PLSQL Basics II

KIT712

Announcements

- **Test 2**: SQL Queries and Optimisations after midsemester break
  - Lectures (SQL Queries, Database Security ,SQL Optimisations)
  - Tutorial (SQL Queries and SQL Optimisation)
  - Question Types & Duration will be sent by email
  - Mock test will be available by Mylo
  - Will be using livesql website.
- **Marks for assignment and Test 1** will be finalised by this Friday. General **comments** will be discussed during the lectures. For specific comments, you need to meet your tutor and discuss

Assignment: Some General Comments

Business Rules & Reflection Topic: AWS Marketplace

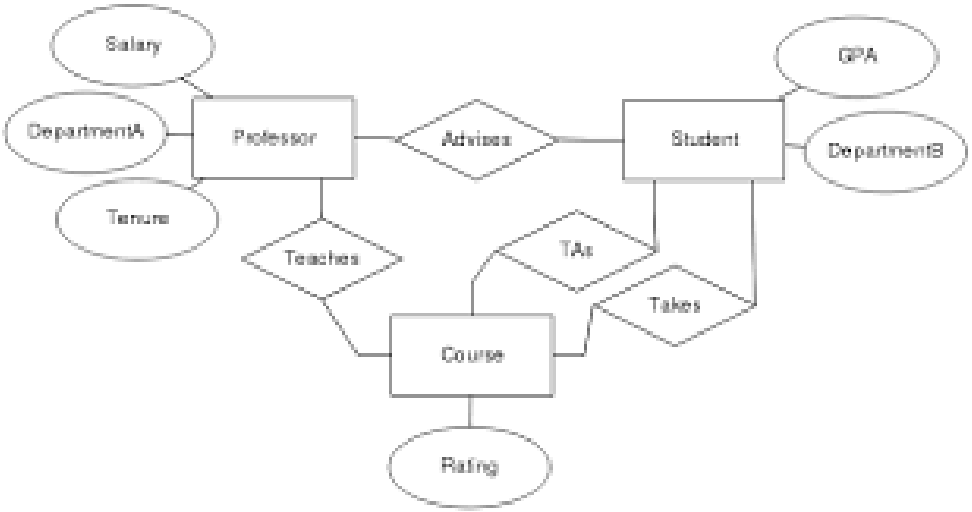
Business Rules

A) If business scenario attempted is given topic, you will get full marks is correct
B) If business scenario on any other topic, you will get ZERO(0)

