ISYS 365 -
Records & Cursors
Agenda

- Summary of last class
  - Loops
    - Simple Loops
    - WHILE Loops
    - FOR Loops
  - Records
- Cursors
Records

- User-defined composite types

- Provides a way to treat separate but logically related variables as a unit

- Similar to “structures” in C

- Use the dot notation to refer to fields within a record
  - `v_StudentInfo.FirstName := 'John';`

- In order to copy one record to another record, both records must be of the same record type
Illegal Assignment

REM This block shows legal and illegal record assignments.

DECLARE
  TYPE t_Rec1Type IS RECORD (
    Field1 NUMBER,
    Field2 VARCHAR2(5));

  TYPE t_Rec2Type IS RECORD (
    Field1 NUMBER,
    Field2 VARCHAR2(5));

  v_Rec1 t_Rec1Type;
  v_Rec2 t_Rec2Type;

BEGIN
  /* Even though v_Rec1 and v_Rec2 have the same field names
     and field types, the record types themselves are different.
     This is an illegal assignment which raises PLS-382. */

  v_Rec1 := v_Rec2;

  /* However, the fields are the same type, so the following
     are legal assignments. */

  v_Rec1.Field1 := v_Rec2.Field1;
  v_Rec2.Field2 := v_Rec2.Field2;
END;
/
Records

- Syntax:
  
  DECLARE
    TYPE t_StudentRecord IS RECORD(
      Student_ID     NUMBER(5),
      FirstName      VARCHAR2(20),
      LastName       VARCHAR2(20)
    );

    v_StudentInfo t_StudentRecord;

    BEGIN
      -- Process data here
    EXCEPTION
      -- Error handling would go here
    END;

    /

Use the dot notation to refer to fields within a record.

Example:

v_StudentInfo.FirstName := 'John';
Records

- To declare a record with the same structure (i.e. fields & field types) as a database row, use `%ROWTYPE`

- Syntax:
  ```sql
  DECLARE
    v_StudentInfo student% ROWTYPE;
  BEGIN
    -- Process data here
  EXCEPTION
  -- Error handling would go here
  END;
  /
  ```

**NOTE:**
Any NOT NULL constraints that are defined for a column within the table will not be applicable to the declared record when `%ROWTYPE` is used.
Cursors

- A pointer to the context area

  Context area contains:
  - Number of rows processed by the statement
  - A pointer to the parsed representation of the statement
  - In the case of a query, the set of rows returned by the query (i.e. the active set, active recordset)

- Follows standard declaration and scoping

- Naming Convention: `c_CursorName`

- Cursor Types:
  - Explicit: user-defined
  - Implicit: system-defined
Explicit Cursors

To use explicit cursors...
- Declare the cursor
- Open the cursor
- Fetch the results into PL/SQL variables
- Close the cursor
Declaring Cursors

DECLARE
  v_StudentID students.id%TYPE;
  v_FirstName students.first_name%TYPE;
  v_LastName students.last_name%TYPE;

CURSOR c_HistoryStudents IS
  SELECT id, first_name, last_name
  FROM students
  WHERE major = 'History';
BEGIN
  -- open cursor, fetch records & then close cursor here
END;
/

DECLARE

    v_StudentID    students.id%TYPE;
    v_FirstName   students.first_name%TYPE;
    v_LastName    students.last_name%TYPE;

CURSOR c_HistoryStudents IS
    SELECT id, first_name, last_name
    FROM students
    WHERE major = 'History';
BEGIN

    OPEN c_HistoryStudents;
    -- fetch records & then close cursor here
    END;
/

OPEN Cursor
FETCH Records

DECLARE
  v_StudentID     students.id%TYPE;
  v_FirstName     students.first_name%TYPE;
  v_LastName      students.last_name%TYPE;

CURSOR c_HistoryStudents IS
  SELECT id, first_name, last_name
  FROM students
  WHERE major = 'History';

BEGIN
  OPEN c_HistoryStudents;
  LOOP
    FETCH c_HistoryStudents INTO v_StudentID, v_FirstName, v_LastName;
    EXIT WHEN c_HistoryStudents%NOTFOUND;
    -- do something with the values that are now in the variables
  END LOOP
  -- close cursor here
END;/

