

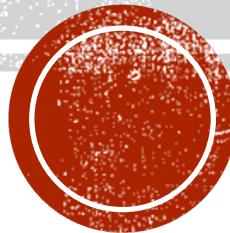
BASI DI DATI I

- PL/SQL
- SQL dinamico





PLSQL



BLOCK STRUCTURE

- I programmi PL/SQL sono divisi in strutture chiamate **blocks**
- Ogni block contiene istruzioni che possono essere
 - PL/SQL
 - Pure SQL

```
[DECLARE
    declaration_statements
]
BEGIN
    executable_statements
[EXCEPTION
    exception_handling_statements
]
END;
/
```

Il blocco DECLARE è opzionale.

Contiene le variabili che saranno utilizzate nel seguito del programma.

Il blocco BEGIN/END contiene tutte le istruzioni che saranno effettivamente eseguite come istruzioni, cicli, ...

Il blocco EXCEPTION è opzionale.

Contiene le istruzioni da eseguire per gestire le eccezioni

Ogni block PL/SQL termina con uno slash (/)



ESEMPIO

```
SET SERVEROUTPUT ON

DECLARE
    v_width  INTEGER;
    v_height INTEGER := 2;
    v_area   INTEGER := 6;
BEGIN
    -- set the width equal to the area divided by the height
    v_width := v_area / v_height;
    DBMS_OUTPUT.PUT_LINE('v_width = ' || v_width);
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('Division by zero');
END;
/
```



TIPI E VARIABILI

- Sono gli stessi che vengono utilizzati per la definizione delle tabelle
 - Allineamento perfetto tra i tipi delle variabili e le colonne del DB

```
v_product_id      INTEGER;
v_product_type_id INTEGER;
v_name            VARCHAR2(30);
v_description     VARCHAR2(50);
v_price           NUMBER(5, 2);
```

DECLARE

```
<nome_variabile> <tipo_del_dato> [:= valore]
<nome_variabile> <tabella>.<attributo>%TYPE
```



OPERAZIONI DI SELECT

- Una operazione di SELECT può restituire tre tipi di output
 1. Una sola riga
 2. Nessuna riga
 3. Un insieme di righe

Nel **primo caso**, posso andare a recuperare i valori restituiti dichiarando nel blocco *DECLARE* le variabili restituite ed associandole all'output della SELECT tramite la keyword **INTO**.

```
DECLARE
    Vnome, Vcognome VARCHAR(20);
BEGIN
    SELECT S.Nome, S.Cognome INTO Vnome, Vcognome
    FROM Studente S
    WHERE S.Matricola = 'N8600001941'
```



OPERAZIONI DI SELECT

- Una operazione di SELECT può restituire tre tipi di output
 1. Una sola riga
 2. Nessuna riga
 3. Un insieme di righe

Nel **secondo e terzo caso**, mi trovo nella situazione in cui non so quante righe sono state restituite.

Devo pertanto definire un **CURSOR** per andare a scorrere il ResultSet ottenuto.



COSTRUTTI CONDIZIONALI

```
IF condition1 THEN  
    statements1  
ELSIF condition2 THEN  
    statements2  
ELSE  
    statements3  
END IF;
```

*condition1 & condition2 sono espressioni booleane
statements1, statements2, statements3 sono istruzioni PLSQL*

```
    IF v_count > 0 THEN  
        v_message := 'v_count is positive';  
        IF v_area > 0 THEN  
            v_message := 'v_count and v_area are positive';  
        END IF  
    ELSIF v_count = 0 THEN  
        v_message := 'v_count is zero';  
    ELSE  
        v_message := 'v_count is negative';  
    END IF;
```



COSTRUTTI ITERATIVI

- Abbiamo 3 tipi di costrutti iterativi

1. *LOOP semplici* che vengono eseguiti finché non si chiude esplicitamente il ciclo
2. *WHILE* che cicla finché è verificata una determinata condizione
3. *FOR* che cicla per un predeterminato numero di volte



LOOP SEMPLICI

LOOP

statements

END LOOP

Per terminare il ciclo si può usare:

- EXIT per terminare il loop immediatamente
- EXIT WHEN per terminare il loop quando una determinata condizione occorre

```
v_counter := 0;  
LOOP  
    v_counter := v_counter + 1;  
    EXIT WHEN v_counter = 5;  
END LOOP;
```