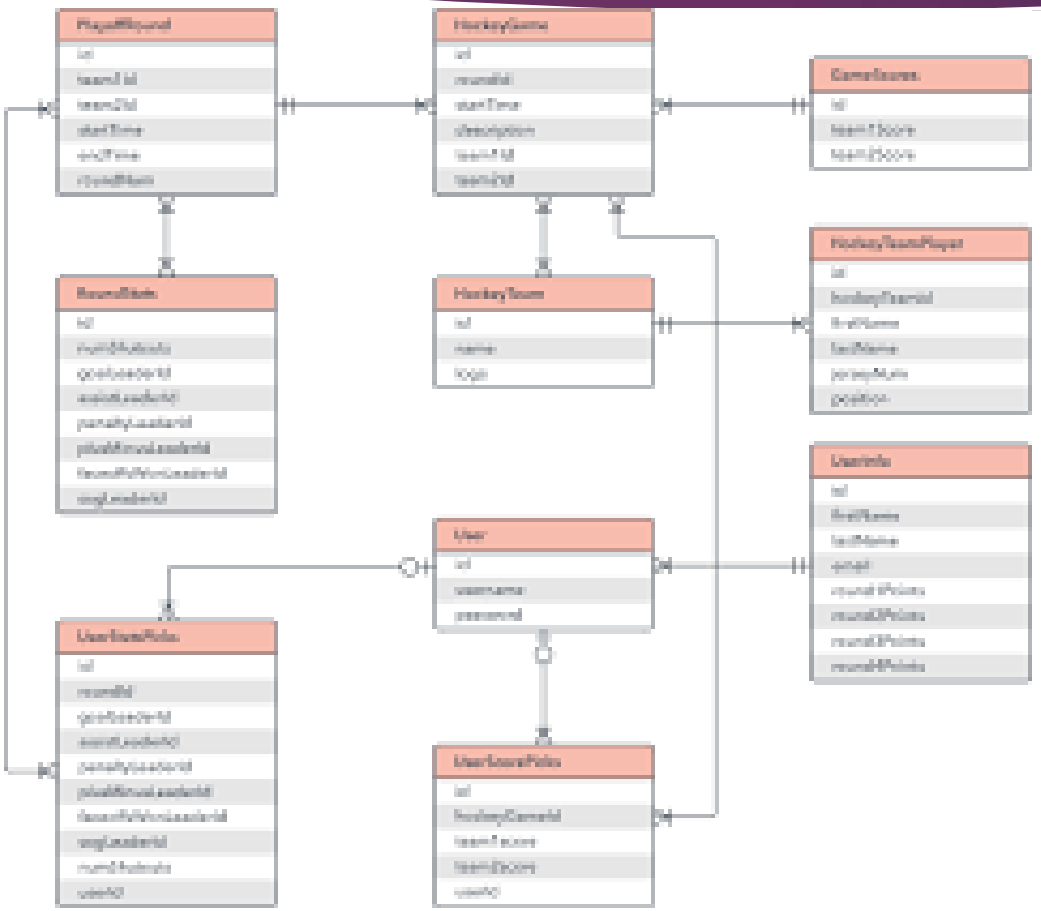
Reflection

a) If correct topic is used to write business rules and compared with unit coordinator as required by marking scheme, you will get full marks.
b) If incorrect topic is used and one recognised it during reflection part, you will get some marks
c) If incorrect topic is used for writing business rules and not even recognised this while comparing UC given business rules, you will get ZERO (0)