DECLARE
  v_StudentID  students.id%TYPE;
  v_FirstName  students.first_name%TYPE;
  v_LastName   students.last_name%TYPE;

CURSOR c_HistoryStudents IS
  SELECT id, first_name, last_name
  FROM students
  WHERE major = 'History';
BEGIN
  OPEN c_HistoryStudents;
  LOOP
    FETCH c_HistoryStudents INTO v_StudentID, v_FirstName, v_LastName;
    DBMS_OUTPUT.PUT_LINE(v_StudentID, v_FirstName, v_LastName);
    EXIT WHEN c_HistoryStudents%NOTFOUND;
    -- do something with the values that are now stored in the variables
  END LOOP
  CLOSE c_HistoryStudents;
END; /
Cursor Attributes

Cursors have four attributes…

- %FOUND
  - TRUE if the previous FETCH returned a row
  - Otherwise, FALSE

- %NOTFOUND
  - TRUE if the previous FETCH did NOT return a row
  - Otherwise, FALSE

- %ISOPEN
  - TRUE if the cursor is open,
  - Otherwise, FALSE

- %ROWCOUNT
  - Returns the # of rows that have been fetched by the cursor so far
DECLARE
    CURSOR c_HistoryStudents IS
        SELECT id, first_name, last_name
        FROM students
        WHERE major = 'History';
    v_StudentData  c_HistoryStudents%ROWTYPE;
BEGIN
    OPEN c_HistoryStudents;
    FETCH c_HistoryStudents INTO v_StudentData;

    WHILE c_HistoryStudents%FOUND LOOP
        INSERT INTO registered_students (student_id, department, course)
            VALUES (v_StudentData.ID, 'HIS', 301);
        INSERT INTO temp_table (num_col, char_col)
            VALUES (v_StudentData.ID,
                    v_StudentData.first_name || ' ' ||
                    v_StudentData.last_name);
        FETCH c_HistoryStudents INTO v_StudentData;
    END LOOP;
    CLOSE c_HistoryStudents;
END;
Take Home Exercise #1

Write an anonymous PL/SQL block that...

- Defines a cursor that points to a record set that contains the sailors’ names, reservation date and boat id where the boat color is red
- Opens the cursor
- Uses a simple loop to fetch each record in the active set
- Displays each last name, reservation date and boat id for each record to the output screen

- **Schema**
  
  Sailor (sid, sname, rating, age)

  Boat (bid, bname, color)

  Reservation(sid, bid, day)
DECLARE
  v_StudentID students.id%TYPE;
  v_FirstName students.first_name%TYPE;
  v_LastName students.last_name%TYPE;
  v_Major students.major%TYPE;
CURSOR c_HistoryStudents IS
  SELECT id, first_name, last_name
  FROM students
  WHERE major = v_Major;
BEGIN
  v_Major := 'History';
  OPEN c_HistoryStudents;
  LOOP
    FETCH c_HistoryStudents INTO v_StudentID, v_FirstName, v_LastName;
    EXIT WHEN c_HistoryStudents%NOTFOUND;
    -- do something with the values that are now stored in the variables
  END LOOP
  CLOSE c_HistoryStudents;
END; /
What are bind variables?

- Variables that are referenced in the cursor declaration

- They must be declared BEFORE the cursor is declared
  - i.e. variable must be declared before it can be used

- The values of bind variables are examined ONLY when the cursor is opened (at run time)
Explicit Cursors with Bind Variables

To use explicit cursors with bind variables…

- Declare bind variables
- Then declare the cursor
- Assign values to bind variables
- Open the cursor
- Fetch the results into PL/SQL variables
- Close the cursor
Take Home Exercise #2

- Modify your answer for Class Exercise #1 such that...
  - It uses bind variables for color of boat instead of hard-coding the values in the WHERE clause
  - The FETCH is done inside of a WHILE loop instead of inside of a simple loop.
Implicit Cursors

Used for INSERT, UPDATE, DELETE and SELECT...INTO queries

- In SQL% NOTFOUND, SQL is called the implicit cursor
- PL/SQL opens & closes implicit cursors, which is also called SQL cursor
- You don’t declare the implicit cursor

