

СУЧАСНІ СУБД

Лекція №13

Тема:

ORACLE PL/SQL

Збережені процедури,
пакети та тригери

Модульний код

Модуляризацією називають розбиття великих блоків коду на менші блоки (модулі), які можна викликати з інших модулів. Цей процес схожий з нормалізацією даних. Модуляризація дозволяє поліпшити ряд характеристик коду.

№	Характеристика	Опис
1	Придатність для повторного використання	Розбиття великих програм на окремі взаємодіючі компоненти здійснюють так, щоб багато модулів було можливо використати більш ніж однією програмою поточного застосунка. При правильній організації такі програми можуть стати в нагоді в інших програмах!
2	Керованість	При використанні модульного підходу код можна протестувати і налагодити на рівні окремих програм (так зване модульне тестування), тобто до того, як окремі модулі будуть скомбіновані для проведення більш складних інтеграційних тестів.
3	Надійність	Код, розбитий на модулі, містить менше помилок, а виявлені помилки легше виправляти, оскільки вони локалізовані на рівні модуля. Крім того, такий код легше супроводжувати іншим програмістам.
4	Зручність читання	Імена модулів відображають їх призначення. Чим більше коду буде переміщено в програмний інтерфейс або приховано за ним, тим легше буде зрозуміти, що робить програма. Модуляризація дає можливість зосередитися на завданні в цілому, а не на окремих фрагментах коду.

Конструкції PL/SQL для розбиття коду на модулі

№	Конструкція	Опис
1	Процедура	Програма, яка здійснює одну або кілька дій і викликається як виконуваний оператор <i>PL / SQL</i> . Передавати дані процедурі і отримувати їх можна за допомогою списку параметрів.
2	Функція	Програма, яка повертає одне значення і використовується як вираз <i>PL/SQL</i> . Для передачі даних функції використовують список її параметрів. Параметри також можна використовувати для повернення інформації з функції, але зазвичай це вважається проявом поганого стилю програмування.