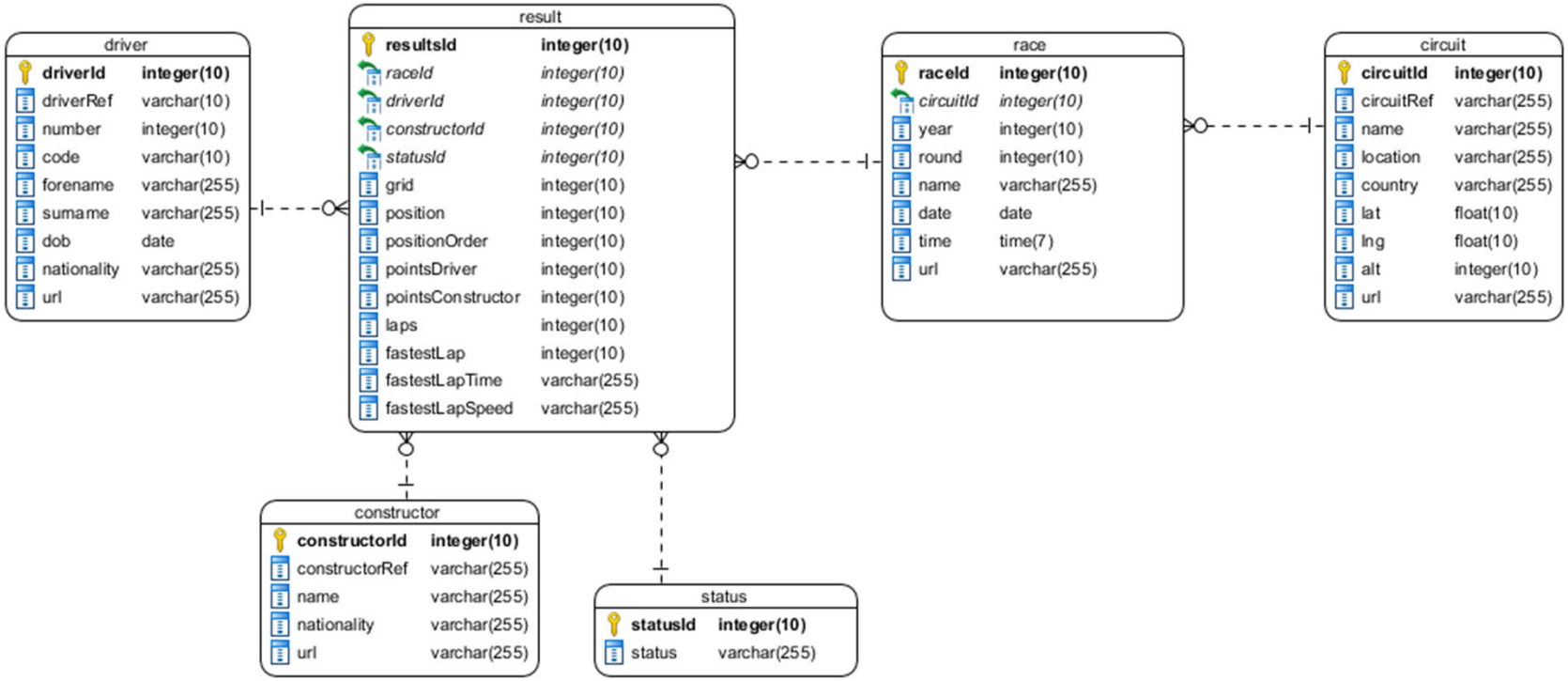
Some Incorrect Conventions



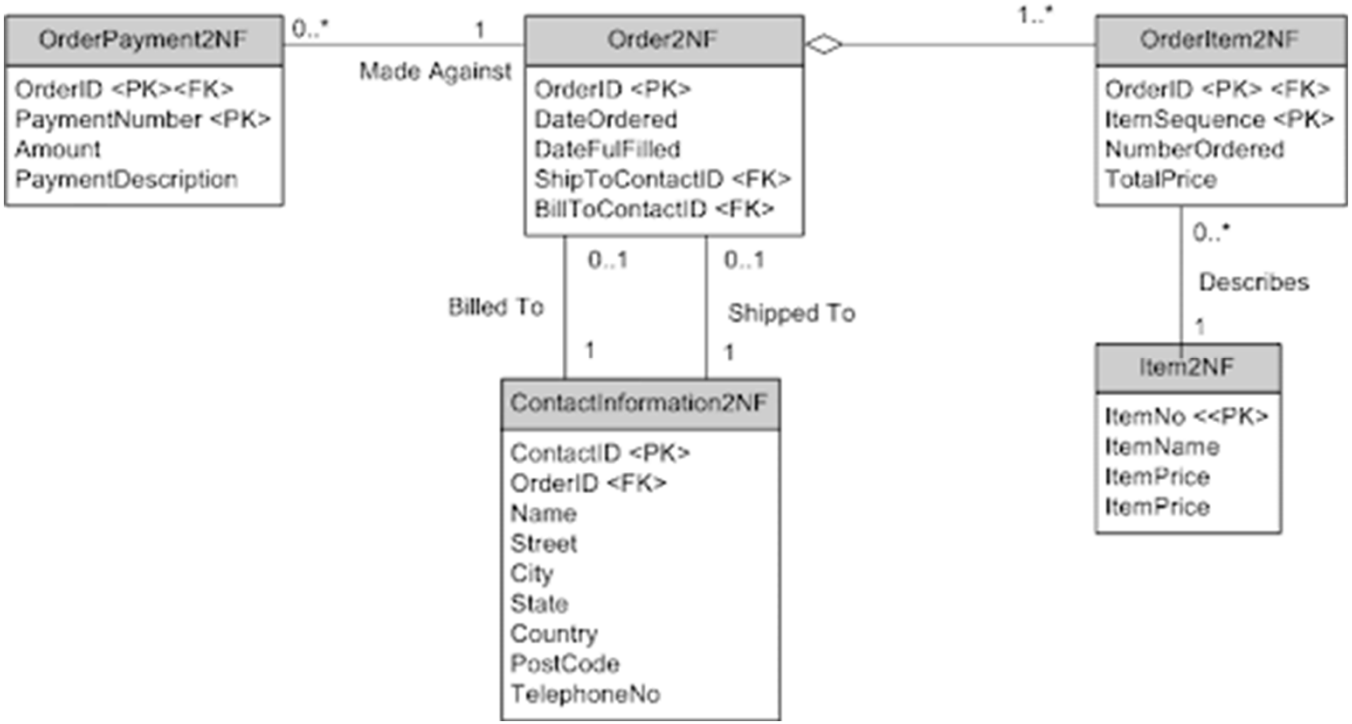
Some Incorrect Conventions



Some Incorrect Conventions



Some Incorrect Conventions



Correct Conventions??