- If the WHERE clause fails...
  - For SELECT...INTO statement, then NO_DATA_FOUND error is raised instead of SQL% NOTFOUND
  - For UPDATEs and DELETEs, SQL% NOTFOUND is set to TRUE
Example of Implicit Cursor

BEGIN

    UPDATE rooms
        SET number_seats = 100
        WHERE room_id = 99980;

    -- If the previous UPDATE statement didn't match any rows,
    -- insert a new row into the rooms table.
    IF SQL% NOTFOUND THEN
        INSERT INTO rooms (room_id, number_seats)
            VALUES (99980, 100);
    END IF;

END;
/

Example of Implicit Cursor

BEGIN

UPDATE rooms

    SET number_seats = 100
    WHERE room_id = 99980;

-- If the previous UPDATE statement didn't match any rows,

-- insert a new row into the rooms table.

IF SQL%ROWCOUNT = 0 THEN

    INSERT INTO rooms (room_id, number_seats)
    VALUES (99980, 100);

END IF;

END;
DECLARE
    -- Record to hold room information.
    v_RoomData rooms%ROWTYPE;
BEGIN
    -- Retrieve information about room ID -1.
    SELECT *
    INTO v_RoomData
    FROM rooms
    WHERE room_id = -1;

    -- The following statement will never be executed, since
    -- control passes immediately to the exception handler.
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND is true!');
    END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('NO_DATA_FOUND raised!');
    END;
/
SHOW ERRORS

- To display error message
- SQL> SHOW ERRORS;
Agenda

- Exceptions & Exception Handling
Summary of last class

- Cursors
  - A pointer to the context area (active set)
  - Name begins with c_
  - Defined within the DECLARE section
  - Types: Explicit vs. Implicit
    - Explicit: (1) Declare, (2) Open, (3) Fetch & (4) Close
  - Bind variables
    - Variables that are referenced in the cursor declaration
    - Must be defined BEFORE the cursor
    - Values examined ONLY at run time

```sql
CURSOR c_HistoryStudents IS
  SELECT id, first_name, last_name
  FROM students
  WHERE major = 'History';
```
DECLARE
  CURSOR c_Reservations IS
    SELECT s.sname, r.day, r.bid
    FROM Sailor S, Reserve R, Boat B
    WHERE R.sid = s.sid
         AND R.bid = b.bid
         AND B.color = 'red';
  v_Reservation c_Reservations%ROWTYPE;
BEGIN
  OPEN c_Reservations;
  LOOP
    FETCH c_Reservations INTO v_Reservation;
    EXIT WHEN c_Reservations%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_Reservation.sname||' '||v_Reservation.day||' '||v_Reservation.bid);
  END LOOP;
  CLOSE c_Reservations;
END;/

---

### Sailor

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

### Boat

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>

### Reserve

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/7/98</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>9/5/98</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>9/8/98</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>9/8/98</td>
</tr>
</tbody>
</table>
Cursor Attributes

- Cursors have four attributes...
  - %FOUND
    - TRUE if the previous FETCH returned a row
    - Otherwise, FALSE
  - %NOTFOUND
    - TRUE if the previous FETCH did NOT return a row
    - Otherwise, FALSE
  - %ISOPEN
    - TRUE if the cursor is open,
    - Otherwise, FALSE
  - %ROWCOUNT
    - Returns the # of rows that have been fetched by the cursor so far
Implicit Cursors

- Used for INSERT, UPDATE, DELETE and SELECT...INTO queries

  - In SQL% NOTFOUND, SQL is called the implicit cursor
  - PL/SQL opens & closes implicit cursors, which is also called SQL cursor
  - You don’t declare the implicit cursor

- If the WHERE clause fails...
  - For SELECT...INTO statement, then NO_DATA_FOUND error is raised instead of SQL% NOTFOUND
  - For UPDATEs and DELETEs, SQL% NOTFOUND is set to TRUE
Exceptions & Exception Handling

- What are exceptions & exception handlers?
  - The method by which the program reacts & deals with runtime errors

- How do they work?
  - When a runtime error occurs, an exception is raised
  - Then control is passed to the exception handler (i.e. the EXCEPTION section)
  - Once control is passed to the exception handler, there is no way to return to the executable section
Declaring (Explicit) Exceptions