LOOP SEMPLICI - CONTINUE

```
v_counter := 0;  
LOOP  
    -- after the CONTINUE statement is executed, control returns here  
    v_counter := v_counter + 1;  
    IF v_counter = 3 THEN  
        CONTINUE; -- end current iteration unconditionally  
    END IF;  
    EXIT WHEN v_counter = 5;  
END LOOP;
```

```
v_counter := 0;  
LOOP  
    -- after the CONTINUE WHEN statement is executed, control returns here  
    v_counter := v_counter + 1;  
    CONTINUE WHEN v_counter = 3; -- end current iteration when v_counter = 3  
    EXIT WHEN v_counter = 5;  
END LOOP;
```



WHILE

WHILE condition LOOP

statements

END LOOP;

```
v_counter := 0;  
WHILE v_counter < 6 LOOP  
    v_counter := v_counter + 1;  
END LOOP;
```



CICLI FOR

- Un ciclo for è eseguito per un determinato numero di volte
- Si imposta il numero di volte specificando il *lower bound* e l' *upper bound* per la variabile di loop
- La variabile è incrementata/decrementata ogni volta che si fa un nuovo giro nel loop.

```
FOR loop_variable IN [REVERSE] lower_bound..upper_bound LOOP  
    statements  
END LOOP;
```



CICLI FOR (2)

- *loop_variable* è la variabile che regola il loop. È possibile utilizzare una variabile che già esiste o se ne utilizza una apposta per il loop (creata nel caso in cui la variabile specificata non esiste). La variabile è incrementata di 1 ogni volta che si passa attraverso il loop. Se non definita al di fuori del loop, la variabile non è più visible all'uscita del FOR.
- *REVERSE* indica se la variabile deve essere incrementata o decrementata. In ogni caso, il lower bound deve essere specificato prima dell'upper bound.

```
FOR v_counter2 IN 1..5 LOOP  
    DBMS_OUTPUT.PUT_LINE(v_counter2);  
END LOOP;
```

```
FOR v_counter2 IN REVERSE 1..5 LOOP  
    DBMS_OUTPUT.PUT_LINE(v_counter2);  
END LOOP;
```



CURSORI

- Un cursore viene dichiarato nel blocco *DECLARE*

DECLARE

```
Vmatricola Studente.Matricola%TYPE;  
Vnome Studente.Nome%TYPE;  
cursor scansiona_cognome IS  
    SELECT S.Nome, S.Matricola  
    FROM Studente S  
    WHERE S.Cognome = 'Russo'
```

BEGIN

```
OPEN scansiona_cognome  
FETCH scansiona_cognome INTO Vnome, Vmatricola;  
...  
...  
CLOSE scansiona_cognome
```

- *La dichiarazione del cursore non esegue l'interrogazione*



CURSORI

```
DECLARE
    cursor scansiona_cognome IS
        SELECT S.Nome, S.Matricola
        FROM Studente S
        WHERE S.Cognome = 'Russo'
    rigacorrente scansiona_cognome%ROWTYPE
BEGIN
    FETCH scansiona_cognome INTO rigacorrente
    ...rigacorrente.Matricola...
    ...rigacorrente.Nome...
```

Posso evitare di definire le variabili che vanno ad ospitare gli attributi recuperati dal cursore definendo un tipo **%ROWTYPE** collegato al cursore



ESEMPIO CURSOR

```
LOOP
    -- fetch the rows from the cursor
    FETCH v_product_cursor
    INTO v_product_id, v_name, v_price;

    -- exit the loop when there are no more rows, as indicated by
    -- the Boolean variable v_product_cursor%NOTFOUND (= true when
    -- there are no more rows)

    EXIT WHEN v_product_cursor%NOTFOUND;

    -- use DBMS_OUTPUT.PUT_LINE() to display the variables
    DBMS_OUTPUT.PUT_LINE(
        'v_product_id = ' || v_product_id || ', v_name = ' || v_name ||
        ', v_price = ' || v_price
    );
END LOOP;
```



```

-- This script displays the product_id, name, and price columns
-- from the products table using a cursor

SET SERVEROUTPUT ON

DECLARE
    -- step 1: declare the variables
    v_product_id products.product_id%TYPE;
    v_name      products.name%TYPE;
    v_price     products.price%TYPE;

    -- step 2: declare the cursor
    CURSOR v_product_cursor IS
        SELECT product_id, name, price
        FROM products
        ORDER BY product_id;
    BEGIN
        -- step 3: open the cursor
        OPEN v_product_cursor;

        LOOP
            -- step 4: fetch the rows from the cursor
            FETCH v_product_cursor
            INTO v_product_id, v_name, v_price;

            -- exit the loop when there are no more rows, as indicated by
            -- the Boolean variable v_product_cursor%NOTFOUND (= true when
            -- there are no more rows)
            EXIT WHEN v_product_cursor%NOTFOUND;

            -- use DBMS_OUTPUT.PUT_LINE() to display the variables
            DBMS_OUTPUT.PUT_LINE(
                'v_product_id = ' || v_product_id || ', v_name = ' || v_name ||
                ', v_price = ' || v_price
            );
        END LOOP;

        -- step 5: close the cursor
        CLOSE v_product_cursor;
    END;
/

