Assignment
Assign expressions to declared variables with :=.

Branches

IF <condition> THEN
   <statement(s)>
ELSE
   <statement(s)>
END IF;

- But in nests, use ELSIF in place of ELSE IF.

Loops

LOOP
   . . .
   EXIT WHEN <condition>
   . . .
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.
Single-Row Select

- Select-from-where in PL/SQL must have an INTO clause listing variables into which a tuple can be placed.
- It is an error if the select-from-where returns more than one tuple; you should have used a cursor.

Example

Find the price Joe charges for Bud (and drop it on the floor).

\[ \text{Sells(\text{bar, beer, price})} \]

```
DECLARE
  p Sells.price%TYPE;
BEGIN
  SELECT price
  INTO p
  FROM Sells
  WHERE bar = 'Joe''s Bar' AND
        beer = 'Bud';
END;
```

run
Cursors

Declare by:

\[
\text{CURSOR } \langle \text{name} \rangle \text{ IS }
\]
\[
\text{select-from-where statement}
\]

- Cursor gets each tuple from the relation produced by the select-from-where, in turn, using a *fetch statement* in a loop.
  - Fetch statement:
    \[
    \text{FETCH } \langle \text{cursor name} \rangle \text{ INTO}\n    \]
    \[
    \text{variable list;}
    \]
  - Break the loop by a statement of the form:
    \[
    \text{EXIT WHEN } \langle \text{cursor name} \rangle \%\text{NOTFOUND;}
    \]
  - True when there are no more tuples to get.
- Open and close the cursor with *OPEN* and *CLOSE*. 
Example

A procedure that examines the menu for Joe’s Bar and raises by $1.00 all prices that are less than $3.00.

\[ \text{Sells(bar, beer, price)} \]

- This simple price-change algorithm can be implemented by a single \texttt{UPDATE} statement, but more complicated price changes could not.
CREATE PROCEDURE joeGouge() AS
    theBeer Sells.beer%TYPE;
    thePrice Sells.price%TYPE;
    CURSOR c IS
        SELECT beer, price
        FROM Sells
        WHERE bar = 'Joe''s bar';
    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;
.
run
Row Types

Anything (e.g., cursors, table names) that has a tuple type can have its type captured with %ROWTYPE.

- We can create temporary variables that have tuple types and access their components with dot.
- Handy when we deal with tuples with many attributes.
Example

The same procedure with a tuple variable bp.

```
CREATE PROCEDURE joeGouge() AS
    CURSOR c IS
        SELECT beer, price
        FROM Sells
        WHERE bar = 'Joe's bar';
    bp c%ROWTYPE;
    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;
    .
run
```
**SQL2 Embedded SQL**

Add to a conventional programming language (C in our examples) certain statements that represent SQL operations.

- Each embedded SQL statement introduced with `EXEC SQL`.
- Preprocessor converts C + SQL to pure C.
  - SQL statements become procedure calls.
Shared Variables

A special place for C declarations of variables that are accessible to both SQL and C.

- Bracketed by
  
  \texttt{EXEC SQL BEGIN/END DECLARE SECTION;}

- In Oracle Pro/C (not C++) the “brackets” are optional.

- In C, variables used normally; in SQL, they must be preceded by a colon.
Example

Find the price for a given beer at a given bar.

Sells(bar, beer, price)

EXEC SQL BEGIN DECLARE SECTION;
    char theBar[21], theBeer[21];
    float thePrice;
EXEC SQL END DECLARE SECTION;

    /* assign to theBar and theBeer */

EXEC SQL SELECT price
    INTO :thePrice
FROM Sells
WHERE beer = :theBeer AND
    bar = :theBar;

    ...
Cursors

Similar to PL/SQL cursors, with some syntactic differences.

Example

Print Joe’s menu.

Sells(bar, beer, price)

EXEC SQL BEGIN DECLARE SECTION;
  char theBeer[21];
  float thePrice;
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE c CURSOR FOR
  SELECT beer, price
  FROM Sells
  WHERE bar = 'Joe’s Bar';
EXEC SQL OPEN CURSOR c;
while(1) {
  EXEC SQL FETCH c
  INTO :theBeer, :thePrice;
  if(NOT FOUND) break;
  /* format and print beer and price */
}
EXEC SQL CLOSE CURSOR c;
Oracle Vs. SQL2 Features

- SQL2 expects `FROM` in fetch-statement.
- SQL2 defines an array of characters `SQLSTATE` that is set every time the system is called.
  - Errors are signaled there.
  - A failure for a cursor to find any more tuples is signaled there.
  - However, Oracle provides us with a header file `sqlca.h` that declares a `communication area` and defines macros to access it.
  - In particular, `NOT FOUND` is a macro that says “the no-tuple-found signal was set.”
Dynamic SQL

Motivation:

- Embedded SQL is fine for fixed applications, e.g., a program that is used by a sales clerk to book an airline seat.

- It fails if you try to write a program like sqlplus, because you have compiled the code for sqlplus before you see the SQL statements typed in response to the SQL> prompt.

- Two special statements of embedded SQL:
  - PREPARE turns a character string into an SQL query.
  - EXECUTE executes that query.
Example: Sqlplus Sketch

```sql
EXEC SQL BEGIN DECLARE SECTION;
    char query[MAX_QUERY_LENGTH];
EXEC SQL END DECLARE SECTION;
/* issue SQL> prompt */
/* read user’s text into array query */
EXEC SQL PREPARE q FROM :query;
EXEC SQL EXECUTE q;
/* go back to reissue prompt */
```

- Once prepared, a query can be executed many times.
  - “Prepare” = optimize the query, e.g., find a way to execute it using few disk-page I/O’s.
- Alternatively, PREPARE and EXECUTE can be combined into:

  ```sql
  EXEC SQL EXECUTE IMMEDIATE :query;
  ```