYPE;
  maxsalary employees.salary%TYPE;
BEGIN
  SELECT MIN(salary), MAX(salary)
   INTO  minsalary, maxsalary
   FROM  employees
   WHERE job_id = :NEW.job_id;
  IF :NEW.salary < minsalary OR
     :NEW.salary > maxsalary THEN
     RAISE_APPLICATION_ERROR(-20505,'Out of range');
  END IF;
END;
/
```

Змінювана таблиця: приклад$_2$

```
UPDATE employees
 SET salary = 3400
 WHERE last_name = 'Stiles';
```

```
UPDATE employees
       *
ERROR at line 1:
ORA-04091: table PLSQL.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "PLSQL.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'PLSQL.CHECK_SALARY'
```

Переваги тригерів бази даних

- Покращена безпека даних:
  - Забезпечення розширеної та комплексної перевірки безпеки
  - Забезпечення розширеного і комплексного аудиту
- Покращена цілісність даних:
  - Дотримання динамічних обмежень цілісності даних
  - Дотримання складних обмежень цілісності посилань
  - Впевненість у тому, що пов’язані операції неявно виконуються разом

Керування тригерами

- Для керування тригерами потрібні наступні системні привілеї:
  - Привілей `CREATE/ALTER/DROP (ANY) TRIGGER`: дозволяє створювати тригер у будь-якій схемі
  - Привілей `ADMINISTER DATABASE TRIGGER`: дає змогу створити тригер в базі даних
  - Привілей `EXECUTE` (якщо тригер посилається на будь-які об’єкти, яких немає у схемі)

Примітка. Оператори в тілі тригера використовують привілеї власника тригера, а не привілеї користувача, який виконує операцію, яка запускає тригер.

Сценарії бізнес-застосунків для впровадження тригерів

- Можна використовувати тригери для забезпечення:
  - Безпеки
  - Аудиту
  - Цілісності даних
  - Цілісності посилань
  - Реплікації таблиць
  - Автоматичного обчислення отриманих даних
  - Формування журналу подій

Перегляд інформації тригера

- Можна переглянути інформацію про тригер шляхом:
  - Перегляду словника даних USER_OBJECTS: об’єкт інформації
  - Перегляду словника даних USER_TRIGGERS: текст тригера
  - Перегляд условника даних USER_ERRORS: синтаксичні помилки PL/SQL (помилки компіляції) тригера

Використання USER_TRIGGERS

Стовпець	Опис стовпця
TRIGGER_NAME	Назва тригера
TRIGGER_TYPE	Типами є BEFORE, AFTER, INSTEAD OF
TRIGGERING_EVENT	DML операція, що активує тригер
TABLE_NAME	Назва таблиці БД
REFERENCING_NAMES	Назва використана для :OLD та :NEW
WHEN_CLAUSE	when_clause використано
STATUS	Статус тригера
TRIGGER_BODY	Дії, які потрібно виконати

Перерахування коду тригерів

```
SELECT trigger_name, trigger_type, triggering_event,
         table_name, referencing_names,
         status, trigger_body
FROM   user_triggers
WHERE  trigger_name = 'RESTRICT_SALARY';
```

TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_NAME	REFERENCING_NAMES	WHEN_CLAUS	STATUS	TRIGGER_BODY
RESTRICT_SALARY	BEFORE EACH ROW	INSERT OR UPDATE	EMPLOYEES	REFERENCING NEW AS NEW OLD AS OLD		ENABLED	BEGIN IF NOT (:NEW.JOB_ ID IN ('AD_PRES ', 'AD_VP')) AND :NE W.SAL

Донецький національний технічний університет

Тема 10. Розробка програмних застосунків в Oracle APEX. Лекція 16. Вступ до Oracle APEX.

Предмет: Захист РБД та ІС
Розділ: Кібербезпека
Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.

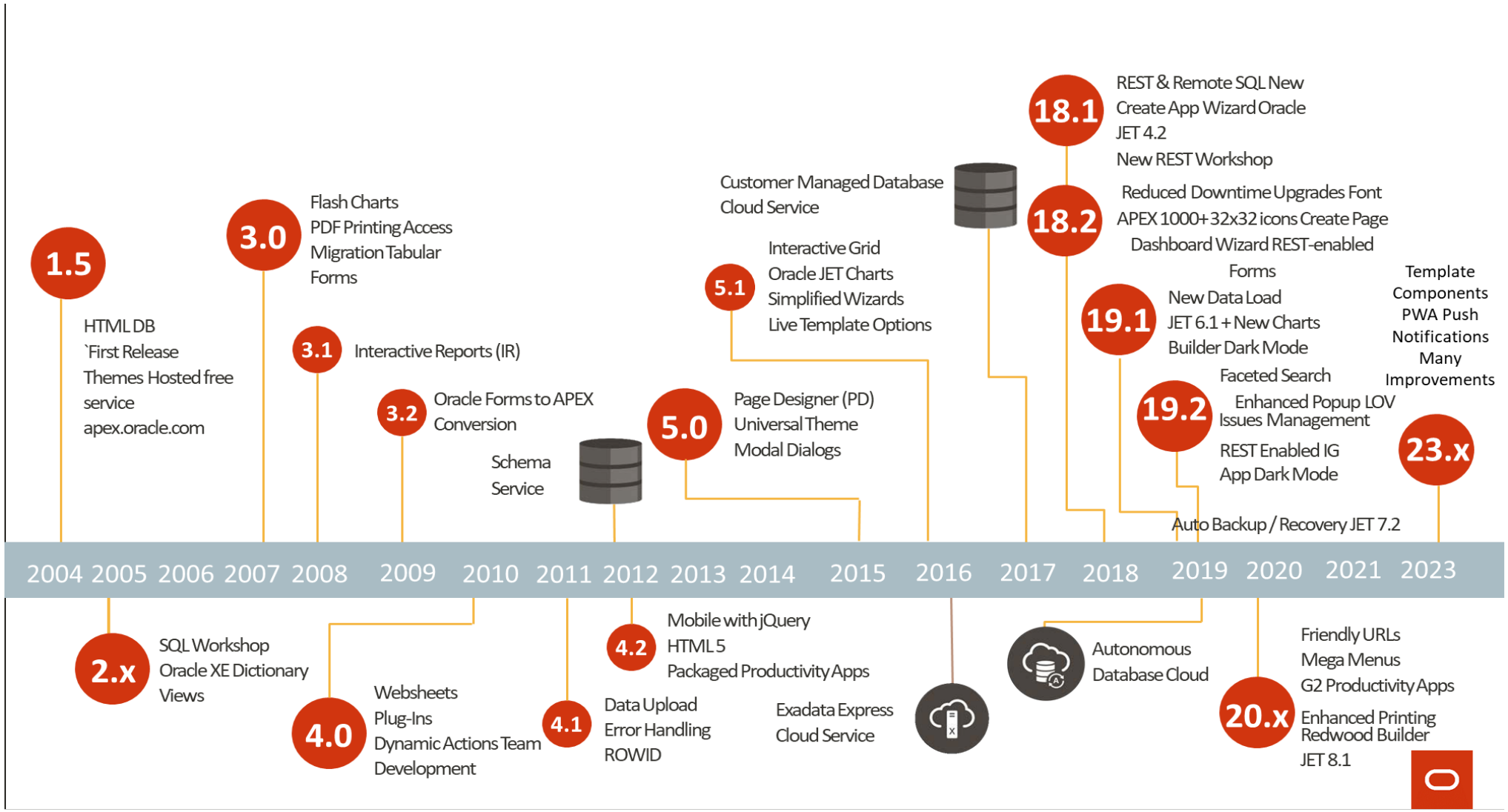
Лекційні питання

- Вступ
- Переваги
- Історія
- Встановлення
- Архітектура
- Створення застосунків
- Практикум SQL
- Конструктор програм
- Компонент «Інтерактивні звіти»
- Компонент «Інтерактивна сітка»
- Компонент створення діаграм
- Динамічні дії

Переваги

- Технологія швидкого розвитку
- Веб-інтерфейс
- Розробники, знайомі з PL/SQL, можуть використовувати той самий набір навичок під час розробки
- Застосунки Apex
- Легко створювати макети
- Простий у розгортанні (кінцевий користувач відкриває URL-адресу для доступу до програми APEX)
- Масштабований (можна розгорнути на ноутбуках, автономних серверах або Oracle RAC)
- Обробка та перевірки на стороні сервера
- Сильна та підтримуюча спільнота користувачів (особливо форум Oracle APEX)
- Базова підтримка розвитку групи
- Безкоштовний хостинг демонстраційних програм, наданий Oracle
- Програми Apex можуть працювати на безкоштовній базі даних Oracle Express Edition (XE).
- Індивідуальний

Історія випусків



Oracle APEX = RAD (Швидка розробка застосунків)

- REST
  - Декларативне створення REST Data Access API з APEX
  - Полегшує інтеграцію та Мікросервіси
  - Легко споживається на будь-якій мові
- APEX
  - Розробка застосунків Low Code
  - IT-професіонали та рядові розробники
  - Рішення LOB Point, SaaS розширення
  - Диференційований, Low Code, сильна спільнота
- База даних
  - Популярність SQL продовжує зростати
  - SQL продовжує приносити найвищу продуктивність
  - SQL легко вивчити, але він приголомшливо потужний

Повне пакетне рішення для бізнес-застосунків, що керуються даними

Oracle APEX – безкоштовний бонус бази даних Oracle

- Безкоштовна повна підтримка функція
  - Будь-яка кількість програм, розробників і кінцевих користувачів
  - Підтримується Oracle 11gR2, 12c, 18c , 19c, 23c
  - Усі версії БД: EE, SE2, XE (безкоштовно видання)
- Входить до складу Oracle Cloud Services
  - Безкоштовний сервіс http://apex.oracle.com
  - Exadata Express Service and Database як а Сервіс
  - http://cloud.oracle.com/database
- Легко встановити
  - За замовчуванням включено до всіх версій Oracle бази даних
  - Завантаження останньої версії з https://apex.oracle.com/otn

Oracle APEX – фреймворк розробки веб-застосунків, орієнтований на бази даних

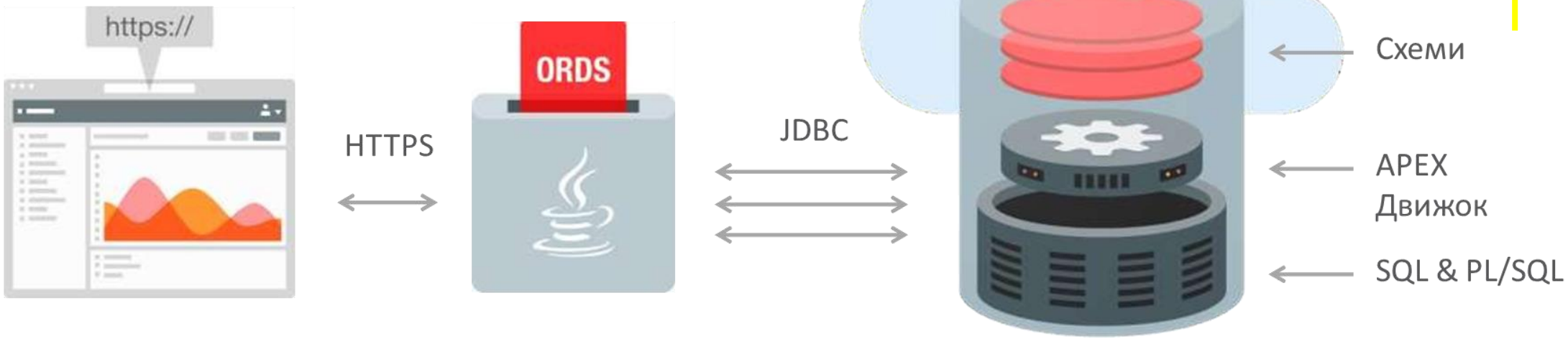
Розробка адаптивних красивих веб-застосунків

Візуалізація та підтримка бази даних даних

Використання SQL-навичок та можливостей бази даних

Oracle APEX використовується для створення адаптивних застосунків для мобільних і настільних комп’ютерів. Можна використовувати потужні звіти та діаграми для візуалізації та керування даними. Oracle APEX орієнтований на SQL.