```

ESEMPIO CON il FOR LOOP per i CURSORI

```

-- This script displays the product_id, name, and price columns
-- from the products table using a cursor and a FOR loop

SET SERVEROUTPUT ON

DECLARE
    CURSOR v_product_cursor IS
        SELECT product_id, name, price
        FROM products
        ORDER BY product_id;
    BEGIN
        FOR v_product IN v_product_cursor LOOP
            DBMS_OUTPUT.PUT_LINE(
                'product_id = ' || v_product.product_id ||
                ', name = ' || v_product.name ||
                ', price = ' || v_product.price
            );
        END LOOP;
    END;
/

```

ECCEZIONI

- Le eccezioni sono utilizzate per gestire errori a tempo di esecuzione negli statement PL/SQL.
- Il blocco *EXCEPTION* è in carica di recuperare le eccezioni e gestirle come specificato (se specificato)
 - Altrimenti ogni eccezione ha un gestore di default che la prende in consegna



Exception	Error	Description	SELF_IS_NULL	ORA-30625	An attempt was made to call a MEMBER method on a null object. That is, the built-in parameter SELF (which is always the first parameter passed to a MEMBER method) is null.
ACCESS_INTO_NULL	ORA-06530	An attempt was made to assign values to the attributes of an uninitialized database object. (You'll learn about objects in Chapter 13.)			
CASE_NOT_FOUND	ORA-06592	None of the WHEN clauses of a CASE statement was selected, and there is no default ELSE clause.	STORAGE_ERROR	ORA-06500	The PL/SQL module ran out of memory or the memory has been corrupted.
COLLECTION_IS_NULL	ORA-06531	An attempt was made to call a collection method (other than EXISTS) on an uninitialized nested table or varray, or an attempt was made to assign values to the elements of an uninitialized nested table or varray. (You'll learn about collections in Chapter 14.)	SUBSCRIPT_BEYOND_COUNT	ORA-06533	An attempt was made to reference a nested table or varray element using an index number larger than the number of elements in the collection.
CURSOR_ALREADY_OPEN	ORA-06511	An attempt was made to open an already open cursor. The cursor must be closed before it can be reopened.	SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	An attempt was made to reference a nested table or varray element using an index number that is outside the legal range (-1 for example).
DUP_VAL_ON_INDEX	ORA-00001	An attempt was made to store duplicate values in a column that is constrained by a unique index.	SYS_INVALID_ROWID	ORA-01410	The conversion of a character string to a universal rowid failed because the character string does not represent a valid rowid.
INVALID_CURSOR	ORA-01001	An attempt was made to perform an illegal cursor operation, such as closing an unopened cursor.	TIMEOUT_ON_RESOURCE	ORA-00051	A timeout occurred while the database was waiting for a resource.
INVALID_NUMBER	ORA-01722	An attempt to convert a character string into a number failed because the string does not represent a valid number. Note: In PL/SQL statements, VALUE_ERROR is raised instead of INVALID_NUMBER.	TOO_MANY_ROWS	ORA-01422	A SELECT INTO statement returned more than one row.
LOGIN_DENIED	ORA-01017	An attempt was made to connect to a database using an invalid user name or password.	VALUE_ERROR	ORA-06502	An arithmetic, conversion, truncation, or size-constraint error occurred. For example, when selecting a column value into a character variable, if the value is longer than the declared length of the variable, PL/SQL aborts the assignment and raises VALUE_ERROR. Note: In PL/SQL statements, VALUE_ERROR is raised if the conversion of a character string into a number fails. In SQL statements, INVALID_NUMBER is raised instead of VALUE_ERROR.
NO_DATA_FOUND	ORA-01403	A SELECT INTO statement returned no rows, or an attempt was made to access a deleted element in a nested table or an uninitialized element in an "index by" table.	ZERO_DIVIDE	ORA-01476	An attempt was made to divide a number by zero.
NOT_LOGGED_ON	ORA-01012	An attempt was made to access a database item without being connected to the database.			
PROGRAM_ERROR	ORA-06501	PL/SQL had an internal problem.			
ROWTYPE_MISMATCH	ORA-06504	The host cursor variable and the PL/SQL cursor variable involved in an assignment have incompatible return types. For example, when an open host cursor variable is passed to a stored procedure or function, the return types of the actual and formal parameters must be compatible.			