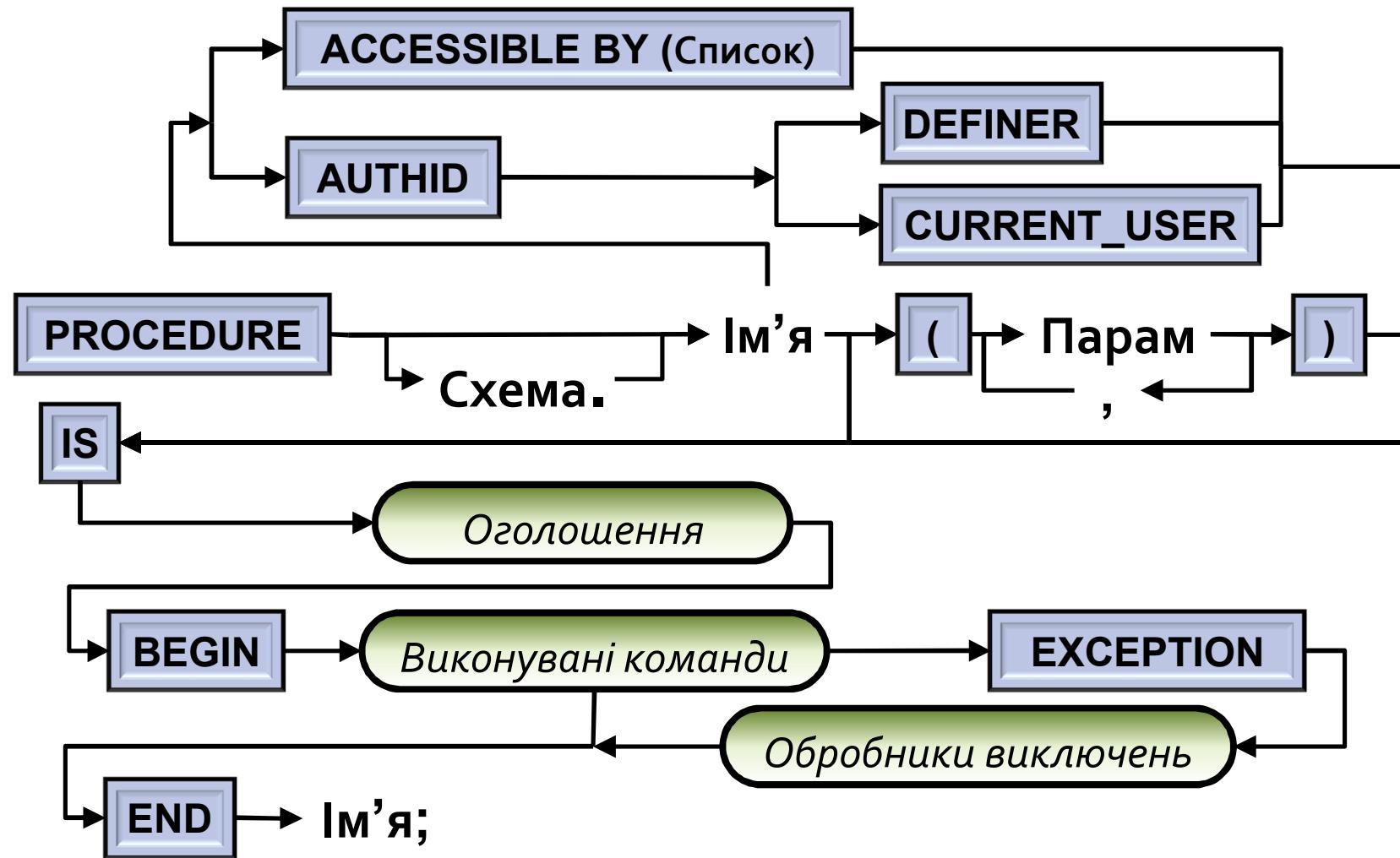
Модульний код

Конструкції PL/SQL для розбиття коду на модулі

№	Конструкція	Опис
3	Тригер бази даних	Набір команд, який викликається при виконанні деякої події (підключення до бази даних, модифікація рядка таблиці або <i>DDL</i> -операція)
4	Пакет	Іменований набір процедур, функцій, типів і змінних. Пакет не є модулем (скоріше, це мета-модуль), але він тісно пов'язаний з реалізацією модульного підходу.
5	Об'єктний тип або екземпляр об'єктного типу	Емуляція об'єктно-орієнтованого класу в Oracle. Об'єктний тип інкапсулює стан і поведінку даних, комбінуючи їх (як реляційна таблиця) з правилами (процедурами і функціями, які маніпулюють цими даними).

Процедури. Синтаксис.

Процедура представляє собою модуль, що виконує одну або кілька дій. Виклик процедури в PL / SQL є окремим виконуваним оператором. Блок PL/SQL коду може складатися тільки з виклику процедури. Процедури відносяться до числа ключових компонентів модульного коду, що забезпечують оптимізацію і повторне використання програмної логіки.



Процедури. Структура.

Схема – ім'я схеми, якій буде належати процедура (необов'язковий аргумент). За замовчуванням застосовується ім'я схеми поточного користувача. Якщо значення схеми відмінне від імені схеми поточного користувача, то цей користувач повинен мати привілей для створення процедури в іншій схемі.

Ім'я — ім'я процедури

Парам – необов'язковий список параметрів, які застосовуються для передачі даних в процедуру і повернення інформації з процедури в точку виклику.

AUTHID - визначає, за якими дозволами буде викликатися процедура: творця (власника) або поточного користувача. У першому випадку процедура виконується з правами творця, у другому - з правами того хто викликає.

Оголошення - оголошення локальних ідентифікаторів цієї процедури. Якщо оголошення відсутні, між ключовими словами **IS** і **BEGIN** не буде жодних виразів.

ACCESSIBLE BY (Oracle Database 12c) – обмежує доступ до процедури програмним модулем, перерахованим в круглих дужках.