PL/SQL

PL/SQL Block Structure

- **DECLARE** – create variables, cursors, and types
- **BEGIN** – SQL, logic, loops, assignment statements
- **EXCEPTION** – error handling
- **END** – close the block

Executing a Block with Errors

- Common Errors
  - Use = rather than :=
  - Not declaring a variable
  - Misspelling a variable name
  - Not ending a statement with ;
  - No data returned from a SELECT statement

Scalar Variables

multiplication

```
DECLARE
  lv_taxrate_num CONSTANT NUMBER(2,2) := .06;
  lv_total_num NUMBER(6,2) := 50;
  lv_taxamt_num NUMBER(4,2);
BEGIN
  lv_taxamt_num := lv_total_num * lv_taxrate_num;
  DBMS_OUTPUT.PUT_LINE(lv_taxamt_num);
END;
/
```

Decision Structures (continued)

- IF Statements
  - Simple IF
  - IF/THEN/ELSE
  - IF/THEN/ELSIF/ELSE
- CASE Statements
  - Basic CASE statement
  - Searched CASE statement
  - CASE expression

PL/SQL Tables

- PL/SQL TABLEs combine characteristics of SQL tables and C/Pascal arrays.
- Like SQL tables:
  - consist of records (must have a numeric primary key)
  - can grow/shrink as elements are added/removed
  - No limit

PL/SQL Tables…

- New table types can be defined via:

  TYPE *TypeName* IS TABLE OF *BaseType*
  INDEX BY BINARY_INTEGER;

- **Example:** a type for tables of employees
  - `TYPE EmpTab IS TABLE OF Employees%ROWTYPE`
    `INDEX BY BINARY_INTEGER;`
  - `first_table EmpTab;`
  - `another_table EmpTab;`

PL/SQL Tables….

- Elements of tables are accessed via *Table*(*Expr*) notation. The expression must be convertable to type BINARY_INTEGER (e.g. INT).
- **Example:** setting up a table from a relation

```
DECLARE -- assume type declaration from above
    TYPE EmpTab IS TABLE OF Employees%ROWTYPE
                    INDEX BY BINARY_INTEGER;
    rich_emps  EmpTab;
    n           INTEGER;
BEGIN
    FOR emp IN (SELECT * FROM Employees) LOOP
       n := n + 1;
       rich_emps(n) := emp;
    END LOOP;
END;
```

PL/SQL Tables….

- A number of built-in operators are defined on PL/SQL tables:
  - COUNT: Number of elements currently in table
  - DELETE: Deletes one or more elements from table
  - FIRST: Returns smallest index into table
  - LAST: Returns largest index into table
  - NEXT: Returns next defined index Into table
  - PRIOR: Returns previous defined index into table
  - EXISTS: Tests whether index value is valid