Oracle APEX - Архітектура



Oracle REST Data Services

(Weblogic, Jetty, Tomcat)

Немає програмної логіки

Перетворює HTTP у виклики API бази даних

Oracle Database

(Pluggable або Dedicated, 11*g*, 12*c*, 18*c*, 23*c*)

Доступ до даних бази даних без затримки

Динамічно керується метаданими APEX

Browser | Mid Tier | Database Tier

Проста архітектура, у якій запити сторінок і подання, зроблені з браузера, тунелюються через проміжний рівень для виконання в базі даних Oracle і повертаються як HTML-відповіді в браузер. Жодна маніпуляція даними або обробка не виконується на середньому рівні, натомість механізм APEX (всередині БД Oracle) приймає сторінку та взаємодіє зі схемами даних у БД.

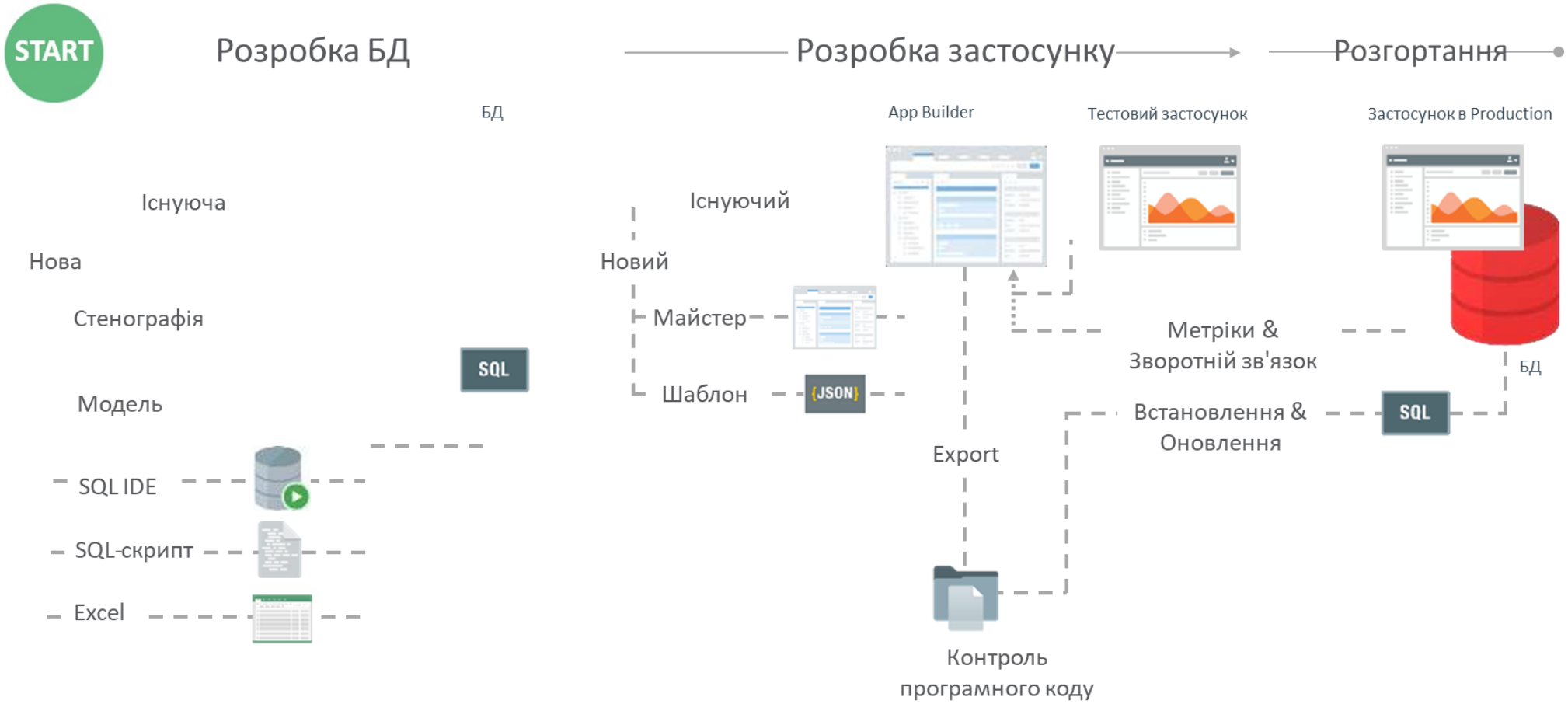
Oracle APEX

Швидка розробка, налаштування та доставка

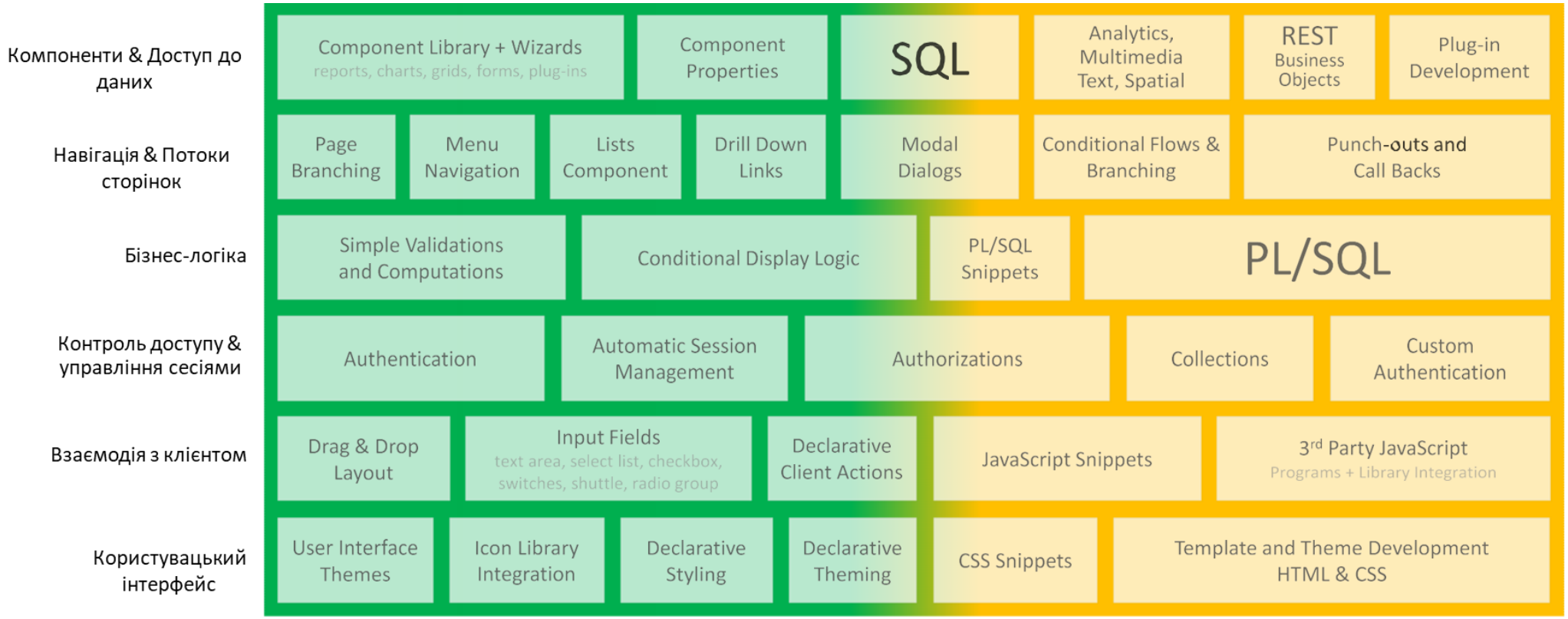
- Перехід від прототипу до впровадження в виробництво за хвилини

В основі Application Express лежить механізм, який забезпечує низку фундаментальних можливостей і операцій програми. Автентифікація програми, контроль доступу на рівні сторінки/об’єкта, взаємодія з базою даних (запити/оновлення тощо), перевірка форм, керування та захист сеансів тощо доступні як стандартні компоненти, які можна використовувати з будь-якої програми без індивідуальної розробки. Об’єкти застосунків, такі як форми, звіти, діаграми, навігація, визначаються декларативно, що дозволяє застосункам бути функціонально завершеними за короткий проміжок часу, підвищуючи гнучкість процесу самої розробки.

Oracle APEX
Розробка першого застосунку Low Code



Оракул APEX
Високопродуктивні компоненти



No Code

Висока продуктивність
Бізнес-користувачі
4GL властивості метаданих

Low Code

Більше контролю
Рядові розробники
Сніпети та плагіни

Programmatic

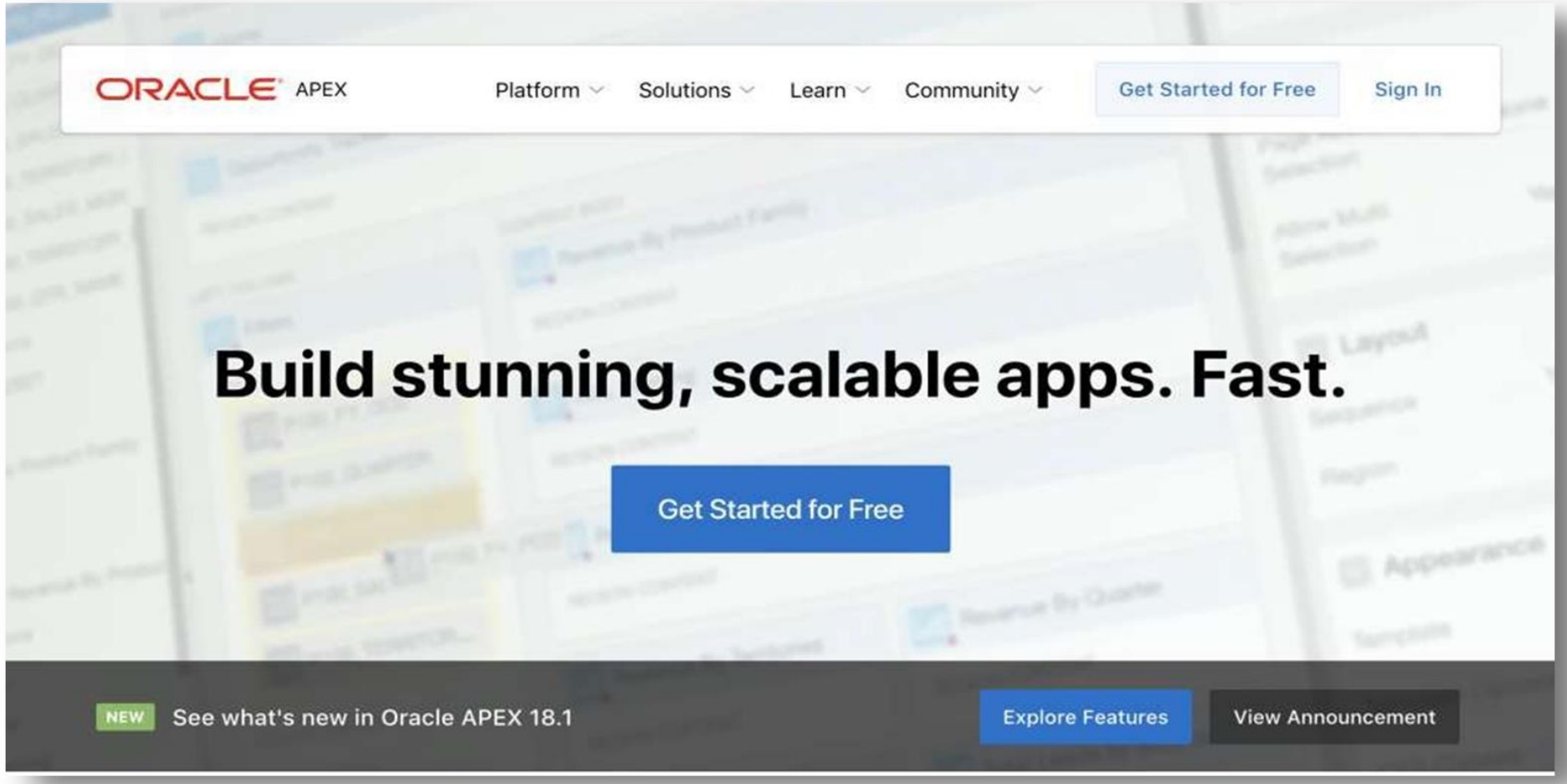
Високий контроль
IT Професіонали
3GL процедурний код/
Бібліотеки

Oracle APEX – створення застосунків