How are explicit exceptions declared?
- Defined within the DECLARE section
- Defined using the keyword data type EXCEPTION
- Name starts with e_

Example:
DECLARE
    e_TooManyStudents EXCEPTION;
    v_CurrentStudents NUMBER(3);
    v_MaxStudents NUMBER(3);
BEGIN
    -- process data here
EXCEPTION
    -- handle exceptions here
END;/
Raising Exceptions

How are exceptions used?
- Within the executable section
  - Test a condition
  - If the condition evaluates to true, then use the keyword `RAISE` to raise an exception
  - Can use the `RAISE` keyword with either predefined exceptions or user-defined exceptions
DECLARE
   e_TooManyStudents EXCEPTION;
   v_CurrentStudents NUMBER(3);
   v_MaxStudents NUMBER(3);
BEGIN
   SELECT current_students, max_students
   INTO v_CurrentStudents, v_MaxStudents
   FROM classes
   WHERE department = 'HIS' AND course = 101;

   IF v_CurrentStudents > v_MaxStudents THEN
      RAISE e_TooManyStudents;
   END IF;

EXCEPTION
   -- handle exceptions here
END;/

Raising Exceptions
Handling Exceptions

- Syntax

DECLARE
e_TooManyStudents EXCEPTION;
BEGIN
    -- process data here
EXCEPTION
    WHEN exception_Name1 THEN
        statements;
    WHEN exception_Name2 THEN
        statements;
    WHEN OTHERS THEN
        statements;
END;/

- An exception can be handled by at most one handler!
DECLARE
    e_TooManyStudents EXCEPTION;
    v_CurrentStudents NUMBER(3);
    v_MaxStudents NUMBER(3);
BEGIN
    SELECT current_students, max_students
    INTO v_CurrentStudents, v_MaxStudents
    FROM classes
    WHERE department = 'HIS' AND course = 101;

    IF v_CurrentStudents > v_MaxStudents THEN
        RAISE e_TooManyStudents;
    END IF;
EXCEPTION
    WHEN e_TooManyStudents THEN
        INSERT INTO log_table (info)
        VALUES ('History 101 has ' || v_CurrentStudents || ' students: max allowed is ' || v_MaxStudents);
    WHEN OTHERS THEN
        INSERT INTO log_table (info) VALUES ('Another error occurred');
END;/
Handling Exceptions

**Built-in Functions**

- **SQLCODE**
  - Returns the error code associated with the error
  - Returns a value of **1 for user-defined exception**
  - Returns a value of **0 if no error** with the last executed statement

- **SQLERRM**
  - Returns the text of the error message
  - Maximum length of an Oracle message is 512 characters
  - Returns “User-defined Exception” for user-defined exception

- **RAISE_APPLICATION_ERROR**
  - RAISE_APPLICATION_ERROR(error#, error_message);
  - Valid error #s: -20,000 and -20,999
  - Error_Message MUST be less than 512 characters
Take Home Exercise 3

Write an anonymous PL/SQL block that...

- Defines a cursor that uses a bind variable that points to a record set that contains the sailors’ IDs, sailors’ names, reservation day and boat id where the boat color is red
- Uses a WHILE loop to fetch data into a cursor variable
- Use the cursor to select the rating of the sailor
- Displays each last name, rating, reservation date and boat id for each record to the output screen
- Uses exception handles
  - If sailor does not exist: invalid_sailor
  - If rating is <8: low_rating

<table>
<thead>
<tr>
<th>Boat</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bid</td>
<td>bname</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sailor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reserve</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>bid</td>
</tr>
<tr>
<td>22</td>
<td>101</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
</tr>
</tbody>
</table>
ISYS 365 - Putting It All Together
Agenda

- Key Concepts (thus far)
- Examples
- Take Home Exercises
Key Concepts (thus far)

- PL/SQL Block
- IF-THEN-ELSE
- CASE
- Loops
  - Simple Loops
  - WHILE Loops
  - FOR Loops
- Records
  - Explicit
  - Implicit
- Cursors
  - Explicit
  - Implicit
- Exception Handling
- Naming Conventions
Key Concepts: PL/SQL Block