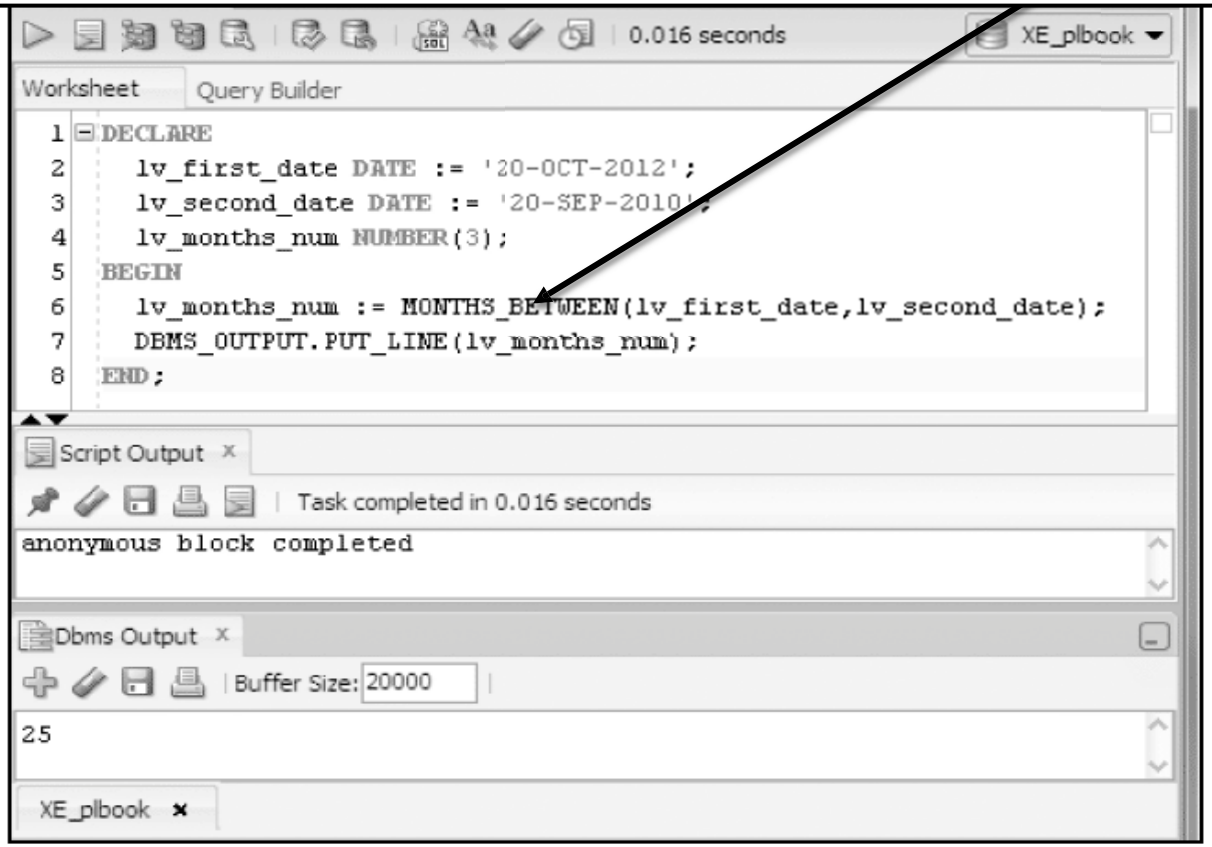
Storing PL/SQL Table data in Oracle Tables

Example: Dumping a PL/SQL table into an Oracle TABLE

```
DECLARE -- assume type declaration from above
         emps EmpTab;
         i INTEGER;
 BEGIN
         i = emps.FIRST;
         WHILE i <= emps.LAST LOOP
            -- Unfortunately, can't do this
            -- INSERT INTO Employees VALUES emps(i);
            -- so we do this ...
            INSERT INTO Employees VALUES
             (emps(i).id#, emps(i).name, emps(i).salary);
            i = emps.NEXT;
         END LOOP;
END;
```

Using SQL Functions

- SQL functions such as MONTHS_BETWEEN can be used within PL/SQL statements



Commenting Your Code

- Prefix single-line comments with two hyphens (--).
- Place a block comment between the symbols /* and */.

```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

Retrieving Data in PL/SQL: Example

- Retrieve `hire_date` and `salary` for the specified employee.