Виконувані команди – команди, що виконуються процедурою при виклику. Між ключовими словами **BEGIN** і **END** або **EXCEPTION** повинна знаходитися принаймні одна виконувана команда.

Обробники виключень – необов'язкові обробники виключень для процедури. Якщо процедура не обробляє ніяких виключень, слово **EXCEPTION** можна опустити і завершити виконуваний розділ ключовим словом **END**.

Процедури. Приклад.

```

PROCEDURE apply_discount
(
    company_id_in IN company.company_id%TYPE,
    discount_in    IN NUMBER
)
IS
    min_discount CONSTANT NUMBER:=.05;
    max_discount CONSTANT NUMBER:=25;
    invalid_discount EXCEPTION;
BEGIN
    IF discount_in BETWEEN min_discount AND max_discount
    THEN
        UPDATE item SET item_amount:=item_amount*(1-discount_in)
        WHERE EXISTS
        (
            SELECT 'x' FROM order
            WHERE order.order_id=item.order_id
            AND
            order.company_id=company_id_in
        );
        IF SQL%ROWCOUNT = 0 THEN RAISE NO_DATA_FOUND; END IF;
        ELSE RAISE invalid_discount;
    END IF;
EXCEPTION
    WHEN invalid_discount
    THEN DBMS_OUTPUT.PUT_LINE('The specified discount is invalid.');
    WHEN NO_DATA_FOUND
    THEN DBMS_OUTPUT.PUT_LINE('No orders in the system for company');
END apply_discount;

```

Заголовок

Оголошення

Виконуваний розділ

Розділ виключень

Процедури. Виконання.

Виклик процедури

Процедура викликається як виконавська команда **PL/SQL**. Її виклик повинен закінчуватися крапкою з комою ; і може передувати іншим командам **SOL** або **PL/SQL** (якщо такі є) в виконуваному розділі блоку **PL/SOL** або слідувати за ними:

BEGIN

```
    apply_discount (new_company_id., 0.15);
END;
```

Якщо процедура не має параметрів, вона може бути викликана з порожніми круглими дужками або без них:

```
display_store_summary();
display_store_summary;
```

Команда RETURN

Ключове слово **RETURN** зазвичай асоціюється з функціями, оскільки вони повинні повернати значення. Однак **PL/SQL** дозволяє використовувати команду **RETURN** в процедурах. Версія цієї команди для процедур не приймає виразів і не може повернати значення в програмний модуль з точкою виклику процедури. Вона просто припиняє виконання процедури і повертає управління в точку виклику коду.