L'ECCEZIONE OTHERS

- Se non sappiamo quale eccezione può essere lanciata e pertanto non sappiamo quale specificare all'interno del blocco, possiamo sempre affidarci all'eccezione *OTHERS*
- L'eccezione *OTHERS* matcha con tutte le eccezioni.

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(1 / 0);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An exception occurred');
END;
/
```

An exception occurred



PROCEDURE

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
    procedure_body
END procedure_name;
```

- *IN* indica che il parametro deve essere settato ad un valore quando la procedura è eseguita
- *OUT* indica che il valore del parametro è impostato ad un valore nel corpo della procedura
- *IN OUT* indica che il parametro può avere un valore quando è eseguita la procedura e che questo può cambiare nel corpo



ESEMPIO

```
CREATE PROCEDURE update_product_price(
    p_product_id IN products.product_id%TYPE,
    p_factor      IN NUMBER
) AS
    v_product_count INTEGER;
BEGIN
    -- count the number of products with the supplied product_id
    -- (the count will be 1 if the product exists)
    SELECT COUNT(*)
    INTO v_product_count
    FROM products
    WHERE product_id = p_product_id;

    -- if the product exists (v_product_count = 1) then
    -- update that product's price
    IF v_product_count = 1 THEN
        UPDATE products
        SET price = price * p_factor
        WHERE product_id = p_product_id;
        COMMIT;
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
END update_product_price;
/
```

- La procedura accetta in input due parametri, entrambi in IN mode; pertanto i valori per i due parametri devono essere impostati quando la procedura è chiamata.
- E soprattutto non possono essere modificati nel corpo della procedura

ATTENZIONE

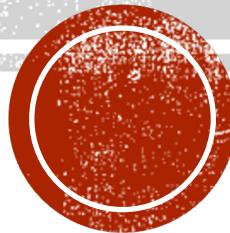
- NON SI PUO' UTILIZZARE UNA SELECT PURA ALL'INTERNO DEL CORPO DELLA PROCEDURA
 - O la si collega ad un cursore
 - O si usa la parola chiave INTO
- NON SI PUO' UTILIZZARE UNA SELECT INTO PER LE QUERY CHE RESTITUISCONO PIU' DI UNA RIGA



STRING FUNCTIONS

ASCII	ASCII('A')	65	Returns an ASCII code value of a character.	REGEXP_REPLACE	REGEXP_LIKE('Year of 2017','\d+')	true	Match a string based on a regular expression pattern.
CHR	CHR('65')	'A'	Converts a numeric value to its corresponding ASCII character.		REGEXP_REPLACE('Year of 2017','\d+', 'Dragon')	'Year of Dragon'	Replace substring in a string by a new substring using a regular expression.
CONCAT	CONCAT('A','B')	'ABC'	Concatenate two strings and return the combined string.	REPLACE	REGEXP_SUBSTR('Number 10', '\d+')	10	Extract substrings from a string using a pattern of a regular expression.
CONVERT	CONVERT('Ã‰', 'US7ASCII', 'WE8ISO8859P1')	'A E I'	Convert a character string from one character set to another.		REPLACE('JACK AND JOND','J','BL');	'BLACK AND BLOND'	Replace all occurrences of a substring by another substring in a string.
DUMP	DUMP('A')	Typ=96 Len=: 65	Return a string value (VARCHAR2) that includes the datatype code, length measured in bytes, and internal representation of a specified expression.	RPAD	RPAD('ABC',5,'*')	'ABC**'	Return a string that is right-padded with the specified characters to a certain length.
INITCAP	INITCAP('hi there')	'Hi There'	Converts the first character in each word in a specified string to uppercase and the rest to lowercase.		RTRIM(' ABC ')	' ABC'	Remove all spaces or specified character in a set from the right end of a string.
INSTR	INSTR('This is a playlist', 'is')	3	Search for a substring and return the location of the substring in a string	SOUNDEX	SOUNDEX('sea')	'S000'	Return a phonetic representation of a specified string.
LENGTH	LENGTH('ABC')	3	Return the number of characters (or length) of a specified string		SUBSTR('Oracle Substring', 1, 6)	'Oracle'	Extract a substring from a string.
LOWER	LOWER('Abc')	'abc'	Return a string with all characters converted to lowercase.	TRANSLATE	TRANSLATE('12345', '143', 'bx')	'b2x5'	Replace all occurrences of characters by other characters in a string.
LPAD	LPAD('ABC','*')	'**ABC'	Return a string that is left-padded with the specified characters to a certain length.		TRIM(' ABC ')	'ABC'	Remove the space character or other specified characters either from the start or end of a string.
LTRIM	LTRIM(' ABC ')	'ABC '	Remove spaces or other specified characters in a set from the left end of a string.	UPPER	UPPER('Abc')	'ABC'	Convert all characters in a specified string to uppercase.
REGEXP_COUNT	REGEXP_COUNT('1 2 3 abc','\d')	3	Return the number of times a pattern occurs in a string.				
REGEXP_INSTR	REGEXP_INSTR('Y2K problem','\d+')	2	Return the position of a pattern in a string.				