```
DECLARE
 v_emp_hiredate    employees.hire_date%TYPE;
 v_emp_salary      employees.salary%TYPE;
BEGIN
  SELECT    hire_date, salary
  INTO      v_emp_hiredate, v_emp_salary
  FROM      employees
  WHERE     employee_id = 100;
  DBMS_OUTPUT.PUT_LINE ('Hire date is :'|| v_emp_hiredate);
  DBMS_OUTPUT.PUT_LINE ('Salary is :'|| v_emp_ salary);
END;
/
```

Naming Ambiguities

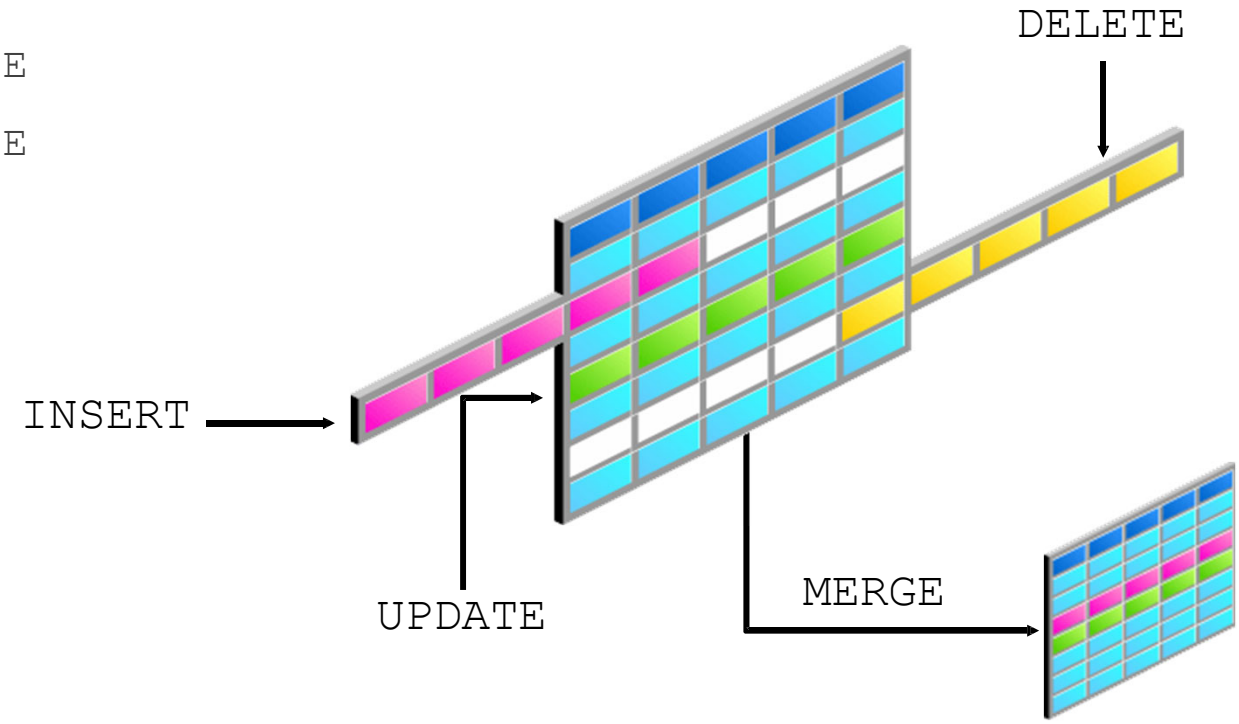
```
DECLARE
   hire_date        employees.hire_date%TYPE;
   sysdate          hire_date%TYPE;
   employee_id      employees.employee_id%TYPE := 176;
BEGIN
   SELECT       hire_date, sysdate
   INTO         hire_date, sysdate
   FROM         employees
   WHERE        employee_id = employee_id;
END;
/
```