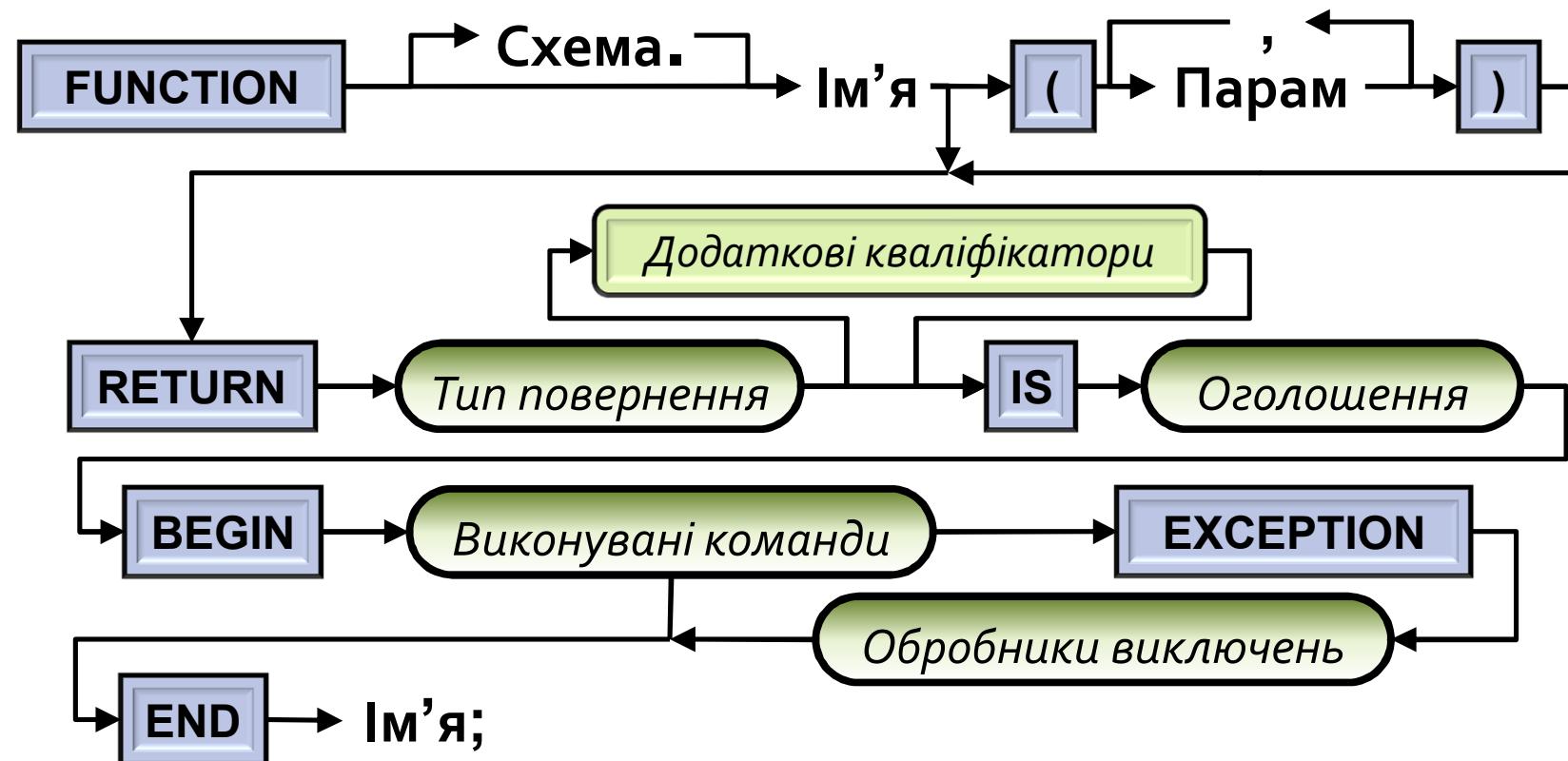


*Використовувати цей різновид **RETURN** не рекомендується, оскільки в цьому випадку в процедурі з'являються дві і більше точки виходу, а це ускладнює логіку виконання. Взагалі бажано уникати використання **RETURN** і **GOTO** для обходу нормальній керуючої структури в програмних елементах.*

Функції. Синтаксис.