- Basic building block/unit of PL/SQL programs
  - Three possible sections of a block
    - Declarative section (optional)
    - Executable section (required)
      - Delimiters: BEGIN, END
    - Exception handling (optional)
- A block performs a logical unit of work in the program
- Blocks can be nested
Key Concepts: IF-THEN-ELSE & CASE

Either specify test case after CASE keyword & OR specify test after WHEN keyword

IF boolean_expression1 THEN
  sequence_of_statements;
[ELSIF boolean_expression2 THEN
  sequence_of_statements]
[ELSE
  sequence_of_statements]
END IF;

CASE
  WHEN boolean_expression1 THEN
    sequence_of_statements;
  WHEN boolean_expression2 THEN
    sequence_of_statements;
  ELSE
    sequence_of_statements;
END CASE;
Key Concepts: PL/SQL Loops

- Used to execute a sequence of statements repeatedly
- When the number of iterations is unknown
  - Simple loops: executes at least once
  - WHILE loops: executes while the condition is true
- When the number of iterations is known in advance
  - Numeric FOR Loops: executes a specific number of times

**Simiple Loop**
```
LOOP
    sequence_of_statements;
    EXIT WHEN condition;
END LOOP;
```

**While Loop**
```
WHILE condition LOOP
    sequence_of_statements;
END LOOP;
```

**For Loop**
```
FOR loop_Counter IN IN [REVERSE] low..high LOOP
    sequence_of_statements;
END LOOP;
```
Key Concepts: Records

Records

- Explicit
  - Name begins with t_
  - Once declared, can be used to declare other variables
  - TYPE t_StudentRecord IS RECORD(
    Student_ID NUMBER(5),
    FirstName VARCHAR2(20),
    LastName VARCHAR2(20));

    v_StudentInfo t_StudentRecord;

- Implicit
  - %ROWTYPE
    - Declares a record with the same structure as
      v_StudentInfo student%ROWTYPE
  - Use dot notation to refer to fields within record
Key Concepts: Cursors

- Cursors
  - A pointer to the context area (active set)
  - Name begins with c_
  - Types: Explicit vs. Implicit
    - Explicit: (1) Declare, (2) Open, (3) Fetch & (4) Close
  - Bind variables
    - Variables that are referenced in the cursor declaration
    - Must be defined BEFORE the cursor
    - Values examined ONLY at run time

```sql
CURSOR c_HistoryStudents IS
    SELECT id, first_name, last_name
FROM students
WHERE major = 'History';
```
Key Concepts: Cursors

- Explicit for SELECT statement
- Implicit for all other DML statements
  - Used for INSERT, UPDATE, DELETE and SELECT...INTO queries
  - PL/SQL opens & closes implicit cursors, which is called SQL cursor
- If the WHERE clause fails...
  - For SELECT...INTO statement, then NO_DATA_FOUND error is raised instead of SQL%NOTFOUND
  - For UPDATEs and DELETEs, SQL%NOTFOUND is set to TRUE
- Four attributes: %FOUND, %NOTFOUND, %ISOPEN, %ROWCOUNT
Key Concepts: Exceptions & Exception Handling

- The method by which the program reacts & deals with runtime errors
- When a runtime error occurs, an exception is raised & control passes to the EXCEPTION section
- Once control is passed to the exception handler, there is no way to return to the executable section

User-defined exceptions
- Defined using the keyword data type EXCEPTION
- Use the keyword RAISE to raise an exception
Key Concepts: Exceptions & Exception Handling

- Pre-defined exceptions
  - NO_DATA_FOUND
    - no data found in SELECT...INTO
  - TOO_MANY_ROWS
    - SELECT...INTO produces more than one row
  - INVALID_CURSOR
    - Cursor already closed
  - CURSOR_ALREADY_OPEN
    - Cursor already open
  - ZERO_DIVIDE
    - Division by zero
  - INVALID_NUMBER
    - Data is not numeric
  - (see book for others)
Key Concepts: Exceptions & Exception Handling

- Built-in Functions
  - SQLCODE
    - Returns the error code associated with the error
    - Returns a value of 1 for user-defined exception
    - Returns a value of 0 if no error with the last executed statement
  - SQLERRM
    - Returns the text of the error message
    - Maximum length of an Oracle message is 512 characters
    - Returns “User-defined Exception” for user-defined exception
  - RAISE_APPLICATION_ERROR
    - RAISE_APPLICATION_ERROR(error#, error_message);
    - Valid error #s: -20,000 and -20,999
    - Error_Message MUST be less than 512 characters
    - Used in procedures and functions
Key Concepts: Exceptions & Exception Handling

