



Databases 1



SQL/PSM and Oracle PL/SQL

SQL DDL (Data Definition Language)

- ▶ Defining a Database Schema
- ▶ Primary Keys, Foreign Keys
- ▶ Local and Global Constraints
- ▶ Defining Views
- ▶ Triggers

SQL DML (Database Modifications)

- ▶ A modification command does not return a result as a query does, but it changes the database in some way.
- ▶ There are three kinds of modifications:
 1. *Insert* a tuple or tuples.
 2. *Delete* a tuple or tuples.
 3. *Update* the value(s) of an existing tuple or tuples.

SQL in Real Programs

- ▶ We have seen only how SQL is used at the generic query interface --- an environment where we sit at a terminal and ask queries of a database.
- ▶ Reality is almost always different: conventional programs interacting with SQL.
- ▶ Persistent Stored Modules (PSM)
- ▶ PL/SQL
- ▶ Embedded SQL

Options

1. Code in a specialized language is stored in the database itself (e.g., PSM, PL/SQL).
2. SQL statements are embedded in a *host language* (e.g., C).
3. Connection tools are used to allow a conventional language to access a database (e.g., CLI, JDBC, PHP/DB).

Stored Procedures

- ▶ PSM, or “*persistent stored modules*,” allows us to store procedures as database schema elements.
- ▶ PSM = a mixture of conventional statements (if, while, etc.) and SQL.
- ▶ Lets us do things we cannot do in SQL alone.

Basic PSM Form

```
CREATE PROCEDURE <name> (  
    <parameter list> )  
    <optional local declarations>  
    <body>;
```

▶ Function alternative:

```
CREATE FUNCTION <name> (  
    <parameter list> ) RETURNS <type>
```

Parameters in PSM

- ▶ Unlike the usual name-type pairs in languages like C, PSM uses mode-name-type triples, where the *mode* can be:
 - ▶ IN = procedure uses value, does not change value.
 - ▶ OUT = procedure changes, does not use.
 - ▶ INOUT = both.

Example: Stored Procedure

- ▶ Let's write a procedure that takes two arguments b and p , and adds a tuple to **Sells(bar, beer, price)** that has bar = 'Joe's Bar', beer = b , and price = p .
 - ▶ Used by Joe to add to his menu more easily.

The Procedure

```
CREATE PROCEDURE JoeMenu (
```

```
  IN b    CHAR(20),
```

```
  IN p    REAL
```

```
)
```

```
  INSERT INTO Sells
```

```
  VALUES('Joe''s Bar', b, p);
```

Parameters are both
read-only, not changed

The body ---
a single insertion

Invoking Procedures

- ▶ Use SQL/PSM statement CALL, with the name of the desired procedure and arguments.

- ▶ **Example:**

```
CALL JoeMenu ('Moosedrool', 5.00);
```

- ▶ Functions used in SQL expressions wherever a value of their return type is appropriate.

Kinds of PSM statements – (1)

- ▶ RETURN <expression> sets the return value of a function.
 - ▶ Unlike C, etc., RETURN *does not* terminate function execution.
- ▶ DECLARE <name> <type> used to declare local variables.
- ▶ BEGIN . . . END for groups of statements.
 - ▶ Separate statements by semicolons.

Kinds of PSM Statements – (2)

- ▶ **Assignment statements:**

SET <variable> = <expression>;

- ▶ **Example:** SET b = 'Bud' ;

- ▶ **Statement labels:** give a statement a label by prefixing a name and a colon.

IF Statements

- ▶ Simplest form:
IF <condition> THEN
 <statements(s)>
END IF;
- ▶ Add ELSE <statement(s)> if desired, as
IF ... THEN ... ELSE ... END IF;
- ▶ Add additional cases by ELSEIF <statements(s)>:
IF ... THEN ... ELSEIF ... THEN ... ELSEIF ...
THEN ... ELSE ... END IF;

Example: IF

- ▶ Let's rate bars by how many customers they have, based on `Frequents(drinker,bar)`.
 - ▶ `<100` customers: 'unpopular'.
 - ▶ `100-199` customers: 'average'.
 - ▶ `>= 200` customers: 'popular'.
- ▶ Function `Rate(b)` rates bar `b`.

Example: IF (continued)

```
CREATE FUNCTION Rate (IN b CHAR(20) )
  RETURNS CHAR(10)
  DECLARE cust INTEGER;
BEGIN
  SET cust = (SELECT COUNT(*) FROM Frequents
              WHERE bar = b);
  IF cust < 100 THEN RETURN 'unpopular'
  ELSEIF cust < 200 THEN RETURN 'average'
  ELSE RETURN 'popular'
  END IF;
END;
```

Number of customers of bar b

Nested IF statement

Loops

- ▶ Basic form:

```
<loop name>: LOOP <statements>  
                END LOOP;
```

- ▶ Exit from a loop by:

```
LEAVE <loop name>
```

Example: Exiting a Loop

loop1: LOOP

...

LEAVE loop1; ← If this statement is executed . . .

...