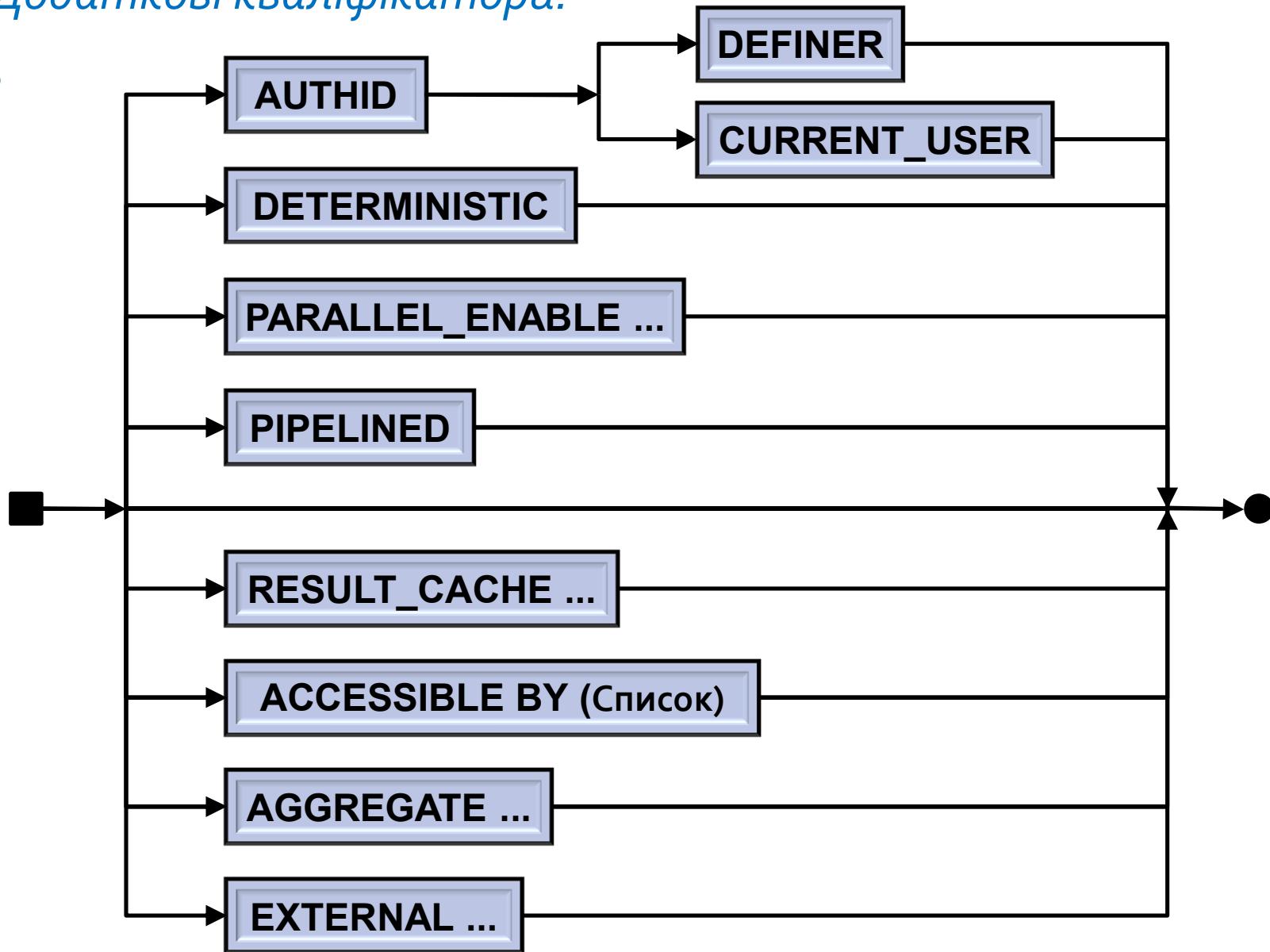
Функція являє собою модуль, який повертає значення командою **RETURN** (замість аргументів **OUT** або **IN OUT**). На відміну від виклику процедури, який являє собою окремий оператор, виклик функції завжди є частиною виконуваного оператора, тобто включається в вираз або править значенням за замовчуванням, що присвоюється змінної при оголошенні.

Значення, що повертається функцією належить до визначеного типу даних. Функції можна використовувати замість виразів, які мають той же тип даних, що і значення, які повертаються.



Функції. Синтаксис.

Додаткові кваліфікатори:



Функції. Структура.

Схема – ім'я схеми, якій буде належати функція (необов'язковий аргумент). За замовчуванням застосовується ім'я схеми поточного користувача. Якщо значення схеми відмінне від імені схеми поточного користувача, то цей користувач повинен мати привілеї для створення функцій в іншій схемі.

Ім'я – ім'я функції

Параметр – необов'язковий список параметрів, які застосовуються для передачі даних в функцію і повернення інформації з неї в точку виклику.

Тип повернення – задає тип значення, що повертається функцією. Тип повернення повинен бути визначений в заголовку функції.

AUTHID – визначає, за якими дозволами буде викликатися функція: власника або поточного користувача. У першому випадку (використовується за замовчанням) застосовується модель прав власника, у другому – модель прав користувача, хто викликає функцію.

