Outline

- Embedded SQL
- Dynamic SQL
- JDBC (API)
- SQLJ (Nested)
- Stored procedures

Why Stored Procedures?

- So far, all data processing is done at the client
- Lots of data may have to be transferred
- Functionality (code) replicated at each client
- Lots of state (e.g., locks, transaction data) at the DBMS
  - While client processes the data
- Stored procedures execute in the same process space as DBMS
  - Encapsulates application logic and is close to the data
  - Reuse of common functionality by different clients
- Vendors introduced their own procedural extensions
  - e.g., Oracle’s PL/SQL

SQL/PSM

- SQL Persistent Stored Modules
  - SQL standard for stored procedures, available in SQL:2003
  - Commercial vendors may offer own extensions of PSM
- Standard language for stored procedures
  - Supports both procedures and functions
  - Functions can return results through RETURN statement
  - Procedures can return results in parameters
- In this course we focus on Oracle PL/SQL
**PL/SQL (Procedural Language SQL)**

- Procedural extension to SQL developed by Oracle
- Most prominent DBMS procedural language
- Another language is T-SQL from Microsoft (MS SQL)

- Only DML allowed in PL/SQL
  - DDL such as creating or dropping tables NOT allowed

- Basic program structure is a block
  - There can be nested blocks

- PL/SQL syntax is not case sensitive (variable names as well)

**PL/SQL Program Structure**

```plsql
DECLARE
  variable_declarations
BEGIN
  procedural_code
EXCEPTION
  error_handling
END;
```

**PL/SQL in SQL Plus**

- Ensure output goes to screen
  - SET SERVEROUTPUT ON

- Executing PL/SQL in command line
  - BEGIN
    - DBMS_OUTPUT.PUT_LINE('Hello World');
  END;

  - The `;` must be by itself on separate line
  - DBMS_OUTPUT.PUT_LINE equivalent of `printf()` in C or `System.out.println()` in Java

**Data Types**

- It is possible to use ORACLE SQL types
  - NUMBER, VARCHAR, etc

- PL/SQL allows directly referring to a column type
  - `tablename.columnname%TYPE`
  - e.g. `SAILORS.SNAME%TYPE`

- Also possible to define a row type (e.g., tuple)
  - `tablename%ROWTYPE`

- Declaring a variable: `<var_name> <TYPE>`,
  - `sailor_rec SAILORS%ROWTYPE;`

- Can later refer to individual fields using column names
  - `DBMS_OUTPUT.PUT_LINE('Name: ' || sailor_rec.name || 'Age: ' || sailor_rec.age);`

  - `||` means string concatenation (like `+` in Java)

**Assignments and Branches**

- Assignment
  - `A := B + C;`

- Branch
  - IF condition THEN statements;
  - ELSIF (condition) statements;
  - ELSIF …
  - ELSE statements;
  - END IF;

**Branch Example**

```plsql
DECLARE
  A NUMBER(6) := 10;
  B NUMBER(6);
BEGIN
  A := 23;
  B := A * 5;
  IF A < B THEN
    DBMS_OUTPUT.PUT_LINE(A || ' is less than ' || B);
  ELSE
    DBMS_OUTPUT.PUT_LINE(B || ' is less-or-equal than ' || A);
  END IF;
END;
```

- Output is: 23 is less than 115
Branch Example (2)

```plsql
DECLARE
  NGRADE NUMBER;
  LGRADE CHAR(2);
BEGIN
  NGRADE := 82.5;
  IF NGRADE > 95 THEN
    LGRADE := 'A+';
  ELSIF NGRADE > 90 THEN
    LGRADE := 'A';
  ELSIF NGRADE > 85 THEN
    LGRADE := 'B+';
  ELSIF NGRADE > 80 THEN
    LGRADE := 'B';
  ELSE
    LGRADE := 'F';
  END IF;
END;
```

Loops

```plsql
LOOP
  statements
  IF condition THEN
    EXIT WHEN condition;
  END IF;
  statements
END LOOP;
```

Loop Example

```plsql
DECLARE
  J NUMBER(6);
BEGIN
  J := 1;
  LOOP
    DBMS_OUTPUT.PUT_LINE('J=' || J);
    J := J + 1;
    EXIT WHEN J > 5;
    DBMS_OUTPUT.PUT_LINE('J=' || J);
  END LOOP;
END;
```