END LOOP; ← Control winds up here

Other Loop Forms

- ▶ WHILE <condition> DO
<statements> END WHILE;
- ▶ REPEAT <statements> UNTIL
<condition> END REPEAT;

Queries

- ▶ General SELECT-FROM-WHERE queries are *not* permitted in PSM.
- ▶ There are three ways to get the effect of a query:
 1. Queries producing one value can be the expression in an assignment.
 2. Single-row SELECT . . . INTO.
 3. Cursors.

Example: Assignment/Query

- ▶ Using local variable p and **Sells(bar, beer, price)**, we can get the price Joe charges for Bud by:

```
SET p = (SELECT price FROM Sells
        WHERE bar = 'Joe''s Bar' AND
              beer = 'Bud');
```

SELECT ... INTO

- ▶ Another way to get the value of a query that returns one tuple is by placing **INTO <variable>** after the SELECT clause.
- ▶ **Example:**

```
SELECT price INTO p FROM Sells
WHERE bar = 'Joe''s Bar' AND
      beer = 'Bud';
```

Cursors

- ▶ A *cursor* is essentially a tuple-variable that ranges over all tuples in the result of some query.
- ▶ Declare a cursor *c* by:
`DECLARE c CURSOR FOR <query>;`

Opening and Closing Cursors

- ▶ To use cursor c , we must issue the command:
`OPEN c;`
 - ▶ The query of c is evaluated, and c is set to point to the first tuple of the result.
- ▶ When finished with c , issue command:
`CLOSE c;`

Fetching Tuples From a Cursor

- ▶ To get the next tuple from cursor c , issue command:

`FETCH FROM c INTO x_1, x_2, \dots, x_n ;`

- ▶ The x 's are a list of variables, one for each component of the tuples referred to by c .
- ▶ c is moved automatically to the next tuple.

Breaking Cursor Loops – (1)

- ▶ The usual way to use a cursor is to create a loop with a FETCH statement, and do something with each tuple fetched.
- ▶ A tricky point is how we get out of the loop when the cursor has no more tuples to deliver.

Breaking Cursor Loops – (2)

- ▶ Each SQL operation returns a *status*, which is a 5-digit character string.
 - ▶ For example, 00000 = “Everything OK,” and 02000 = “Failed to find a tuple.”
- ▶ In PSM, we can get the value of the status in a variable called SQLSTATE.

Breaking Cursor Loops – (3)

- ▶ We may declare a *condition*, which is a boolean variable that is true if and only if SQLSTATE has a particular value.
- ▶ **Example:** We can declare condition `NotFound` to represent 02000 by:

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

Breaking Cursor Loops – (4)

- ▶ The structure of a cursor loop is thus:

```
cursorLoop: LOOP
```

```
...
```

```
FETCH c INTO ... ;
```

```
IF NotFound THEN LEAVE cursorLoop;
```

```
END IF;
```

```
...
```

```
END LOOP;
```

Example: Cursor

- ▶ Let's write a procedure that examines **Sells(bar, beer, price)**, and raises by \$1 the price of all beers at Joe's Bar that are under \$3.
- ▶ Yes, we could write this as a simple **UPDATE**, but the details are instructive anyway.

The Needed Declarations

```
CREATE PROCEDURE JoeGouge( )
```

```
  DECLARE theBeer CHAR(20);
```


```
  DECLARE thePrice REAL;
```

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

```
  DECLARE c CURSOR FOR
```

```
    (SELECT beer, price FROM Sells  
     WHERE bar = 'Joe''s Bar');
```

Used to hold
beer-price pairs
when fetching
through cursor c




Returns Joe's menu



The Procedure Body

```
BEGIN
  OPEN c;
  menuLoop: LOOP
    FETCH c INTO theBeer, thePrice;
    IF NotFound THEN LEAVE menuLoop END IF;
    IF thePrice < 3.00 THEN
      UPDATE Sells SET price = thePrice + 1.00
      WHERE bar = 'Joe''s Bar' AND beer = theBeer;
    END IF;
  END LOOP;
  CLOSE c;
END;
```

Check if the recent
FETCH failed to
get a tuple



PL/SQL

- ▶ Oracle uses a variant of SQL/PSM which it calls PL/SQL.
- ▶ PL/SQL not only allows you to create and store procedures or functions, but it can be run from the *generic query interface* (sqlplus), like any SQL statement.
- ▶ Triggers are a part of PL/SQL.

Trigger Differences

- ▶ Compared with SQL standard triggers, Oracle has the following differences:
 1. Action is a PL/SQL statement.
 2. New/old tuples referenced automatically.
 3. Strong constraints on trigger actions designed to make certain you can't fire off an infinite sequence of triggers.
- ▶ See on-line [or-triggers.html](#) document.

SQLPlus

- ▶ In addition to stored procedures, one can write a PL/SQL statement that looks like the body of a procedure, but is executed once, like any SQL statement typed to the generic interface.
 - ▶ Oracle calls the generic interface “sqlplus.”
 - ▶ PL/SQL is really the “plus.”

Form of PL/SQL Statements

DECLARE

<declarations>

BEGIN

<statements>

END;

.

run

▶ The DECLARE section is optional.