DETERMINISTIC – визначає функцію як детерміновану, тобто значення повернення повністю визначається значеннями її аргументів. Якщо включити цю секцію, ядро SQL зможе оптимізувати виконання функції при її виклику в запитах.

PARALLEL_ENABLE – використовується для оптимізації і дозволяє функції виконуватися паралельно в разі, коли вона викликається з команди **SELECT**.

PIPELINED – вказує, що результат табличній функції повинен повертатися в ітеративному режимі за допомогою команди **PIPE ROW**.

RESULT_CACHE – вказує, що вхідні дані і результат виклику функції повинен бути збережений в кеші результатів. (Можливо лише в Oracle 11g і вище).

ACCESSIBLE BY (*Oracle Database 12c*) – обмежує доступ до функції програмними модулями, перерахованими в круглих дужках.

EXTERNAL – визначає функцію з «зовнішньою реалізацією» - тобто написану на мові С.

AGGREGATE – використовується при визначенні агрегатних функцій.

Оголошення – оголошення локальних ідентифікаторів цієї функції. Якщо оголошення відсутні, між ключовими словами **IS** і **BEGIN** не буде жодних виразів.

Виконувані команди – команди, що виконуються функцією при виклику. Між ключевими словами **BEGIN** і **END** або **EXCEPTION** повинна знаходитися принаймні одна виконувана команда.

Обробники виключень – необов'язкові обробники виключень для функції. Якщо функція не виконує жодних виключень, слово **EXCEPTION** можна опустити і завершити виконуваний розділ ключовим словом **END**.

Функції. Приклад.

```

FUNCTION tot_sales
(
    company_id_in IN company.company_id%TYPE,
    statusjn IN order.status_code%TYPE:=NULL
)
RETURN NUMBER
IS
    status intorder.status_code%TYPE:=UPPER(status_in);
    CURSOR sales_cur (IN status_code%TYPE) IS
        SELECT SUM (amount*discount) FROM item
        WHERE EXISTS
        (
            SELECT 'X' FROM order
            WHERE order.order_id=item.order_id AND
                company_id=company_id_in AND
                status_code LIKE statusjn
        );
    returnn_value NUMBER;
BEGIN
    OPEN sales_cur (statusjnt);
    FETCH sales_cur INTO return_value;
    IF sales_cur%NOTFOUND
        THEN CLOSE sales_cur; RETURN NULL;
    ELSE CLOSE sales_cur; RETURN return_value;
    END IF;
END tot_sales;

```

Заголовок функції

Оголошення

Виконуваний розділ

Функції. Тип повернення.

Тип повернення

Функція PL/SQL може повертати дані практично будь-якого типу, підтримуваного PL/SQL, – від скалярів (одиничних значень на зразок дати або рядка) до складних структур: колекцій, об'єктних типів, курсорних змінних, тощо.

Повернення рядку:

```
FUNCTION favorite_nickname
(
    name_in IN VARCHAR2
)
RETURN VARCHAR2
IS
BEGIN
...
END favorite_nickname;
```

Повернення курсорної змінної:

```
TYPE overdue_rt IS RECORD
(
    isbn books.isbn% TYPE,
    days_overdue PLS_INTEGER
);
TYPE overdue_rct IS REF CURSOR RETURN overdue_rt;
FUNCTION overdue_info
(
    username_in IN lib_users.username% TYPE
)
RETURN overdue_rct
IS
BEGIN
...
END overdue_info;
```

```
FUNCTION onerow
(
    isbn_in IN books.isbn% TYPE
)
RETURN books% ROWTYPE
IS
BEGIN
...
END onerow;
```

Повернення запису таблиці:

Виклик функції.