SQL DINAMICO



SQL DINAMICO

- Il Dynamic SQL consente di eseguire comandi SQL prodotti a runtime.
- Le differenze principali con l'SQL statico è che mentre nell'SQL statico a parte i parametri, la struttura del comando rimane la stessa, nell'SQL dinamico è l'intero comando ad essere prodotto a tempo di esecuzione.

SQL

- CREATE, ALTER, DROP
- INSERT, UPDATE, DELETE
- COMMIT, ROLLBACK

PL/SQL

```
CREATE PROCEDURE nome_procedura
IS {or AS} -- la clausola IS sostituisce DECLARE definizioni;
BEGIN
    corpo_procedura;
EXCEPTION
    gestione delle eccezioni
END;
/
```



SQL DINAMICO

- Il Dynamic SQL consente di eseguire comandi SQL prodotti a runtime.
- Le differenze principali con l'SQL statico è che mentre nell'SQL statico a parte i parametri, la struttura del comando rimane la stessa, nell'SQL dinamico è l'intero comando ad essere prodotto a tempo di esecuzione.

SQL DINAMICO

```
CREATE PROCEDURE CANCELLA_TABELLA (table_name IN VARCHAR2)
AS
    sql_istr VARCHAR2(100);
BEGIN
    sql_istr := 'DROP TABLE ' || table_name;
    EXECUTE IMMEDIATE (sql_istr );
EXCEPTION
    gestione delle eccezioni
END;
/
```



MODALITÀ DI INTERAZIONE

- SQL Dinamico mette a disposizione due modalità di interazione
 - 1. La gestione dell'interrogazione avviene in due fasi
 - PREPARE in cui vi è una fase di preparazione del comando
 - EXECUTE in cui il comando è mandato in esecuzione
 - 2. L'interrogazione è eseguita immediatamente O in un parametro di tipo stringa che contiene il comando O in un comando specificato direttamente come parametro della EXECUTE IMMEDIATE



MODALITA' 1: PREPARE & EXECUTE

- Il comando PREPARE analizza l'istruzione SQL e ne prepara una traduzione
PREPARE <nome comando> FROM <comando SQL>
- Dove **<nome comando>** è il nome associato da **PREPARE** alla traduzione del comando SQL
- Il comando SQL può contenere dei parametri in ingresso rappresentati da ?
PREPARE comandoSQL
FROM 'SELECT nome FROM studente WHERE matricola = ?'