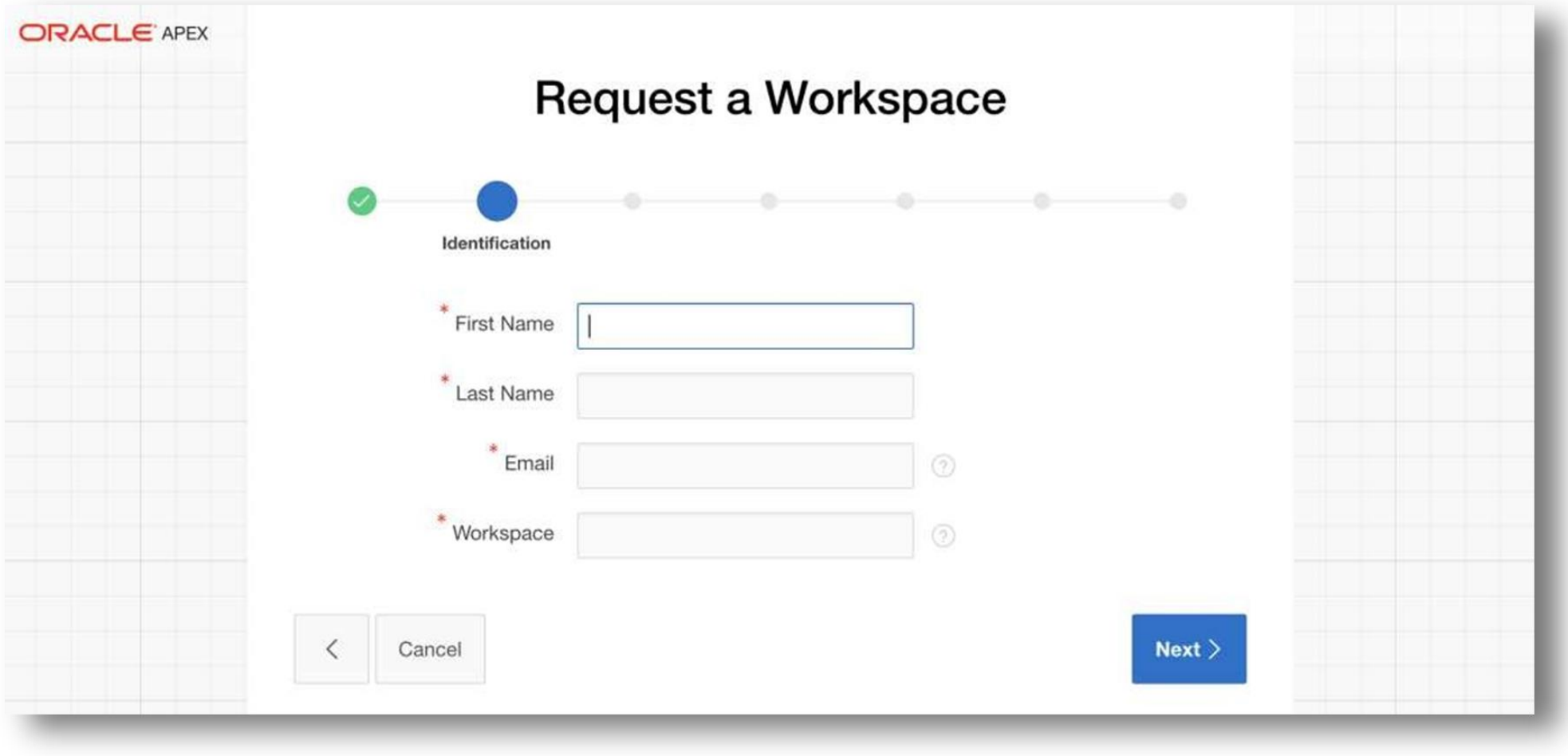
- Перейти до https://apex.oracle.com
- Натиснути Увійти та надіслати запит на створення Workspace
- В листі на електронній пошті натиснути Створити Workspace
- Натиснути SQL Workshop > Sample Datasets для завантаження тестових зразків даних
- Натиснути Create Application
- Заповнити форму та натиснути Create Application
- Натиснути Run Application
- Поширити URL-адресу для обміну з іншими партнерами

Workspace Request Approved

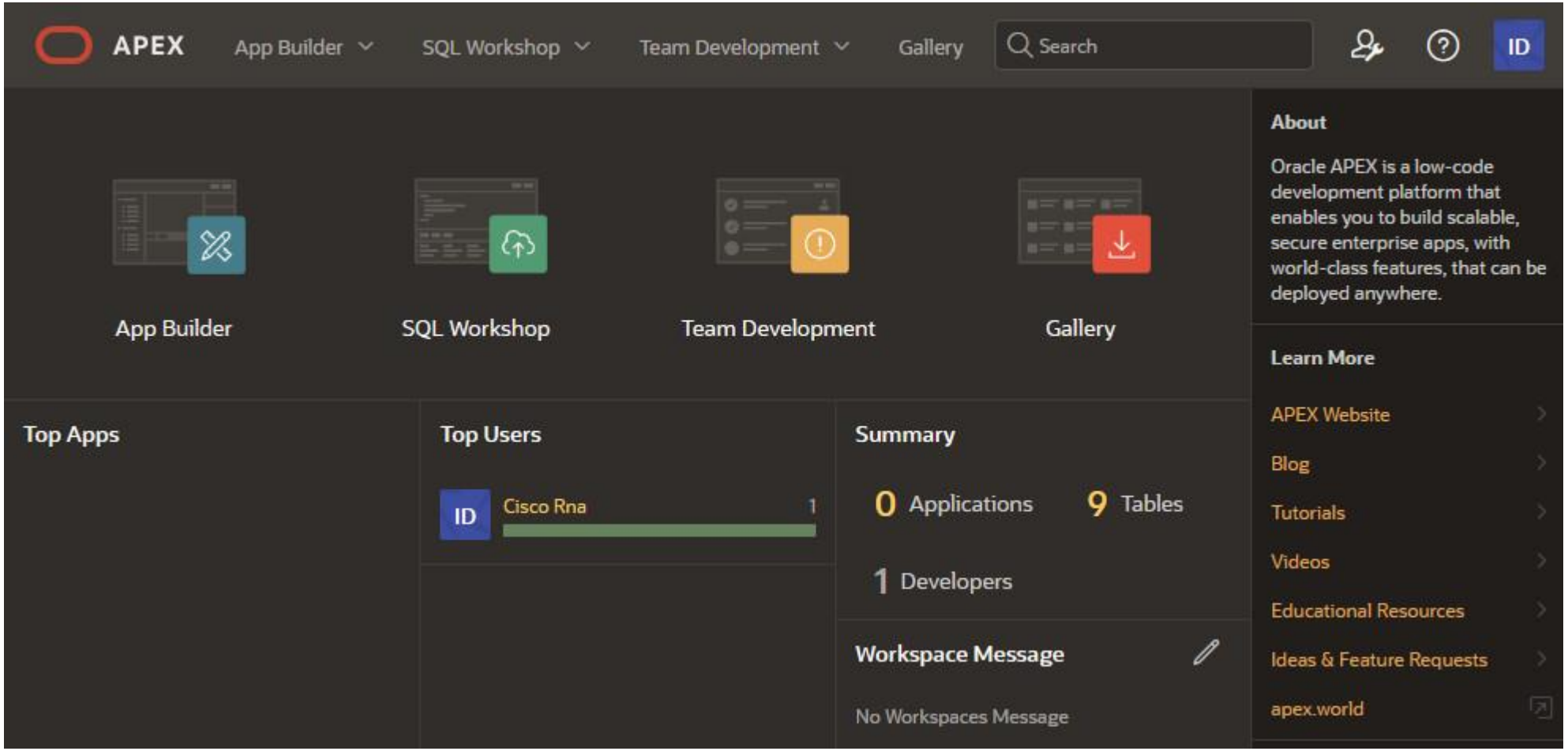
apex.oracle.com



apex.oracle.com

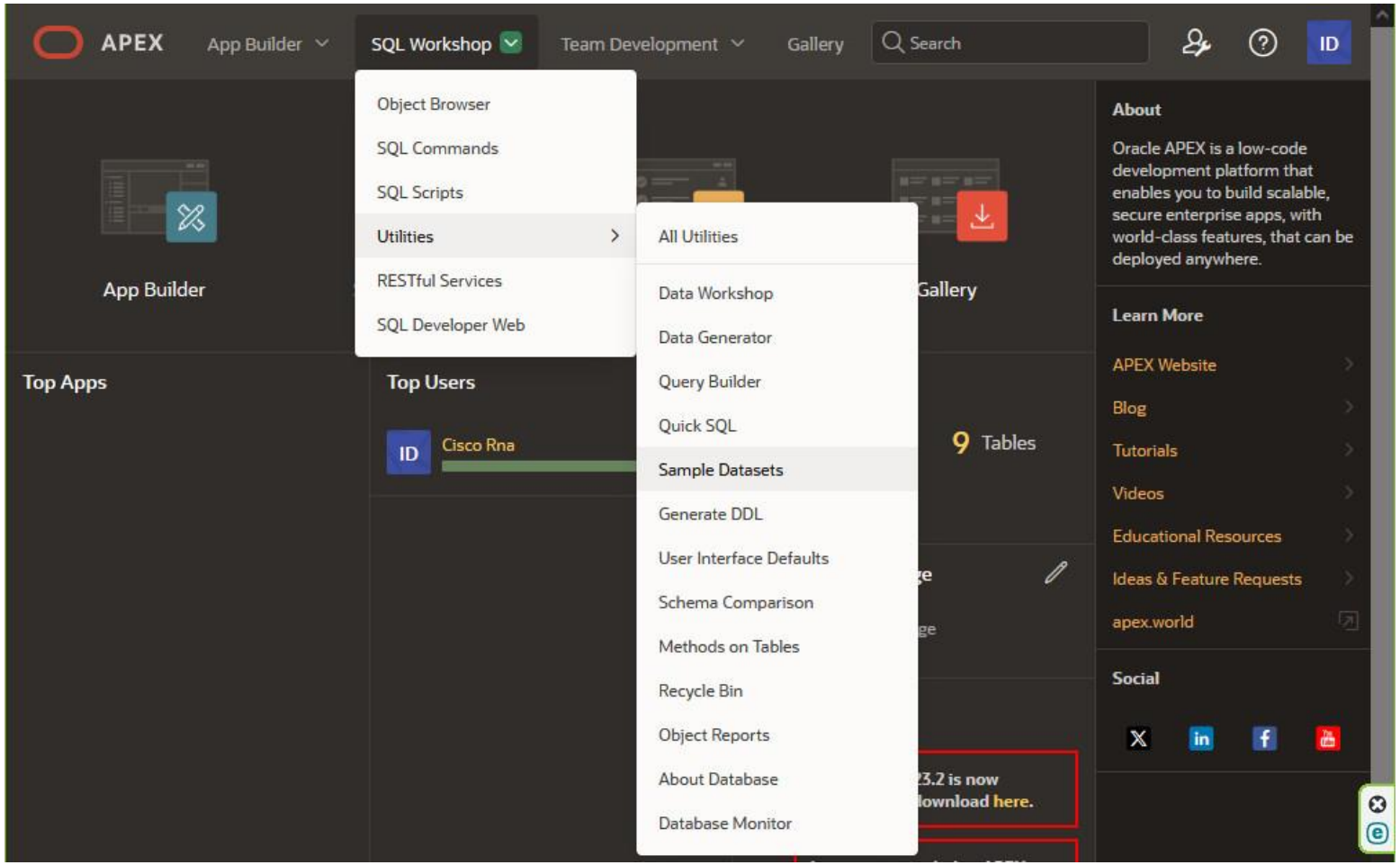


Створення застосунку APEX



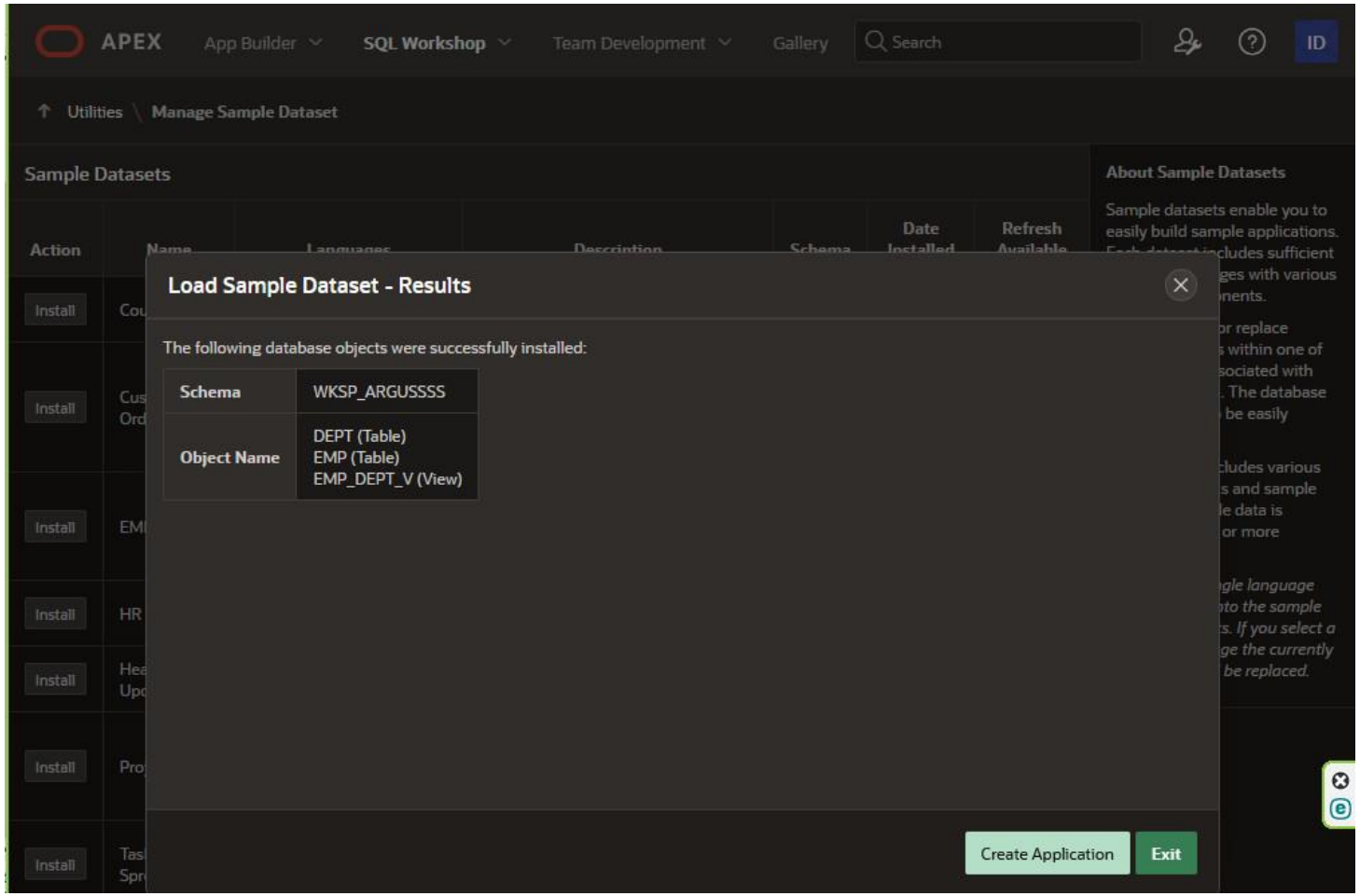
Створення застосунку APEX

Створення тестових зразків даних

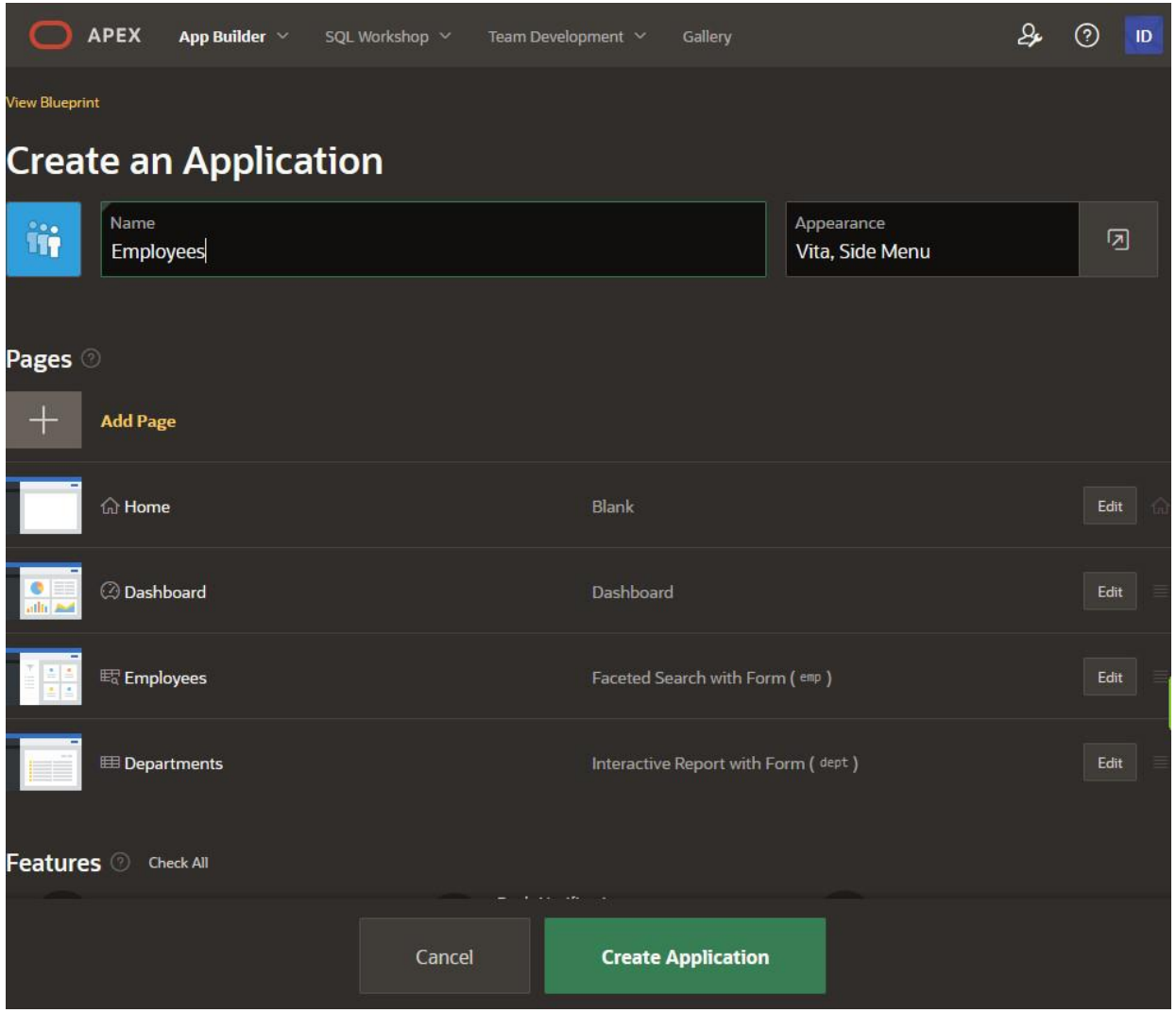


Створення застосунку APEX

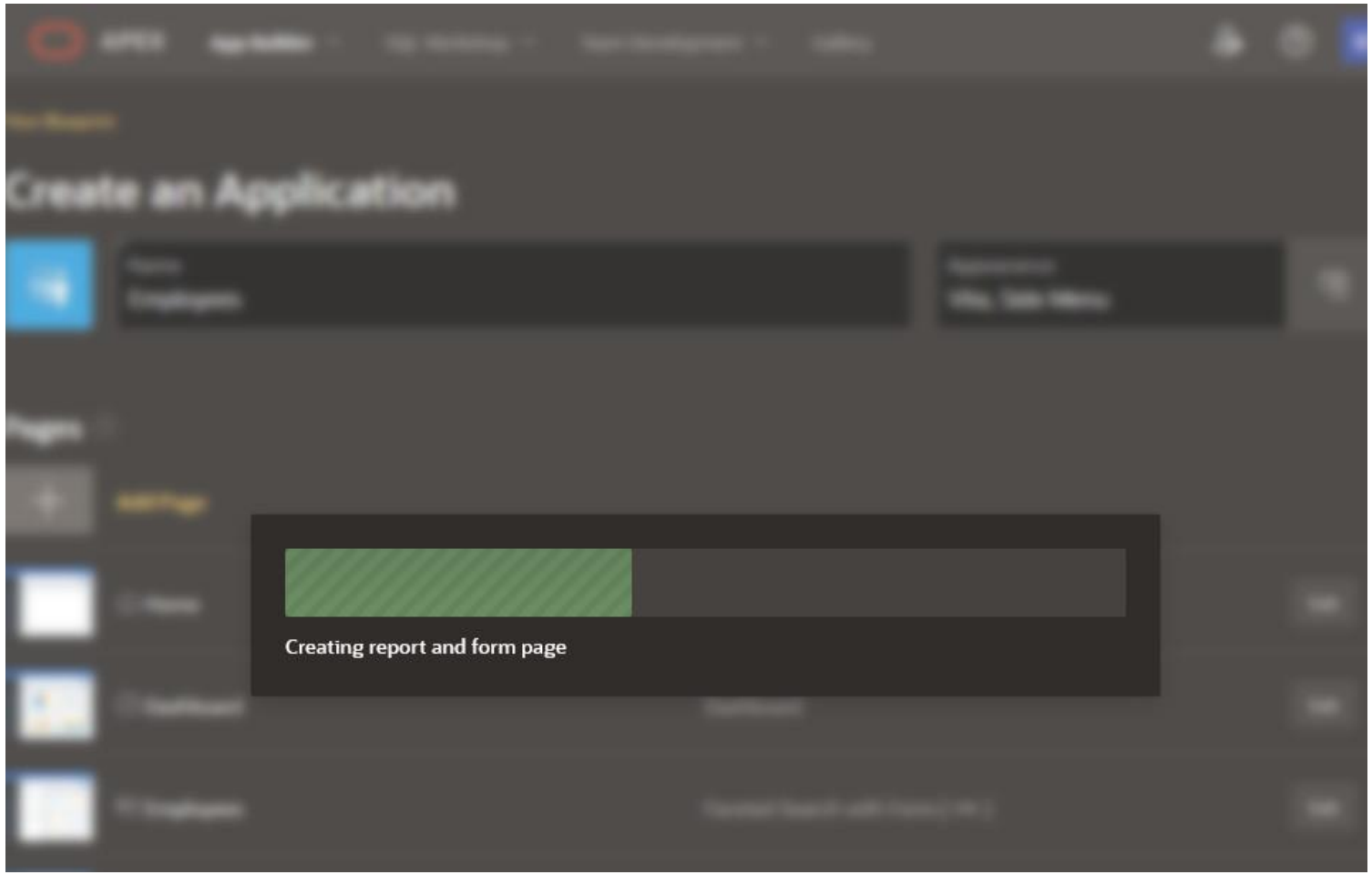
Create Application



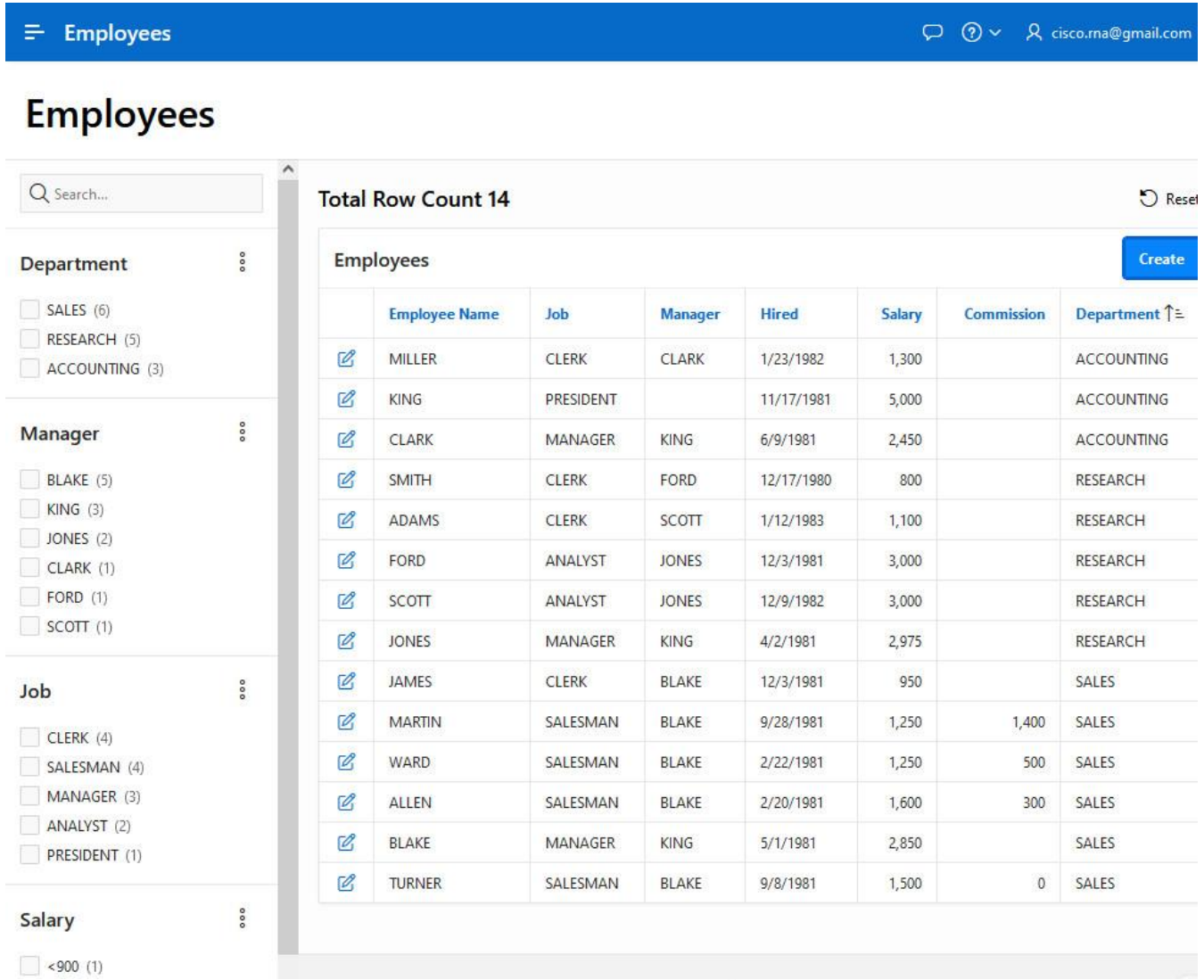
Приклад застосунку APEX$_1$



Приклад застосунку APEX$_2$

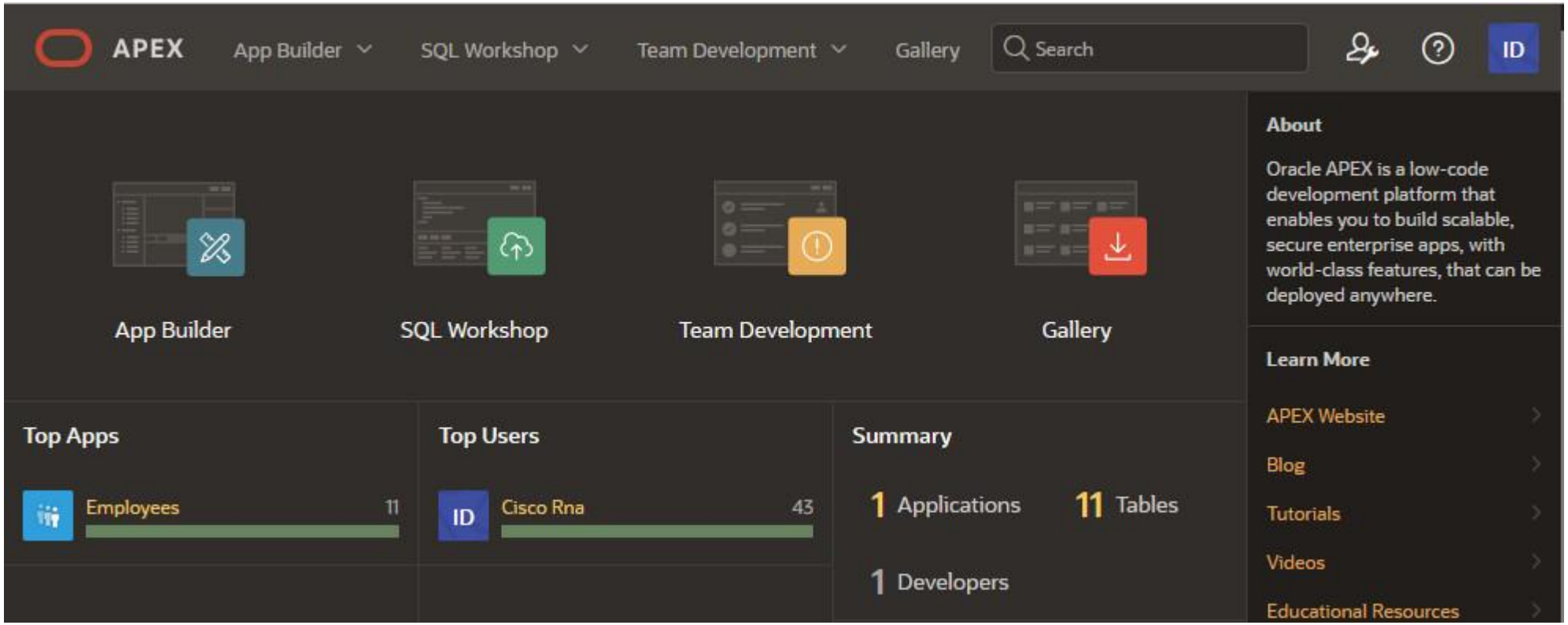


Приклад застосунку APEX$_3$



Oracle APEX

Інтегроване середовище розробки в браузері

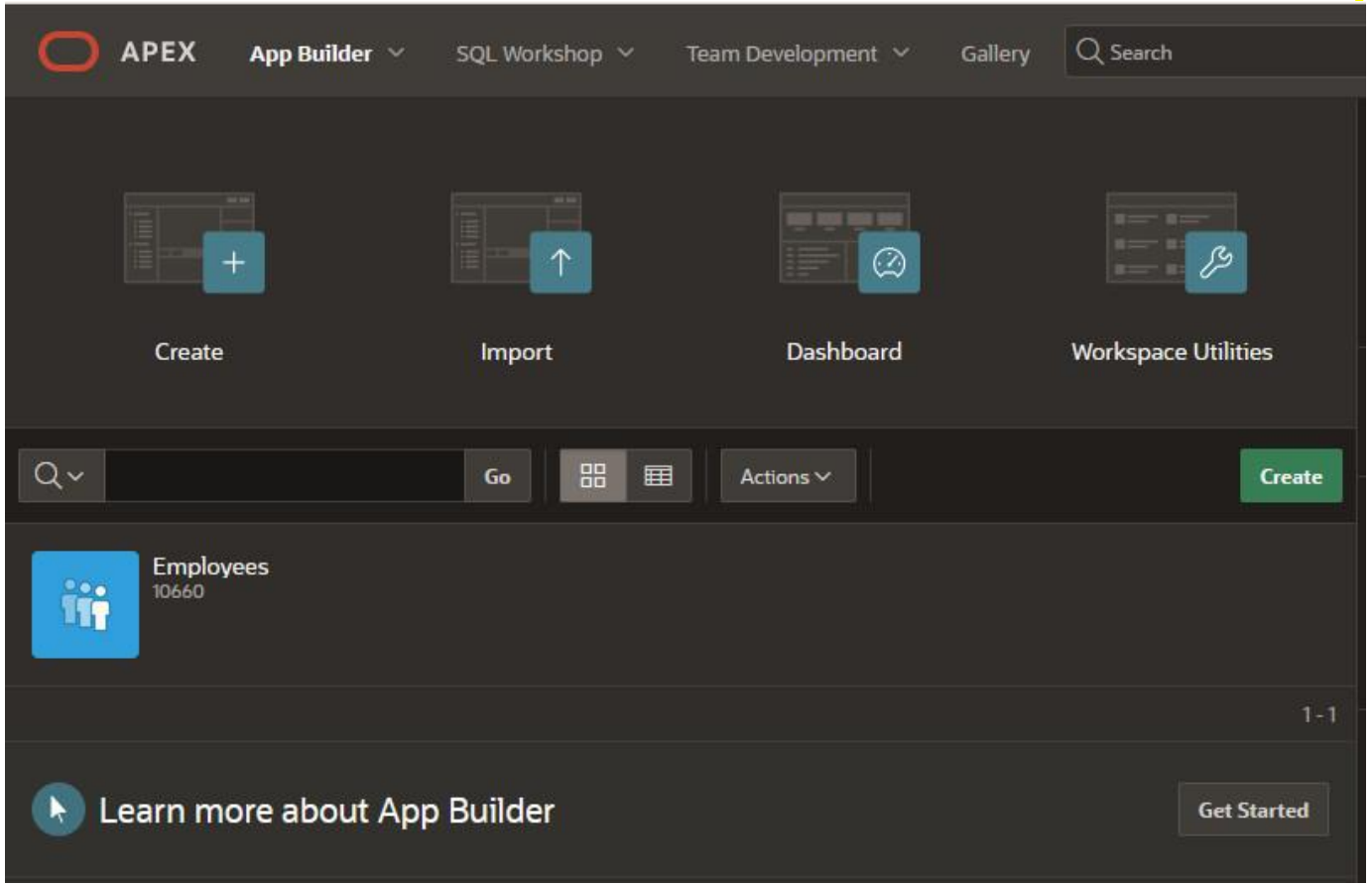


- SQL WorkShop
- Application Builder
- Багатокористувацький
- Декларативний
- Зразки готових застосунків

Oracle APEX

Application Builder

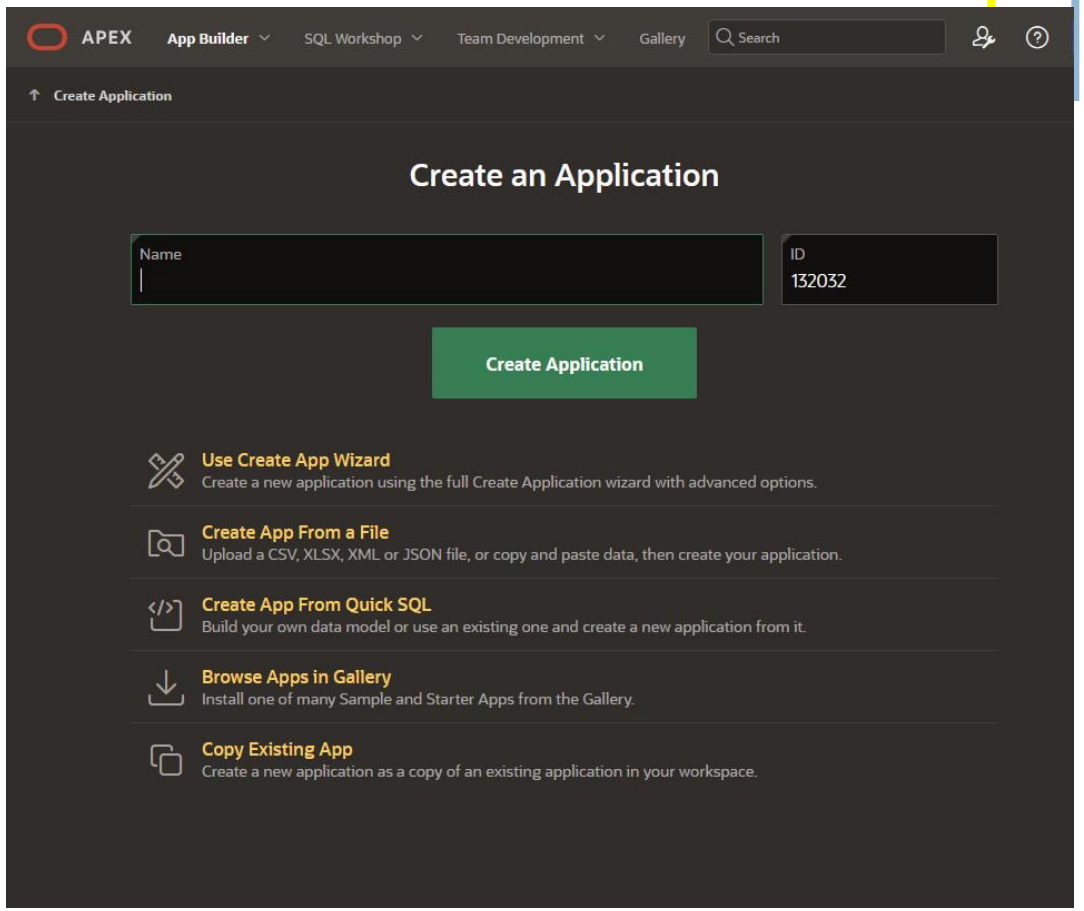
- Створення застосунків
- Розвиток існуючих застосунків
- Експорт застосунку в файл
- Імпорт застосунку з файл
- Перегляд інформаційної панелі



Oracle APEX

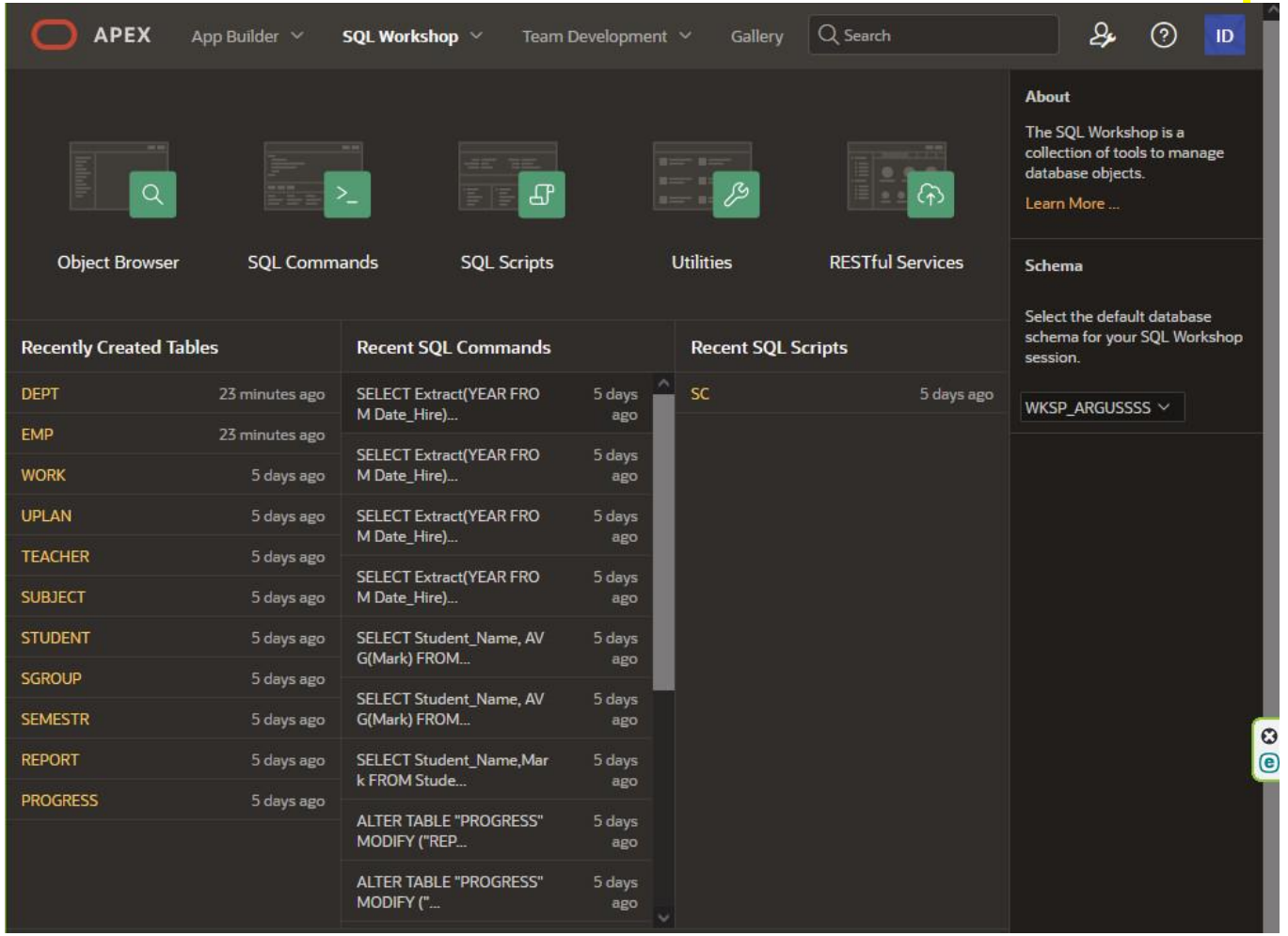
Low Code Application Builder

- Простіші та модернізовані майстри для створення сторінки
- Дозволяє створювати більш розширені сторінки, такі як Dashboards, Master-Detail, тощо
- Підтримує додавання загальних фреймворків або «Функцій» під час створення програми, наприклад контроль доступу, звіти про діяльність, вибір теми та інші
- Налаштування таких параметрів інтерфейсу користувача, як теми, значок програми та піктограми сторінки