Функція може бути викликана з будь-якої частини виконуваної команди PL/SQL, де допускається використання виразу.

```
DECLARE
```

```
    v_nickname VARCHAR2 (100) := favorite_nickname ('Steven');
```

```
...
```

```
DECLARE
```

```
    TYPE pet_t IS OBJECT
```

```
(
```

```
        tag_no INTEGER,  
        name VARCHAR2 (60),  
        breed VARCHAR2(100),  
        dob DATE,
```

```
    MEMBER FUNCTION age RETURN NUMBER
```

```
);
```

```
my_parrot pet_t: =pet_t
```

```
(
```

```
    1001,
```

```
    'Mercury',
```

```
    'African Grey',
```

```
    TO_DATE ('09 / 23/1996 ',' MM / DD / YYYY ')
```

```
);
```

```
BEGIN
```

```
    IF my_parrot.age () < INTERVAL '50' YEAR
```

```
    THEN DBMS_OUTPUT.PUT_LINE ( 'Still a youngster!');
```

```
END IF;
```

```
...
```

```
END;
```

```
DECLARE
```

```
    my_first_book books% ROWTYPE;
```

```
BEGIN
```

```
    my_first_book: = book_info.onerow ('1-56592-335-9');
```

```
...
```

```
END;
```

Виклик функції.

DECLARE

```
    l_name employees.last_name% TYPE;
BEGIN
    SELECT last_name INTO l_name FROM employees
    WHERE employee_id = hr_info_pkg.employee_of_the_month ('FEBRUARY');
    ...
END;
```

```
CREATE VIEW young_managers AS
    SELECT managers.employee_id AS manager_employee_id
    FROM employees managers
    WHERE most_reports_before_manager
    (
        CURSOR
        (
            SELECT reports.hire_date FROM employees reports
            WHERE reports.manager_id = managers.employee_id
            ),
            managers.hire_date
        ) = 1;
```



В PL/SQL, неможливо просто проігнорувати значення повернуття функції, навіть якщо воно не буде використане. Наприклад, для виклику функції: **BEGIN favorite_nickname ('Steven');** **END;** буде видана помилка **PLS-00221: 'FAVORITE_NICKNAME' is not a procedure or is undefined.** Функцію неможна використовувати так, як процедуру.

Параметри процедур і функцій.

Параметри використовуються для передачі інформації між модулем і блоком виклику. Параметри модуля є частиною його заголовка (або сигнатури). Вони не менш важливі компоненти модуля, ніж виконувані команди, що складають його тіло.



Кількість параметрів. Якщо процедура або функція має занадто мало параметрів, це знижує її універсальність. Занадто велика кількість параметрів ускладнює її повторне використання. Зазвичай, кількість параметрів визначається вимогами програми, але існують різні варіанти їх визначення (наприклад, кілька параметрів можна об'єднати в одну запис).



Типи параметрів. При виборі типу необхідно враховувати, для яких цілей будуть використовуватися параметри: тільки для читання, тільки для запису, для читання і запису.



Імена параметрів. Параметрам слід привласнювати прості імена, що відображають їх призначення в модулі.



Значення за замовчуванням. Значення за замовчуванням варто присвоювати завжди. Це підвищує стійкість програми. Коли параметру слід задати значення за замовчуванням, ставлять найчастіше вживане значення, а коли потрібно змусити програміста ввести певне значення, ставлять нетпове, або абсурдне щодо контексту задачі значення з обов'язковою його посттробокою.

Режими передачі параметрів.

Режим	Призначення	Використання параметрів
IN	Тільки для читання	Значення параметра може застосовуватися, але не може бути змінено в модулі. Якщо режим параметра не заданий, використовується режим IN
OUT	Тільки для запису	У модулі можна привласнити значення параметру, але не можна використовувати його. Втім, це «офіційне» визначення - насправді Oracle дозволяє читати значення параметра OUT в підпрограмі
IN OUT	Для читання і запису	У модулі можна використовувати і змінювати значення параметра

```

PROCEDURE predict_activity
(
    last_date_in IN DATE,
    task_desc_inout IN OUT VARCHAR2,
    next_date_out OUT DATE
)

```

Режими передачі параметрів.