Loop Variants

```plsql
WHILE condition
LOOP
  various_statements
END LOOP;

FOR counter IN startvalue .. endvalue
LOOP
  various_statements
END LOOP;
```

“For For Loop” Example

```plsql
BEGIN
  FOR K IN 1..5
  LOOP
    DBMS_OUTPUT.PUT_LINE(K=' || K);
  END LOOP;
END;
```

SQL Statements

- Data can be manipulated (DML) from PL/SQL
  - SELECT must have INTO when cursors not used

```plsql
DECLARE
  SID NUMBER(6);
BEGIN
  SID := 20;
  INSERT INTO Sailors (sid,name) VALUES (SID,'Rusty');
  SID := SID + 1;
  INSERT INTO Sailors (sid,name) VALUES (SID,'Yuppy');
END;
```
As before, there are two cases

1. Single-tuple result (the "easy" case)

   ```sql
   SELECT selectfields INTO declared_variables
   FROM table_list WHERE search_criteria;
   DECLARE
   VAR_NAME Sailors.name%TYPE;
   VAR_AGE Sailors.age%TYPE;
   BEGIN
   SELECT name, age INTO VAR_NAME, VAR_AGE
   FROM Sailors WHERE SID = 10;
   DBMS_OUTPUT.PUT_LINE('Age of ' || VAR_NAME || ' is ' || VAR_AGE);
   END;
   ```

2. Multiple-tuples result: cursors are needed

   ```sql
   CURSOR cursorname IS SELECT_statement;
   OPEN cursorname;
   FETCH cursorname INTO variable_list;
   CLOSE cursorname;
   ```

### Cursor Attributes

- `%NOTFOUND`: Evaluates to TRUE when cursor has no more rows to read, FALSE otherwise
- `%FOUND`: Evaluates to TRUE if last FETCH was successful and FALSE otherwise
- `%ROWCOUNT`: Returns the number of rows that the cursor has already fetched from the database
- `%ISOPEN`: Returns TRUE if this cursor is already open, and FALSE otherwise

### Declaring a Procedure

```sql
CREATE OR REPLACE PROCEDURE procedure_name ( parameters ) IS
  variable declarations
BEGIN
  procedure_body
END;
```

Parameters can be IN, OUT or INOUT, default is IN

```sql
CREATE OR REPLACE PROCEDURE SUM_AB (A INT, B INT, C OUT INT) IS
BEGIN
  C := A + B;
END;
```
Exceptions

- Exceptions defined per block (similar to Java)
  - Each BEGIN...END has its own exception handling
  - If blocks are nested, exceptions are handled in an “inside to outside” fashion
  - If no block in the nesting handles the exception, a runtime error occurs

- There are multiple types of exceptions
  - Named system exceptions (most frequent) – we only cover these
  - Unnamed system exceptions
  - User-defined exceptions

---

Named System Exceptions

<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Reason</th>
<th>Error Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURSOR_ALREADY_OPEN</td>
<td>When you open a cursor that is already open.</td>
<td>ORA-06511</td>
</tr>
<tr>
<td>INVALID_CURSOR</td>
<td>When you perform an invalid operation on a cursor like closing a cursor or fetching data from a cursor that is not opened.</td>
<td>ORA-01001</td>
</tr>
<tr>
<td>NO_DATA_FOUND</td>
<td>When a SELECT...INTO clause does not return any row from a table.</td>
<td>ORA-01403</td>
</tr>
<tr>
<td>TOO_MANY_ROWS</td>
<td>When you SELECT or fetch more than one row into a record or variable.</td>
<td>ORA-01422</td>
</tr>
<tr>
<td>ZERO_DIVIDE</td>
<td>When you attempt to divide a number by zero.</td>
<td>ORA-01476</td>
</tr>
</tbody>
</table>

---

Exceptions

DECLARE
---
BEGIN
EXCEPTION
  WHEN ex_name1 THEN
      error handling statements
  WHEN ex_name2 THEN
      error handling statements
  ...
  WHEN Others THEN
      error handling statements
END;