Oracle APEX
SQL WorkShop

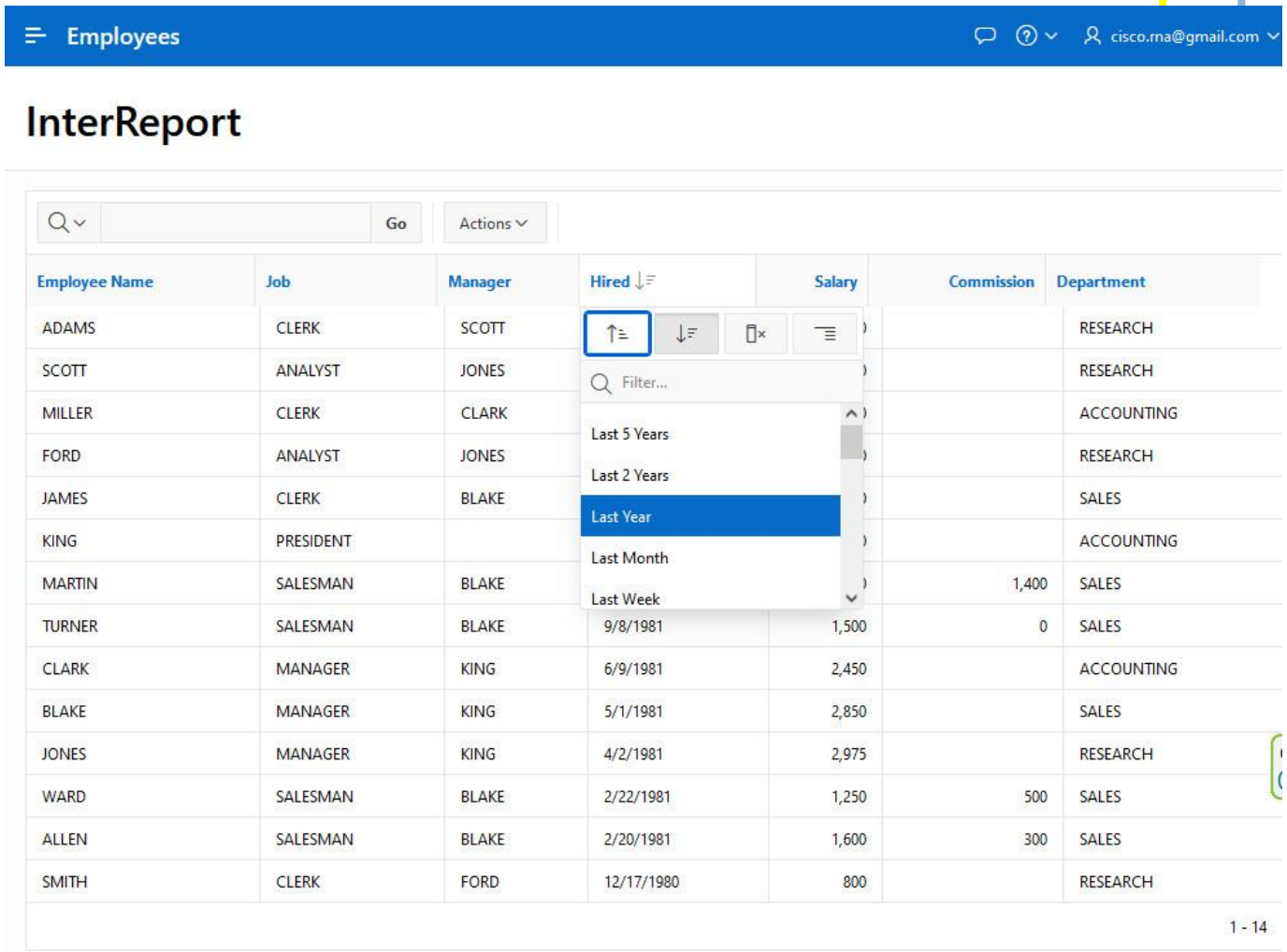
- SQL команди
- SQL сценарії
- Браузер об'єктів
- Завантаження даних
- Зразки даних
- Сервіси REST
- Керування своїми об’єктами бази даних і створення застосунків за допомогою одного інструменту



Oracle APEX

Компонент «Інтерактивні звіти»

- Керований SQL
- Потужний засіб звітності
- Фільтри
- Контрольні перерви
- Опорні точки
- Створення діаграм
- Вибіркове відображення
- Скачування
- Автоматизована електронна пошта

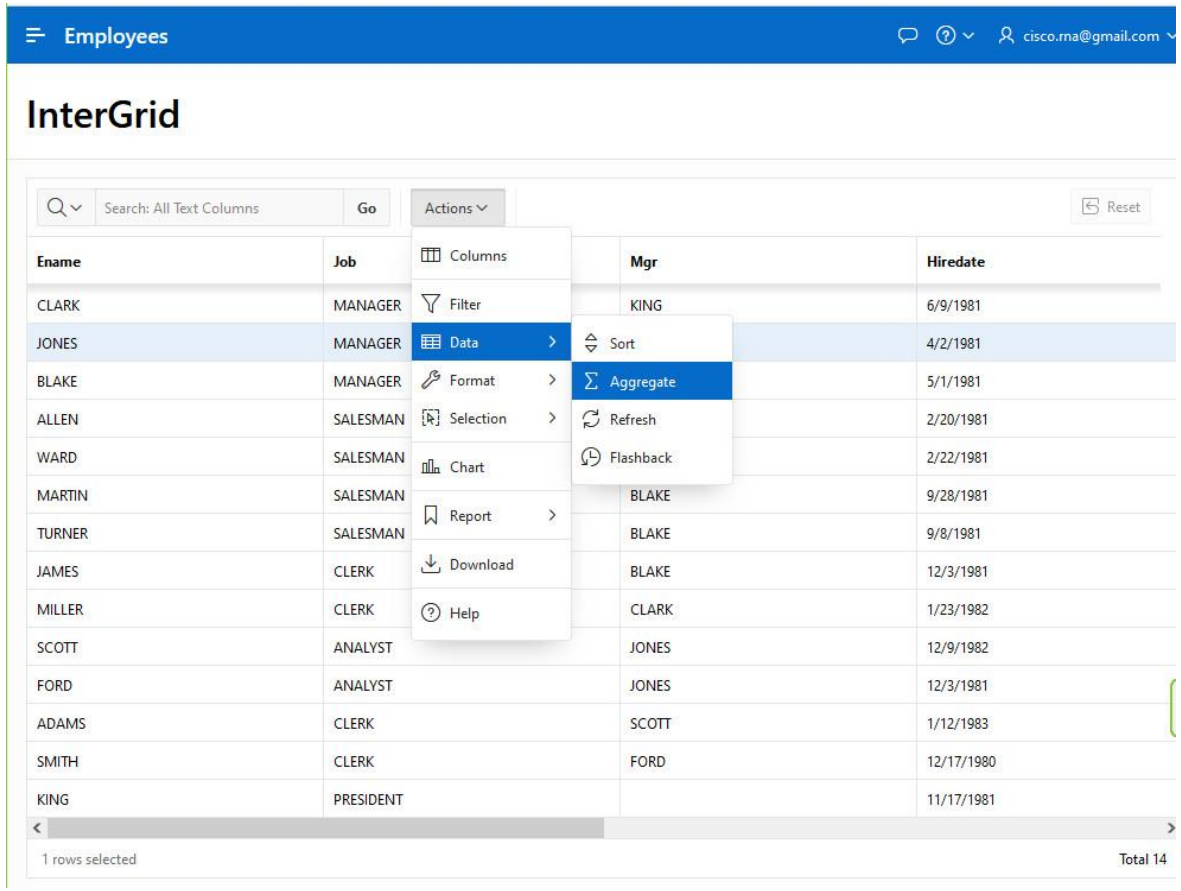


Employee Name	Job	Manager	Hired	Salary	Commission	Department
ADAMS	CLERK	SCOTT				RESEARCH
SCOTT	ANALYST	JONES				RESEARCH
MILLER	CLERK	CLARK				ACCOUNTING
FORD	ANALYST	JONES				RESEARCH
JAMES	CLERK	BLAKE				SALES
KING	PRESIDENT					ACCOUNTING
MARTIN	SALESMAN	BLAKE			1,400	SALES
TURNER	SALESMAN	BLAKE	9/8/1981	1,500	0	SALES
CLARK	MANAGER	KING	6/9/1981	2,450		ACCOUNTING
BLAKE	MANAGER	KING	5/1/1981	2,850		SALES
JONES	MANAGER	KING	4/2/1981	2,975		RESEARCH
WARD	SALESMAN	BLAKE	2/22/1981	1,250	500	SALES
ALLEN	SALESMAN	BLAKE	2/20/1981	1,600	300	SALES
SMITH	CLERK	FORD	12/17/1980	800		RESEARCH

Oracle APEX

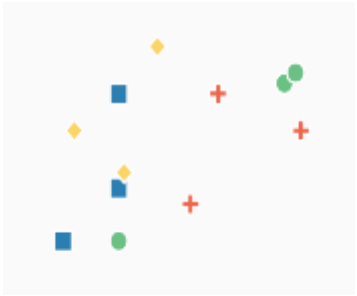
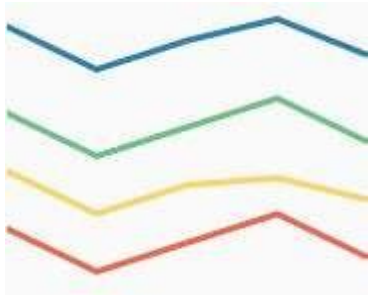
Компонент «Інтерактивна сітка»

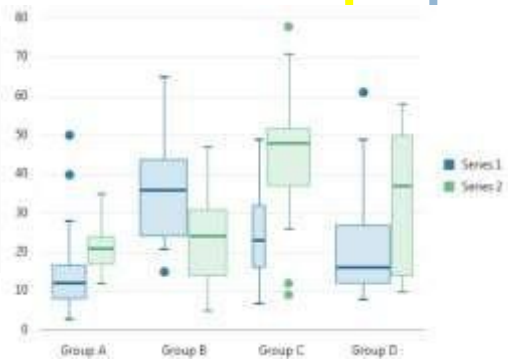
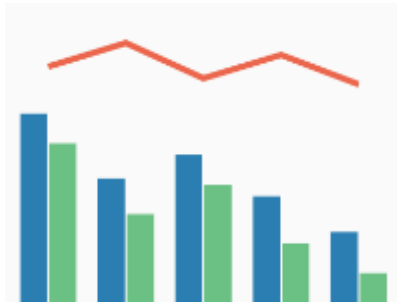
- Сучасний, насичений та інтерактивний багаторядковий компонент для редагування
- Декларативна підтримка для Cascading LOV і динамічних дій
- Утиліта оновлення для табличних форм

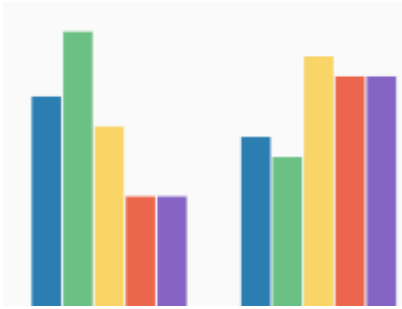


Oracle APEX

Компонент створення діаграм



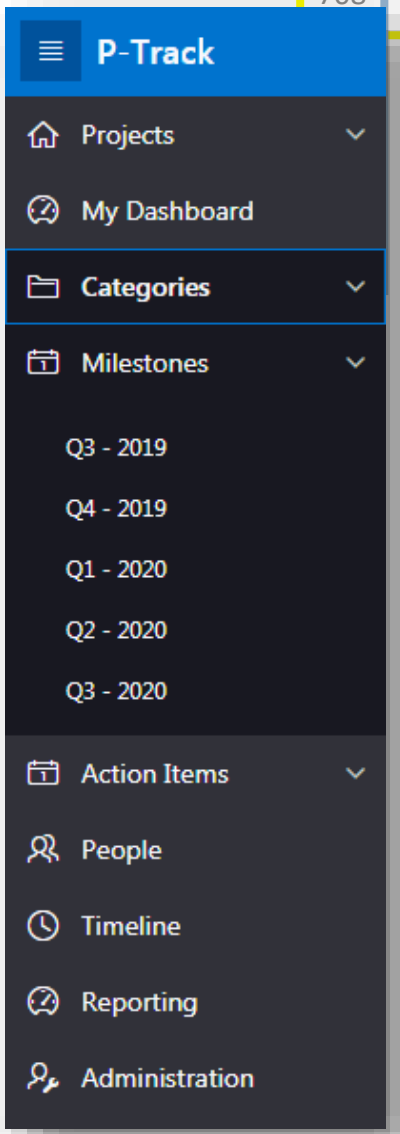





Oracle APEX

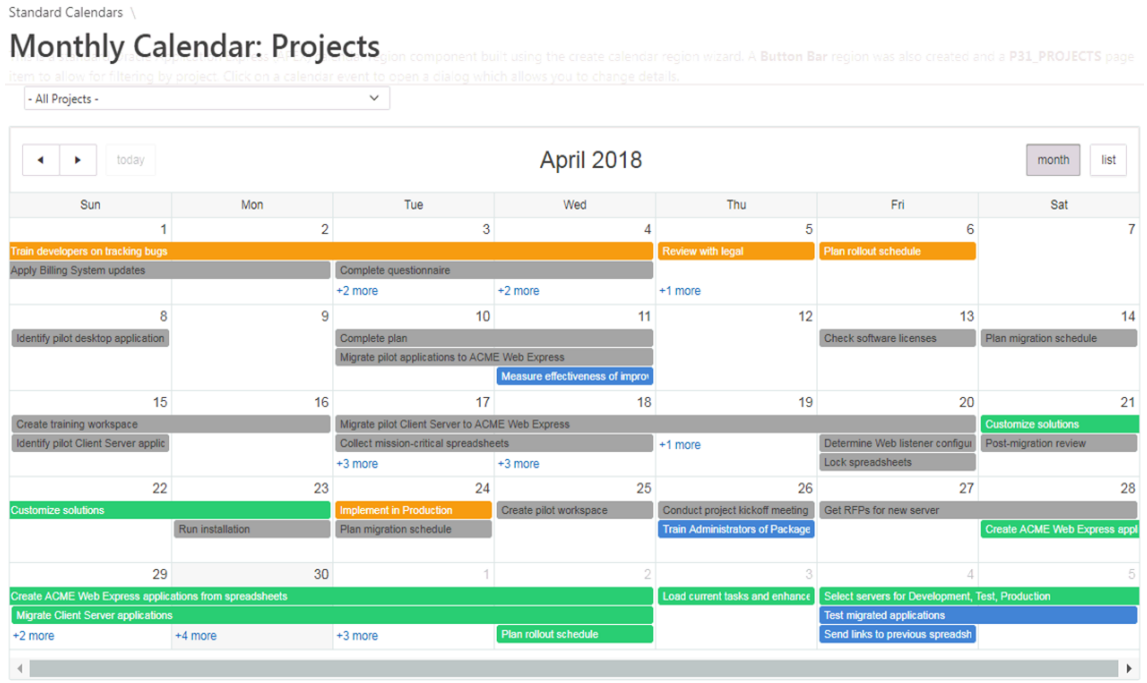
Меню навігації на основі списків

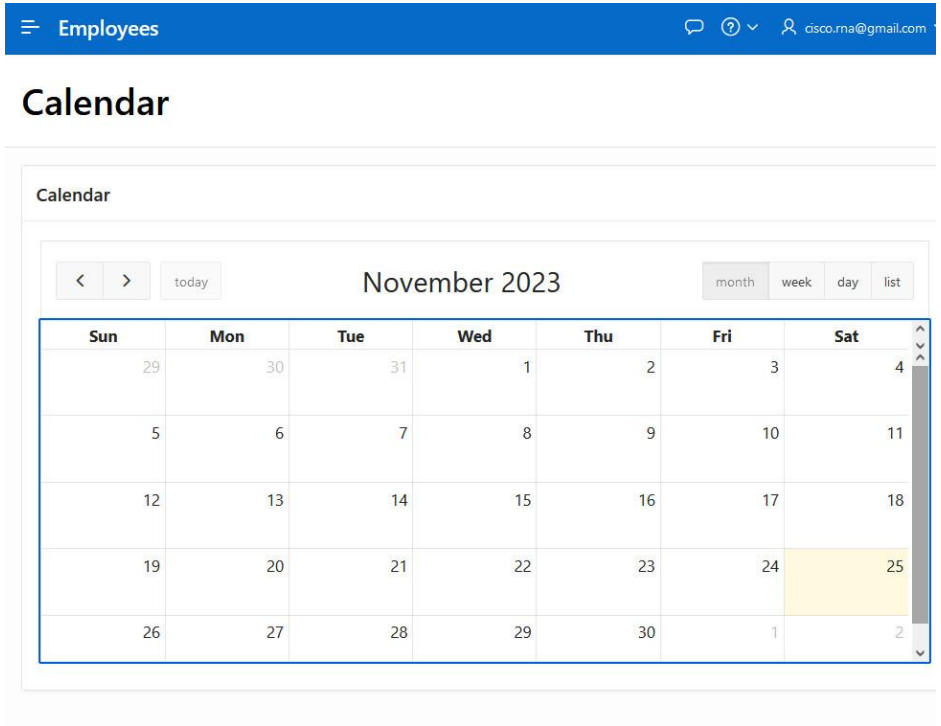
- Альтернатива використанню традиційних вкладок
- Доступне як верхнє навігаційне меню, так і бічні меню