Режим IN

Параметр **IN** дозволяє передавати значення модулю, але не може використовуватися для передачі інформації з модуля викликає блоку **PL/SQL**. В контексті викликаної програми параметри **IN** працюють як константи. Значення формального параметра **IN**, як і значення константи, не може бути змінено всередині програми. Параметру **IN** не можна присвоїти нове значення або змінити його іншим чином – компілятор видасть повідомлення про помилку. Режим **IN** використовується за умовчанням; якщо режим параметра не заданий, параметр автоматично вважається визначеним в режимі **IN**. Проте варто завжди вказувати режим параметра, щоб очікуване використання було явно зазначено в коді. Фактичним значенням параметра **IN** може бути змінна, іменована константа, літерал або складний вираз.

Режим OUT

Параметр **OUT** за змістом протилежний параметру **IN**. Параметри **OUT** можуть використовуватися для повернення значень з програми викликає блоку **PL/SQL**. Параметр **OUT** схожий з значенням повернення функції, але він включається в список параметрів, і кількість таких параметрів не обмежена (64 000). У програмі параметр **OUT** працює як неініціалізованих змінна. Він не містить ніякого значення до успішного завершення викликаної програми. Під час виконання програми всі операції присвоювання параметру **OUT** насправді виконуються з внутрішньою копією параметра. Коли програма успішно завершується і повертає управління в точку виклику, значення локальної копії переміщається в параметр **OUT** і стає доступним в блоці з точкою виклику.

Режими передачі параметрів.



*Параметрами **OUT** неможна задавати значення за замовчуванням. Значення параметра **OUT** може здаватися тільки в тілі модуля.*



*Всі операції присвоювання параметрам **OUT** скасовуються при ініціюванні виключення в програмі. Так як значення параметра **OUT** присвоюється тільки в разі успішного завершення програми, все проміжні присвоювання ігноруються. Якщо обробник НЕ перехопить виключення і не присвоїть значення параметру **OUT**, параметр залишиться незмінним. Змінна збереже значення, яке вона мала до виклику програми.*



*Фактичний параметр, відповідний формальному параметру **OUT**, не може бути константою, літералом або виразом. Інакше кажучи, він повинен підтримувати присвоювання.*

Режим **IN OUT**

У параметрі **IN OUT** можна передавати значення програмі і повернати їх на сторону виклику (або вихідне, незмінне значення, або нове значення, задане в програмі). На параметри **IN OUT** поширюються два обмеження параметрів **OUT**:

- *Параметр **IN OUT** не може бути константою, літералом або виразом.*
- *Фактичний параметр **IN OUT** повинен бути змінною.*

В цьому режимі параметр не може бути константою, літералом або виразом, тому що ці формати не можуть використовуватися **PL/SQL** як приймачі для розміщення вихідних значень. Параметри **IN OUT** можуть використовуватися в обох сторонах привласнення, тому що вони працюють як ніціалізовані змінні. **PL/SQL** не втрачає значення параметра **IN OUT** на початку виконання програми. Це значення може використовуватися в програмі будь-де.

Зв'язування формальних та фактичних параметрів.

В PL/SQL представлена два методи встановлення відповідності між формальними і фактичними параметрами:

- за позицією (неявне зв'язування);
- за іменем (явне зв'язування із зазначенням імені формального параметра і позначення $=>$).

Заголовок процедури

```
PROCEDURE calc_all (id_in IN INTEGER, total_out OUT NUMBER)
```

Виклик процедури

```
calc_all (1007, tot_sales);
```

Щоб встановити відповідність параметрів по імені, слід при виклику підпрограми явно зв'язати формальний параметр з фактичним. Для цього використовується комбінація символів $=>$:

ім'я_формального_параметра $=>$ значення_параметра

```
new_sales := total_sales (company_id_in => order_pkg.company_id, status_in =>'N');
new_sales := total_sales (status_in =>'N', company_id_in => order_pkg.company_id);
new_sales := total_sales (order_pkg.company_id, status_in =>'N');
```

```
l_new_sales := total_sales (company_id_in => order_pkg.company_id, 'N');
l_new_sales := total_sales ('N', company_id_in => order_pkg.company_id);
```