Outline

- Database application programming.
  - SQL limitations.
- SQL Persistent, Stored Modules (PSM).
  - Extension of SQL.
  - PL/SQL: Oracle’s version of PSM.

DB Application Programming

- Application is written in general-purpose programming language: C, C++, Java...
  - Not in SQL!
- Application-driven database queries.
  - E.g., user makes a bid, starts an auction.
- Impedance mismatch:
  - Sets (relations) are first class objects in DBMS, but not in C, Java...
  - Vice versa for pointers, conditional statements.

Interface Solutions

1. Extend SQL with general-purpose programming: PSM.
2. Execute DB queries within application code: embedded SQL.
3. Call function from DB library: call-level interface (CLI), ODBC, JDBC.

Persistent Stored Modules

- Stored procedures as DB elements.
- Combine general-purpose programming with SQL.
- Extends functionality of DBMS.

Basic PSM Form

CREATE PROCEDURE <name> (  
  <parameters>  
)  
<declarations>  
<body>;  

For functions:
CREATE FUNCTION <name> (  
  <parameters>  
)  
RETURNS <type>;
Parameters in PSM

- For each parameter:
  - Mode: IN, OUT, INOUT
  - Name: as usual
  - Type: as usual

Examples:

- IN newprice NUMBER
- OUT oldprice NUMBER
- INOUT drinker VARCHAR[30]

Example

- A procedure to add a beer and price to Spoon's menu:
  ```sql
  CREATE PROCEDURE spoonMenu(
  IN beer VARCHAR[30],
  IN price NUMBER
)
  INSERT INTO Sells
  VALUES('Spoon', beer, price);
  ```

Invoking Procedures

- Using SQL/PSM command CALL
  ```sql
  CALL spoonMenu('BudHeavy', '7.50')
  ```

- Functions can be used in SQL expressions, provided that the return type is appropriate.

PSM Statements

- DECLARE <name> <type>;
- SET <variable> = <expression>
- BEGIN <statements> END
- RETURN <expression>
  - Does not terminate execution!

IF Statements

- Simplest form:
  ```sql
  IF <condition> THEN
  <statements>
  END IF;
  ```

- With ELSE:
  ```sql
  IF...THEN...ELSE...END IF;
  ```

- Nested:
  ```sql
  IF...THEN...ELSEIF...ELSEIF...ELSE...END IF;
  ```

Loops

- Basic form:
  ```sql
  LOOP <statements> END LOOP;
  ```

- Exiting loops:
  ```sql
  <loop name>: LOOP
  ...LEAVE <loop name>...
  END LOOP;
  ```

- Other forms:
  ```sql
  WHILE <cond> DO <stmts> END WHILE;
  ```
  ```sql
  REPEAT <stmts> UNTIL <cond> END REPEAT;
  ```
Queries in PSM

- The following rules apply to the use of queries:
  1. Queries returning a single value can be used in assignments.
  2. Queries returning a single tuple can be used with INTO.
  3. Queries returning several tuples can be used via a cursor.

Cursors

- A cursor serves as a tuple-variable that ranges over the tuples of the result of a query.

```
DECLARE c CURSOR FOR (<query>);
```

- Opening a cursor evaluates <query>.

```
OPEN c;
```

- Closed with CLOSE c;

Fetching Tuples From a Cursor

- Get next tuple:

```
FETCH c INTO a1, a2, ..., ak;
```

  - a1, a2, ..., ak are the attributes of the result of the query of c.
  - c is moved to the next tuple.
  - A cursor is used by creating a loop around FETCH.

End of Cursor

- SQL operations return status in SQLSTATE (in PSM).

```
FETCH returns '02000' in SQLSTATE when no more tuples are found.
```

- Useful declaration:

```
DECLARE NotFound CONDITION FOR SQLSTATE '02000';
```

Cursor Structure

```
DECLARE c CURSOR FOR...

cursorLoop: LOOP
...
FETCH c INTO...;
IF NotFound THEN LEAVE cursorLoop;
END IF;
...
END LOOP;
```

Cursor Example

- Write a procedure that makes free all beers sold for more than $5 at Spoon.

```
CREATE PROCEDURE FreeBeer() 
BEGIN
  DECLARE aBeer VARCHAR[30];
  DECLARE aPrice REAL;
  DECLARE NotFound CONDITION FOR SQLSTATE '02000';
  DECLARE CURSOR c FOR
    SELECT beer, price FROM Sells WHERE bar = 'Spoon';
  ...
END;
```
Example