- Реалізоване як стандартні списки APEX
- Підтримує багаторівневу ієрархічну структуру меню
- Забезпечує доступні випадні меню



Oracle APEX

Календарі з SQL

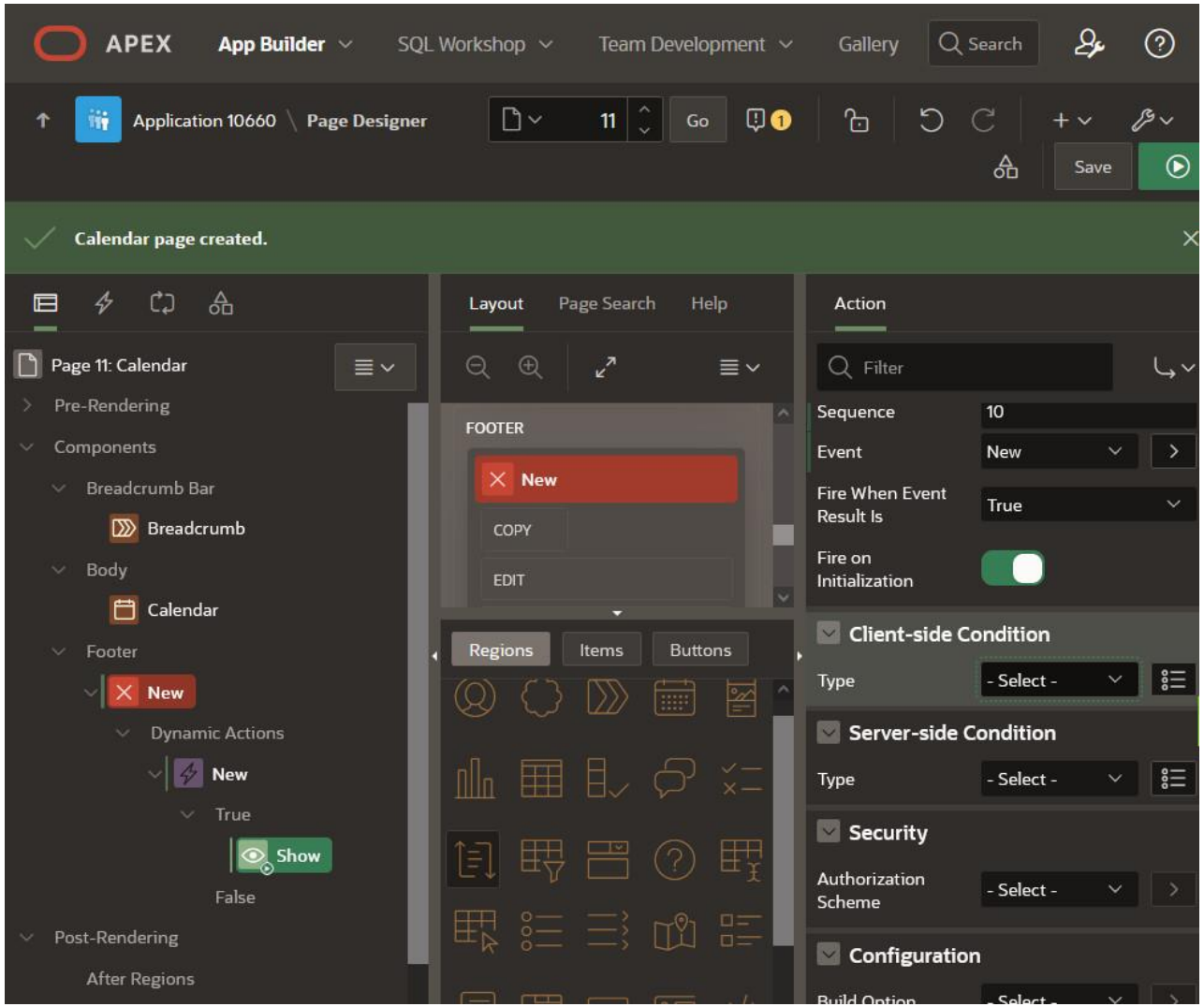




Oracle APEX

Динамічні дії

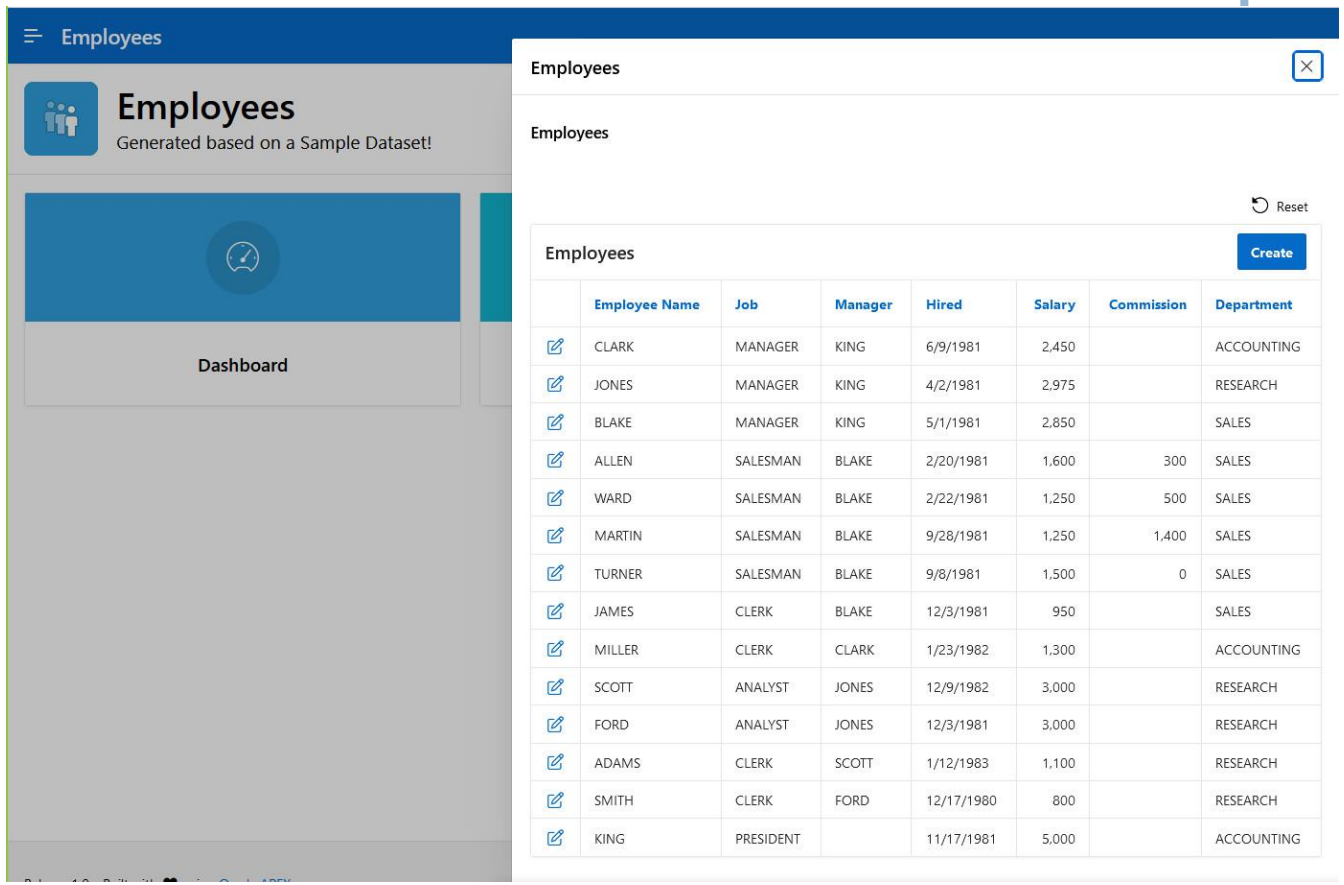
- Декларативно визначає розширену інтерактивність на стороні клієнта без написання JavaScript або AJAX
- Виконує часткове оновлення сторінки
- Можливість показати/приховати компоненти
- Можливість надіслати (запушити) стан на сервер
- Можливість обчислювати значення



Oracle APEX

Модальні діалоги

- Легко перемикатися між режимами звичайної, модальної та немодальної сторінки
- Модальне діалогове вікно — це окрема сторінка, а не область на сторінці
- Будь-яка сторінка може бути створена як діалогова сторінка
- Підтримує всі функції звичайної сторінки, включаючи обчислення, перевірки, процеси та гілки



	Employee Name	Job	Manager	Hired	Salary	Commission	Department
	CLARK	MANAGER	KING	6/9/1981	2,450		ACCOUNTING
	JONES	MANAGER	KING	4/2/1981	2,975		RESEARCH
	BLAKE	MANAGER	KING	5/1/1981	2,850		SALES
	ALLEN	SALESMAN	BLAKE	2/20/1981	1,600	300	SALES
	WARD	SALESMAN	BLAKE	2/22/1981	1,250	500	SALES
	MARTIN	SALESMAN	BLAKE	9/28/1981	1,250	1,400	SALES
	TURNER	SALESMAN	BLAKE	9/8/1981	1,500	0	SALES
	JAMES	CLERK	BLAKE	12/3/1981	950		SALES
	MILLER	CLERK	CLARK	1/23/1982	1,300		ACCOUNTING
	SCOTT	ANALYST	JONES	12/9/1982	3,000		RESEARCH
	FORD	ANALYST	JONES	12/3/1981	3,000		RESEARCH
	ADAMS	CLERK	SCOTT	1/12/1983	1,100		RESEARCH
	SMITH	CLERK	FORD	12/17/1980	800		RESEARCH
	KING	PRESIDENT		11/17/1981	5,000		ACCOUNTING

Oracle APEX
Плагіни

- Розширюйте додатки користувацькими компонентами, такими як елементи та регіони

712

Список рекомендованих джерел для самопідготовки$_1$

1. Rist, Oliver. "Getting Started with VirtualBox." Packt Publishing, 2018. (190 pages)