MODALITÀ 1: PREPARE & EXECUTE

- L'esecuzione del comando **PREPARE** associa alla variabile **comandoSQL** la traduzione dell'interrogazione, con un parametro in ingresso che rappresenta la matricola dello studente.
- L'esecuzione della query avviene tramite il comando **EXECUTE**

EXECUTE <nome comando> [INTO <variabiliTarget>] [USING <lista parametri>]

- **<variabiliTarget>** contiene l'elenco dei parametri in cui deve essere scritto il risultato del comando
- **<lista parametri>** specifica i valori che devono essere assunti dai parametri variabili



MODALITA' 1: PREPARE & EXECUTE

EXECUTE comandoSQL INTO nomeStudente USING matr_studente

dove **matr_studente** è una variabile in cui è inserita la matricola dello studente
N86001941

- Pertanto l'esecuzione del comando effettua la query

SELECT nome

FROM studente

WHERE matricola = 'N86001941'



MODALITÀ 2: EXECUTE IMMEDIATE

- Nella seconda modalità, quella che verrà utilizzata più frequentemente, la **prepare** non è eseguita esplicitamente con un comando apposta, bensì viene effettuata contestualmente alla preparazione.
- Il comando che si utilizza è

EXECUTE IMMEDIATE <nome comando> [INTO <listaTarget>] [USING <listaPar>]

```
comandoSQL := “DELETE FROM Studente WHERE Matricola='N86001941' ”;
EXECUTE IMMEDIATE comandoSQL;
```



ESEMPIO

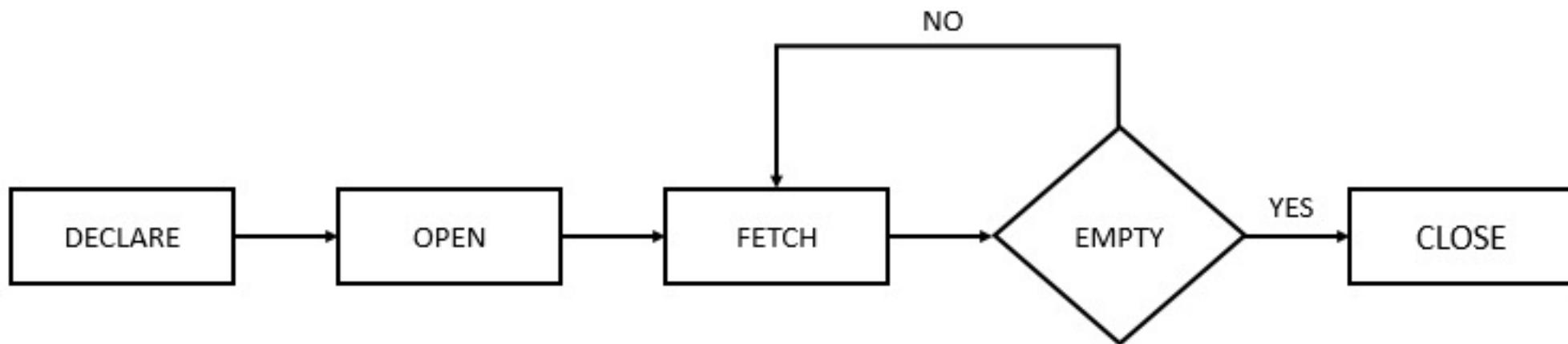
```
DECLARE
    sql_stmt      VARCHAR2(200);
    plsql_block   VARCHAR2(500);
    emp_id        NUMBER(4) := 7566;
    salary         NUMBER(7,2);
    dept_id       NUMBER(2) := 50;
    dept_name     VARCHAR2(14) := 'PERSONNEL';
    location       VARCHAR2(13) := 'DALLAS';
    emp_rec        emp%ROWTYPE;
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE bonus (id NUMBER, amt NUMBER)';
    sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';
    EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name, location;
    sql_stmt := 'SELECT * FROM emp WHERE empno = :id';
    EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING emp_id;
    plsql_block := 'BEGIN emp_pkg.raise_salary(:id, :amt); END;';
    EXECUTE IMMEDIATE plsql_block USING 7788, 500;
    sql_stmt := 'UPDATE emp SET sal = 2000 WHERE empno = :1
                RETURNING sal INTO :2';
    EXECUTE IMMEDIATE sql_stmt USING emp_id RETURNING INTO salary;
    EXECUTE IMMEDIATE 'DELETE FROM dept WHERE deptno = :num'
                    USING dept_id;
    EXECUTE IMMEDIATE 'ALTER SESSION SET SQL_TRACE TRUE';
END;
```



CLASSICO CURSOR

Quando il comando SQL è esplicito

```
CURSOR <nome_cursore> IS <comando SQL>
OPEN <nome_cursore>;
FETCH <nome_cursore> INTO <lista parametri>;
CLOSE <nome_cursore>;
```