- **Syntax**

  DECLARE
  
  e_TooManyStudents EXCEPTION;

  BEGIN
  
  -- process data here

  EXCEPTION
  
  WHEN exception_Name1 THEN
    statements;

  WHEN exception_Name2 THEN
    statements;

  WHEN OTHERS THEN
    statements;

  END;/

- An exception can be handled by at most one handler!
### Key Concepts: Naming Conventions

<table>
<thead>
<tr>
<th>Item</th>
<th>Naming Convention</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary keys</td>
<td>*_pk</td>
<td>* = tablename</td>
</tr>
<tr>
<td>foreign keys</td>
<td>*__fk1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*__fk2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*__fk#</td>
<td>* = tablename; # = a sequential number</td>
</tr>
<tr>
<td>unique keys</td>
<td>*__u1</td>
<td>* = tablename</td>
</tr>
<tr>
<td></td>
<td>*__u2</td>
<td>* = tablename</td>
</tr>
<tr>
<td></td>
<td>*__u#</td>
<td># represents a sequential number</td>
</tr>
<tr>
<td>checks</td>
<td>*__ck1</td>
<td>* = tablename</td>
</tr>
<tr>
<td></td>
<td>*__ck2</td>
<td>* = tablename</td>
</tr>
<tr>
<td></td>
<td>*__ck#</td>
<td># represents a sequential number</td>
</tr>
<tr>
<td>sequences</td>
<td>*__sequence</td>
<td>* = field name</td>
</tr>
<tr>
<td>script files</td>
<td>*.sql</td>
<td>* can be any name you choose</td>
</tr>
<tr>
<td>spooled files</td>
<td>*.lst</td>
<td>* can be any name you choose (e.g. TEST.LST)</td>
</tr>
<tr>
<td>cursors</td>
<td>c_</td>
<td></td>
</tr>
<tr>
<td>exceptions</td>
<td>e_</td>
<td></td>
</tr>
<tr>
<td>records</td>
<td>t_</td>
<td>Explicit records</td>
</tr>
<tr>
<td>variables</td>
<td>v_</td>
<td></td>
</tr>
</tbody>
</table>
Take Home Exercise #1

- Write an anonymous PL/SQL block that...
  - Defines a cursor that points to a record set that contains the sailors’ last names, reservation date and boat id where the boat color is red
  - Opens the cursor
  - Uses a simple loop to fetch each record in the active set
  - Displays each last name, reservation date and boat id for each record to the output screen

<table>
<thead>
<tr>
<th>Boat</th>
</tr>
</thead>
<tbody>
<tr>
<td>bid</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>102</td>
</tr>
<tr>
<td>103</td>
</tr>
<tr>
<td>104</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sailor</th>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reserve</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>10/7/98</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>9/5/98</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>9/8/98</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>9/8/98</td>
<td></td>
</tr>
</tbody>
</table>
Take Home Exercise 2

- Modify Exercise #1 such that...
  - It uses bind variables for color instead of hard-coding the values in the WHERE clause
  - The FETCH is done inside of a WHILE loop instead of inside of a simple loop.
Take Home Exercise 3

- Write an anonymous PL/SQL block that...
  - Defines a cursor that uses a bind variable that points to a record set that contains the sailors’ IDs, sailors’ names, reservation day and boat id where the boat color is red
  - Uses a WHILE loop to fetch data into a cursor variable
  - Use the cursor to select the rating of the sailor
  - Displays each last name, rating, reservation date and boat id for each record to the output screen
  - Uses exception handles
    - If sailor does not exist: invalid_sailor
    - If rating is <8: low_rating

<table>
<thead>
<tr>
<th>Boat</th>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sailor</th>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>65.5</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td></td>
<td>74</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reserve</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>104</td>
<td>10/7/98</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>101</td>
<td>9/5/98</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>102</td>
<td>9/8/98</td>
</tr>
<tr>
<td></td>
<td>74</td>
<td>103</td>
<td>9/8/98</td>
</tr>
</tbody>
</table>