2. Preece, Robert. "VirtualBox 6.0: Beginners Guide on how to Install, Configure, and Use VirtualBox." Independently published, 2019. (70 pages)

3. Ragar, Victor. "Mastering VirtualBox: Expert techniques for Linux and Windows virtualization, 2nd Edition." Packt Publishing, 2016. (342 pages)

4. Russell, Dusty. "Oracle VM VirtualBox: Beginner's Guide." Packt Publishing, 2018. (284 pages)

5. Russell, Dusty. "Oracle VM VirtualBox 4.0 on OS X Snow Leopard." CreateSpace Independent Publishing Platform, 2010. (128 pages)

6. Herter, Nicholas. "VirtualBox 6.0 – The Ultimate Beginner's Guide." Independently published, 2019. (97 pages)

7. Silverman, Steven. "Oracle VM VirtualBox: An Ultimate Guide Book on Oracle Virtualization Essentials." Independently published, 2018. (87 pages)

Список рекомендованих джерел для самопідготовки$_2$

8. Oracle Database 23c Documentation: Офіційна документація Ora-cle Database 23c, яка включає інструкції з встановлення та конфігурації бази даних. – Режим доступу: https://docs.oracle.com/en/database/ (дата звернення: 12.10.2023 р.).

9. Oracle APEX Documentation: Документація Oracle Application Express (APEX), яка містить відомості про встановлення та розгортання APEX. – Режим доступу: https://docs.oracle.com/en/database/oracle/apex/23.1/ (дата звернення: 12.10.2023 р.).

10. Oracle Database 23c Installation Guide: Керівництво з встанов-лення Oracle Database 23c, яке надає детальні інструкції щодо процесу вста-новлення та налаштування. – Режим доступу: https://docs.oracle.com/en/database/oracle/oracle-database/23/xeinl/index.html (дата звернення: 12.10.2023 р.).

11. Oracle Database 23c XE Community: Спільнота користувачів та фахівців Oracle Database 23c XE, де можна знайти поради, трюки та відповіді на питання. – Режим доступу: https://forums.oracle.com/ords/apexds/domain/dev-community/category/apex (дата звернення: 12.10.2023 р.).