Form of PL/SQL Procedure

CREATE OR REPLACE PROCEDURE

<name> (<arguments>) AS

<optional declarations>

← Notice AS
needed here

BEGIN

<PL/SQL statements>

END;

/

run

← Needed to store
procedure in database;
does not really run it.

PL/SQL Declarations and Assignments

- ▶ The word DECLARE does not appear in front of each local declaration.
 - ▶ Just use the variable name and its type.
- ▶ There is no word SET in assignments, and := is used in place of =.
 - ▶ Example: `x := y;`

PL/SQL Procedure Parameters

- ▶ There are several differences in the forms of PL/SQL argument or local-variable declarations, compared with the SQL/PSM standard:
 1. Order is name-mode-type, not mode-name-type.
 2. INOUT is replaced by IN OUT in PL/SQL.
 3. Several new types.

PL/SQL Types

- ▶ In addition to the SQL types, NUMBER can be used to mean INT or REAL, as appropriate.
- ▶ You can refer to the type of attribute x of relation R by $R.x\%TYPE$.
 - ▶ Useful to avoid type mismatches.
 - ▶ Also, $R\%ROWTYPE$ is a tuple whose components have the types of R 's attributes.

Example: JoeMenu

- ▶ Recall the procedure **JoeMenu(b,p)** that adds beer b at price p to the beers sold by Joe (in relation **Sells**).
- ▶ Here is the PL/SQL version.

Procedure JoeMenu in PL/SQL

```
CREATE OR REPLACE PROCEDURE JoeMenu (  
    b IN Sells.beer%TYPE,  
    p IN Sells.price%TYPE  
) AS  
BEGIN  
    INSERT INTO Sells  
    VALUES ('Joe"s Bar', b, p);  
END;  
  
/  
  
run
```

Notice these types
will be suitable
for the intended
uses of *b* and *p*.

PL/SQL Branching Statements

- ▶ Like IF ... in SQL/PSM, but:
- ▶ Use ELSIF in place of ELSEIF.
- ▶ Viz.: IF ... THEN ... ELSIF ... THEN ... ELSIF ... THEN ... ELSE ... END IF;

PL/SQL Loops

- ▶ LOOP ... END LOOP as in SQL/PSM.
- ▶ Instead of LEAVE ... , PL/SQL uses
EXIT WHEN <condition>
- ▶ And when the condition is that cursor *c* has found no tuple, we can write `c%NOTFOUND` as the condition.

PL/SQL Cursors

- ▶ The form of a PL/SQL cursor declaration is:
CURSOR <name> IS <query>;
- ▶ To fetch from cursor c, say:
FETCH c INTO <variable(s)>;

Example: JoeGouge() in PL/SQL

- ▶ Recall **JoeGouge()** sends a cursor through the Joe's-Bar portion of Sells, and raises by \$1 the price of each beer Joe's Bar sells, if that price was initially under \$3.

Example: JoeGouge() Declarations

```
CREATE OR REPLACE PROCEDURE
    JoeGouge () AS
theBeer Sells.beer%TYPE;
thePrice Sells.price%TYPE;
CURSOR c IS
    SELECT beer, price FROM Sells
    WHERE bar = 'Joe''s Bar';
```

Example: JoeGouge() Body

```
BEGIN
```

```
  OPEN c;
```

```
  LOOP
```

```
    FETCH c INTO theBeer, thePrice;
```

```
    EXIT WHEN c%NOTFOUND;
```

```
    IF thePrice < 3.00 THEN
```

```
      UPDATE Sells SET price = thePrice + 1.00;
```

```
      WHERE bar = 'Joe''s Bar' AND beer = theBeer;
```

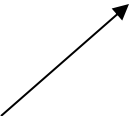

```
    END IF;
```

```
  END LOOP;
```

```
  CLOSE c;
```

```
END;
```

How PL/SQL
breaks a cursor
loop



Note this is a SET clause
in an UPDATE, not an assignment.
PL/SQL uses := for assignments.

Tuple-Valued Variables

- ▶ PL/SQL allows a variable x to have a tuple type.
- ▶ $x \text{ R}\%ROWTYPE$ gives x the type of R 's tuples.
- ▶ R could be either a relation or a cursor.
- ▶ $x.a$ gives the value of the component for attribute a in the tuple x .

Example: Tuple Type

- ▶ Repeat of JoeGouge() declarations with variable *bp* of type beer-price pairs.

```
CREATE OR REPLACE PROCEDURE
    JoeGouge () AS
CURSOR c IS
SELECT beer, price FROM Sells
WHERE bar = 'Joe''s Bar';
bp c%ROWTYPE;
```

JoeGouge() Body Using *bp*

```
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO bp;
    EXIT WHEN c%NOTFOUND;
    IF bp.price < 3.00 THEN
      UPDATE Sells SET price = bp.price + 1.00
      WHERE bar = 'Joe's Bar' AND beer =bp.beer;
    END IF;
  END LOOP;
  CLOSE c;
END;
```

Components of bp are obtained with a dot and the attribute name