BEGIN
OPEN c;
menuLoop: LOOP
  FETCH INTO aBeer;
  IF NotFound THEN LEAVE menuLoop END IF;
  IF aPrice > 5.00 THEN
    UPDATE Sells
    SET price = 0
    WHERE bar = 'Spoon' and beer = aBeer;
  END IF;
END LOOP;
CLOSE c;
END;

PL/SQL

- Oracle’s version of PSM (Persistent, Stored Modules).
- Use via sqlplus.
- Can be used as stored procedures or directly in sqlplus
  - This extra functionality is the “plus”.

PL/SQL Basic Structure

DECLARE
  declarations (optional)
BEGIN
  actions (required)
END;
.run

- Dot and run (or a slash in place of run;) are needed to end the statement and execute it.

Simplest Form: Sequence of Modifications

BEGIN
  INSERT INTO Likes VALUES('Leo', 'Bud');
  DELETE FROM Likes
  WHERE drinker = 'Nick' AND beer = 'Miller';
END;
.run;

Procedures

- Stored database objects that use a PL/SQL statement in their bodies.
  CREATE OR REPLACE PROCEDURE
  <name>(<arglist>) AS
    <declarations>
    BEGIN
      <PL/SQL statements>
    END;
  END;
.run;

Arguments

- Argument list has name-mode-type triples.
- Mode: IN, OUT, or IN OUT for read-only, write-only, read/write, respectively.
- Types: standard SQL + generic types like NUMBER = any integer or real type.
- Since types in procedures must match their types in the DB schema, you should generally use an expression of the form
  relation.attribute %TYPE
  to capture the type correctly.
Example

- A procedure to add a beer and price to Spoon's menu:
  
  ```sql
  CREATE PROCEDURE spoonMenu(
    b IN Sells.beer %TYPE,
    p IN Sells.price %TYPE
  ) AS
  BEGIN
    INSERT INTO Sells
    VALUES('Spoon', b, p);
  END;
  END;
  
  Note run only stores the procedure; it doesn't execute the procedure.
  ```

Invoking Procedures

- Call the procedure directly
  
  ```sql
  CALL spoonMenu('Guinness', 7.50);
  ```

- A procedure call may appear in the body of a PL/SQL statement.
  
  ```sql
  BEGIN
    spoonMenu('Bud', 5.50);
    spoonMenu('LionBlood', 12.00);
  END;
  END;
  ```

PL/SQL Constructs

- Assignments:
  
  ```sql
  <variable> := <expression>
  ```

- Variables must be declared.

- Branches
  
  ```sql
  IF <condition> THEN
  <statement(s)>
  ELSE
  <statement(s)>
  END IF;
  ```

  But in nests, use ELSIF in place of ELSEIF.

More Constructs

- Loops:
  
  ```sql
  LOOP
   . . .
  EXIT WHEN <condition>
   . . .
  END LOOP;
  ```

- For-Loops:
  
  ```sql
  FOR <var> IN <start>..<finish>
   LOOP
   . . .
  END LOOP;
  ```

Queries in PL/SQL

1. Single-row selects allow retrieval into a variable of the result of a query that is guaranteed to produce one tuple.
2. Cursors allow the retrieval of many tuples, with the cursor and a loop used to process each in turn.

Cursors in PL/SQL

- The cursor declaration is:
  
  ```sql
  CURSOR <name> IS <query>;
  ```

- Fetching is done with:
  
  ```sql
  FETCH c INTO <variables>;
  ```

- Also, with tuple variables:
  
  ```sql
  FETCH c INTO <variable>;
  ```
Example

- The FreeBeer in PL/SQL:
  CREATE OR REPLACE PROCEDURE FreeBeer() AS
  aBeer Sells.beer%TYPE;
  aPrice Sells.price%TYPE;
  CURSOR c IS
    SELECT beer, price
    FROM Sells
    WHERE bar = 'Spoon';
  BEGIN
    OPEN c;
    LOOP
      FETCH c INTO aBeer, aPrice;
      EXIT WHEN c%NOTFOUND;
      IF aPrice > 5.00 THEN
        UPDATE Sells SET price = 0
        WHERE bar = 'Spoon' AND beer = aBeer;
      END IF;
    END LOOP;
    CLOSE c;
  END;