12. Oracle Learning Library: Безкоштовні онлайн-уроки та матеріали з Oracle Database та Oracle APEX. – Режим доступу: https://www.oracle.com/technetwork/tutorials/index.html (дата звернення: 12.10.2023 р.).

Список рекомендованих джерел для самопідготовки$_3$

13. Harper, J., & Li, K. (2009). Oracle SQL*Plus: The Definitive Guide (Definitive Guides). O'Reilly Media.

14. Loney, K., & Koch, G. (2004). Oracle Database 10g: The Complete Reference. McGraw-Hill Osborne Media.

15. Priyadarshi, S. (2015). Oracle SQL and PL/SQL Hand Book: Solved SQL and PL/SQL Questions and Answers Including Queries and Tips. Packt Publishing.

16. Feuerstein, S., & Pribyl, B. (2009). Oracle PL/SQL Programming. O'Reilly Media.

Список рекомендованих джерел для самопідготовки$_4$

17. Docker Overview – Docker Docs [Електронний ресурс]. – Режим дос-тупу: https://docs.docker.com/get-started/overview/ (дата звернення: 24.10.2023 р.).

18. Docker Compose Overview – Docker Docs [Електронний ресурс]. – Режим до-ступу: https://docs.docker.com/compose/ (дата звернення: 24.10.2023 р.).

19. OpenLDAP, Documentation [Електронний ресурс]. – Режим доступу: https://www.openldap.org/doc/ (дата звернення: 24.10.2023 р.).

20. LDAP – Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/LDAP (дата звернення: 24.10.2023 р.).

Список рекомендованих джерел для самопідготовки$_5$

21. How to Configure TDE in Oracle 19c [Електронний ресурс]. – Режим доступу: https://anuragkumarjoy.blogspot.com/2021/04/how-to-configure-tde-in-oracle-19c.html (дата звернення: 03.12.2023р.).

22. Transparent Data Encryption (TDE) in Oracle [Електронний ресурс]. – Режим доступу: https://desktop.arcgis.com/ru/arcmap/latest/extensions/maritime-bathymetry-guide/oracle/transparent-data-encryption-tde-in-oracle.htm (дата звернення: 03.12.2023р.).

23. An Interactive Report with Dynamic Column Headings [Електронний ресурс]. – Режим доступу: https://planetoftheapex.wordpress.com/2016/12/12/an-interactive-report-with-dynamic-column-headings/ (дата звернення: 15.12.2023р.).