CURSOR PER L'SQL DINAMICO

- Un cursore può anche non essere legato obbligatoriamente ad una query in particolare.
- Ciò significa che il cursore potrebbe essere aperto per una qualsiasi query
- Il vantaggio principale di questo tipo di cursori è il seguente:
 1. Innanzitutto la possibilità di utilizzare un cursore per una query costruita dinamicamente

```
comandoSQL := 'SELECT * FROM studente WHERE ' || nomeAttributo || '=' || nomeValore;  
OPEN nomeCursore FOR comandoSQL;
```

2. Inoltre, è possibile restituire una variabile di tipo cursore come output di una function



```
CREATE OR REPLACE FUNCTION get_direct_reports(
    in_manager_id IN employees.manager_id%TYPE)
RETURN SYS_REFCURSOR
AS
    c_direct_reports SYS_REFCURSOR;
BEGIN
    OPEN c_direct_reports FOR
        SELECT
            employee_id,
            first_name,
            last_name,
            email
        FROM
            employees
        WHERE
            manager_id = in_manager_id
        ORDER BY
            first_name,
            last_name;
    RETURN c_direct_reports;
END;
```

```
DECLARE
    c_direct_reports SYS_REFCURSOR;
    l_employee_id employees.employee_id%TYPE;
    l_first_name employees.first_name%TYPE;
    l_last_name employees.last_name%TYPE;
    l_email      employees.email%TYPE;
BEGIN
    -- get the ref cursor from function
    c_direct_reports := get_direct_reports(46);

    -- process each employee
    LOOP
        FETCH
            c_direct_reports
        INTO
            l_employee_id,
            l_first_name,
            l_last_name,
            l_email;
        EXIT
        WHEN c_direct_reports%notfound;
            dbms_output.put_line(l_first_name || ' ' || l_last_name || ' - ' ||
        END LOOP;
    -- close the cursor
    CLOSE c_direct_reports;
END;
/
```

FUNCTION

- Una function è simile ad una procedura, con la differenza che viene definito un valore di ritorno della funzione.

```
CREATE [OR REPLACE] FUNCTION function_name
  [(parameter_name [IN | OUT | IN OUT] type [, ...])]  
RETURN type  
{IS | AS}  
BEGIN  
    function_body  
END function_name;
```



ESEMPIO DI UNA FUNZIONE

```
CREATE FUNCTION circle_area (
    p_radius IN NUMBER
) RETURN NUMBER AS
    v_pi    NUMBER := 3.1415926;
    v_area NUMBER;
BEGIN
    -- circle area is pi multiplied by the radius squared
    v_area := v_pi * POWER(p_radius, 2);

    RETURN v_area;
END circle_area;
/
```



CALL DI UNA PROCEDURE VS CALL DI UNA FUNCTION

- È possibile chiamare la procedura direttamente dalla console del DBMS

CALL <nome_procedura([lista_parametri])>;

- Mentre in JDBC

```
CallableStatement cs = null;
cs = this.con.prepareCall("{call SHOW_SUPPLIERS}");
ResultSet rs = cs.executeQuery();
```

- Per quanto riguarda la function è necessario chiamarla all'interno di una select

```
SELECT <nomefunzione([lista_parametri])>
FROM nome_tabella;
```

- Quando non è possibile definire una tabella su cui si deve effettuare la funzione, si usa la tabella **dual**, la quale rappresenta una dummy table (esclusivamente per ORACLE).



FUNCTIONS IN JDBC

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Types;
public class CallingFunctionsExample {
    public static void main(String args[]) throws SQLException {
        //Registering the Driver
        DriverManager.registerDriver(new com.mysql.jdbc.Driver());
        //Getting the connection
        String mysqlUrl = "jdbc:mysql://localhost/mydatabase";
        Connection con = DriverManager.getConnection(mysqlUrl, "root", "password");
        System.out.println("Connection established.....");
        //Preparing a CallableStatement to call a function
        CallableStatement cstmt = con.prepareCall("{? = call getDob(?)}");
        //Registering the out parameter of the function (return type)
        cstmt.registerOutParameter(1, Types.DATE);
        //Setting the input parameters of the function
        cstmt.setString(2, "Amit");
        //Executing the statement
        cstmt.execute();
        System.out.print("Date of birth: "+cstmt.getDate(1));
    }
}
```





FINE

Per eventuali domande: (in ordine di preferenza personale)

- Ora.
- Chat di Teams
- Mail: silvio.barra@unina.it