24. Customizing DML in an APEX Interactive Grid [Електронний ресурс]. – Режим доступу: https://mikesmithers.wordpress.com/2019/07/23/customizing-dml-in-an-apex-interactive-grid/ (дата звернення: 15.12.2023р.).

25. Working with cursors and dynamic queries in PL/SQL [Електронний ре-сурс]. – Режим доступу: https://blogs.oracle.com/connect/post/working-with-cursors (дата звернення: 15.12.2023р.).

Список рекомендованих джерел для самопідготовки$_6$

26. PL SQL Transactions – COMMIT, ROLLBACK And SAVEPOINT [Електронний ресурс]. – Режим доступу: https://www.softwaretestinghelp.com/pl-sql-transactions/ (дата звернення: 17.12.2023р.).

27. PL/SQL Transactions [Електронний ресурс]. – Режим доступу: https://www.codingninjas.com/studio/library/pl-sql-transactions (дата звернен-ня: 17.12.2023р.).

28. Creating Procedures, Functions, and Packages in PL/SQL [Електронний ресурс]. – Режим доступу: https://www.techsupper.com/2022/09/creating-procedures-functions-and-packages-in-pl-sql/ (дата звернення: 17.12.2023р.).

29. Oracle PL/SQL Stored Procedure & Functions with Examples [Елект-ронний ресурс]. – Режим доступу: https://www.guru99.com/subprograms-procedures-functions-pl-sql.html (дата звернення: 17.12.2023р.).

30. Stored Procedure and Function in PL/SQL [Електронний ресурс]. – Ре-жим доступу: https://www.studytonight.com/plsql/plsql-procedure-and-function (дата звернення: 17.12.2023р.).

Список рекомендованих джерел для самопідготовки$_7$

31. Triggers in PL/SQL [Електронний ресурс]. – Режим доступу: https://www.studytonight.com/plsql/plsql-triggers (дата звернення: 17.12.2023р.).

32. PL/SQL – Triggers [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/plsql/plsql_triggers.htm (дата звернення: 17.12.2023р.).

33. Oracle Trigger [Електронний ресурс]. – Режим доступу: https://www.oracletutorial.com/plsql-tutorial/oracle-trigger/ (дата звернення: 17.12.2023р.).