```
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
01422. 00000 -  "exact fetch returns more than requested number of rows"
*Cause:    The number specified in exact fetch is less than the rows returned.
*Action:   Rewrite the query or change number of rows requested
```

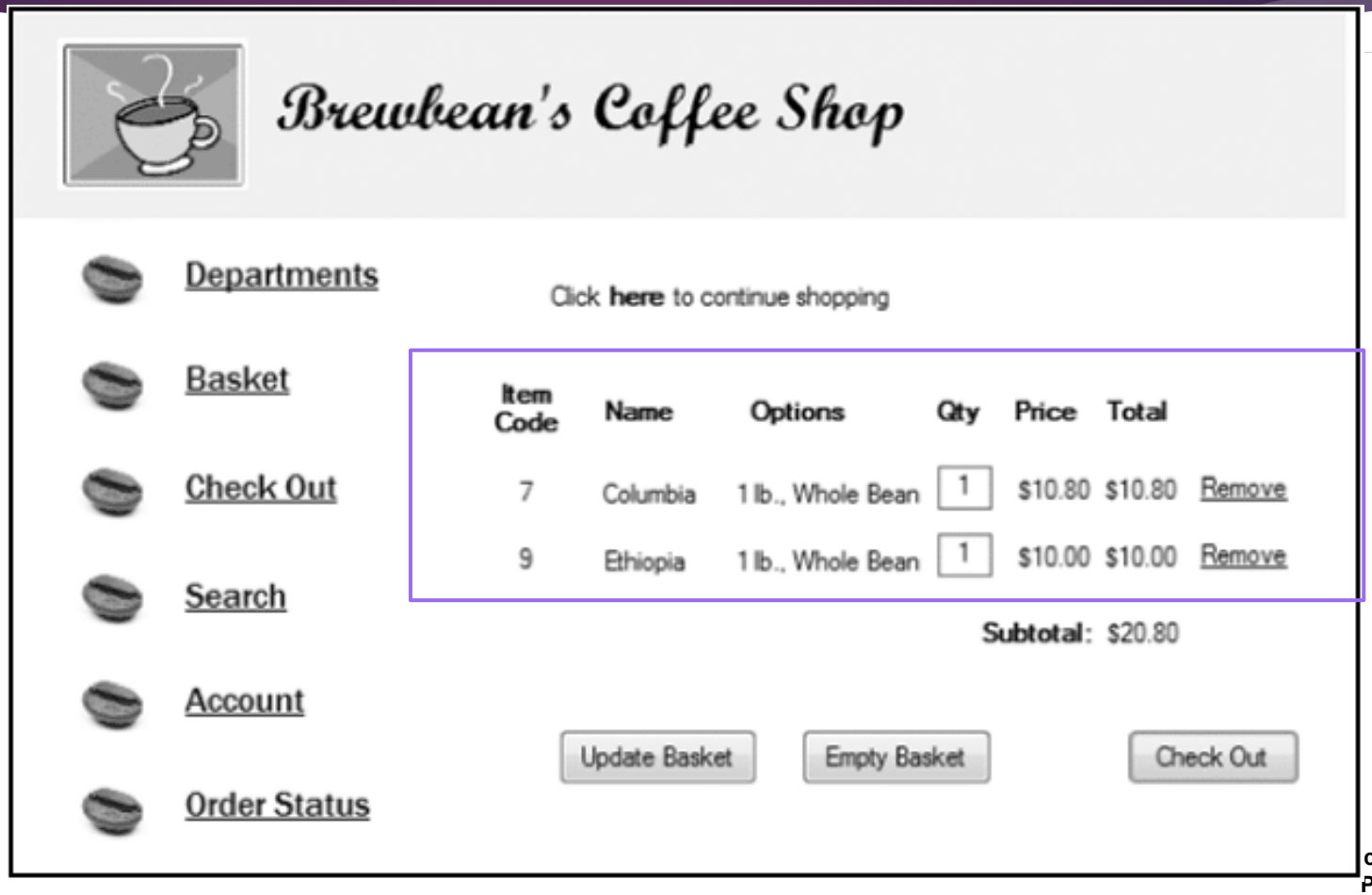
Cursors: Retrieving more than one row

Using PL/SQL to Manipulate Data

- Make changes to database tables by using DML commands:
  - INSERT
  - UPDATE
  - DELETE
  - MERGE



Processing multiple data rows



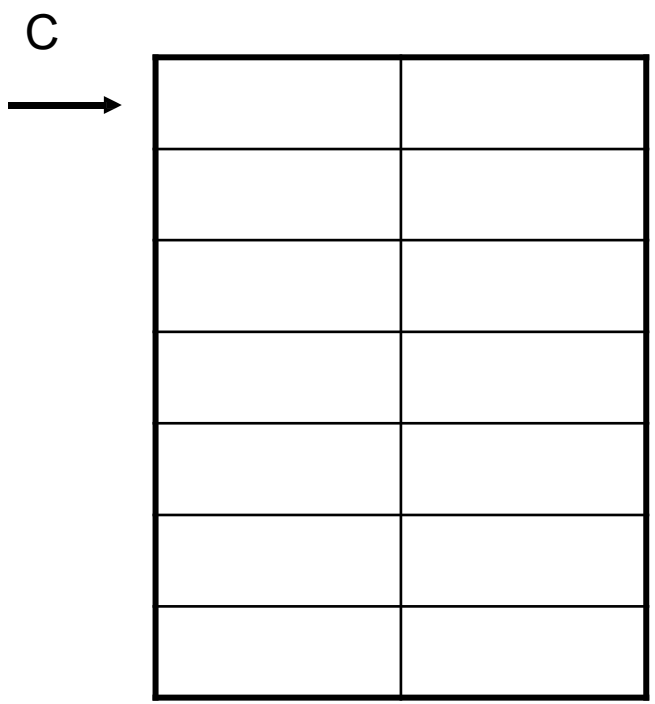
Retrieving More than One Row Using a Cursor

Impedance mismatch:

- SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records. No such data structure exist traditionally in procedural programming languages such as C++.
  - PL/SQL supports a mechanism called a ***cursor*** to handle this.

Cursors

- A *cursor* is a variable/pointer that can be used to access the result of a particular SQL query.
- Cursors can move sequentially from row to row.
- Two types
  - **Implicit:** Created and managed internally by the Oracle Server to process SQL statements
  - **Explicit:** Declared explicitly by the programmer



Cursors

- Every SQL query statement in PL/SQL has an implicit cursor. It is also possible to declare and manipulate cursors explicitly:

```
DECLARE
   CURSOR e IS
        SELECT * FROM Employees
            WHERE salary > 30000.00;
BEGIN
     ...
END;
```

- Cursors provide flexibility in processing rows of a query.

Cursors (contd.)

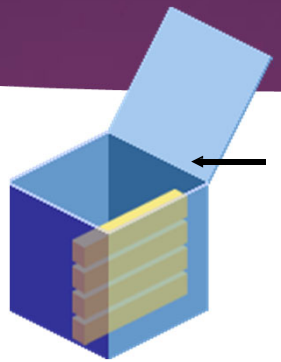
- Simplest way to deal with a cursor is to loop over all rows using a FOR loop:

```
DECLARE
    CURSOR e IS
    SELECT * FROM Employees WHERE salary > 30000.00;
     total INTEGER := 0;
 BEGIN
    FOR emp IN e LOOP
      total := total + emp.salary;
 END LOOP;
  dbms_output.put_line( 'Total Salaries: ' || total);
END;
```

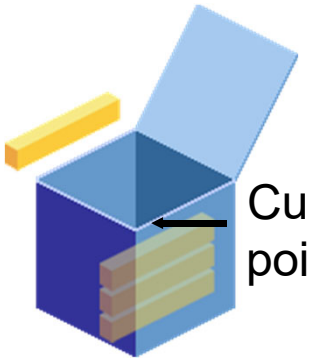
- Cursor loop variables are implicitly declared as the ROWTYPE for the SELECT result.
  - E.g. emp is **implictly** declared as Employees%ROWTYPE.

Controlling Explicit Cursors

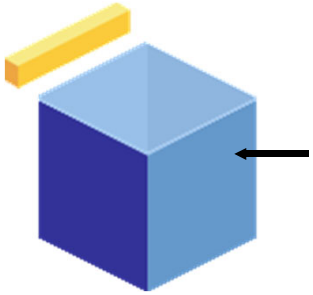
1 Open the cursor.



2 Fetch a row.



3 Close the cursor.



FETCH instead of FOR

- -- assume declarations as before

```
 OPEN e FOR SELECT * FROM EMPLOYEES;
 LOOP
    FETCH e INTO emp;
    EXIT WHEN e%NOTFOUND;
    total := total + emp.salary;
END LOOP;
CLOSE e; ...
```

- The FETCH operation can also extract components of a row:
  - **FETCH e INTO my_id, my_name, my_salary;**

Cursors Attributes

- Cursors have several built-in attributes:
  - %FOUND ... true whenever a row is successfuly fetched
  - %ISOPEN ... true if cursor is currently active
  - %NOTFOUND ... true after last row has been read
  - %ROWCOUNT ... returns number of rows in cursor-relation
- Yet another method for cursor iteration:
  - -- assume declarations as before

```
OPEN e;
FOR i IN 1..e%ROWCOUNT LOOP
   FETCH e INTO emp; -- process emp in some way
END LOOP;
```

Exceptions: Handling Errors

What Is an Exception?

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John''s last name is :' ||v_lname);
END;
```

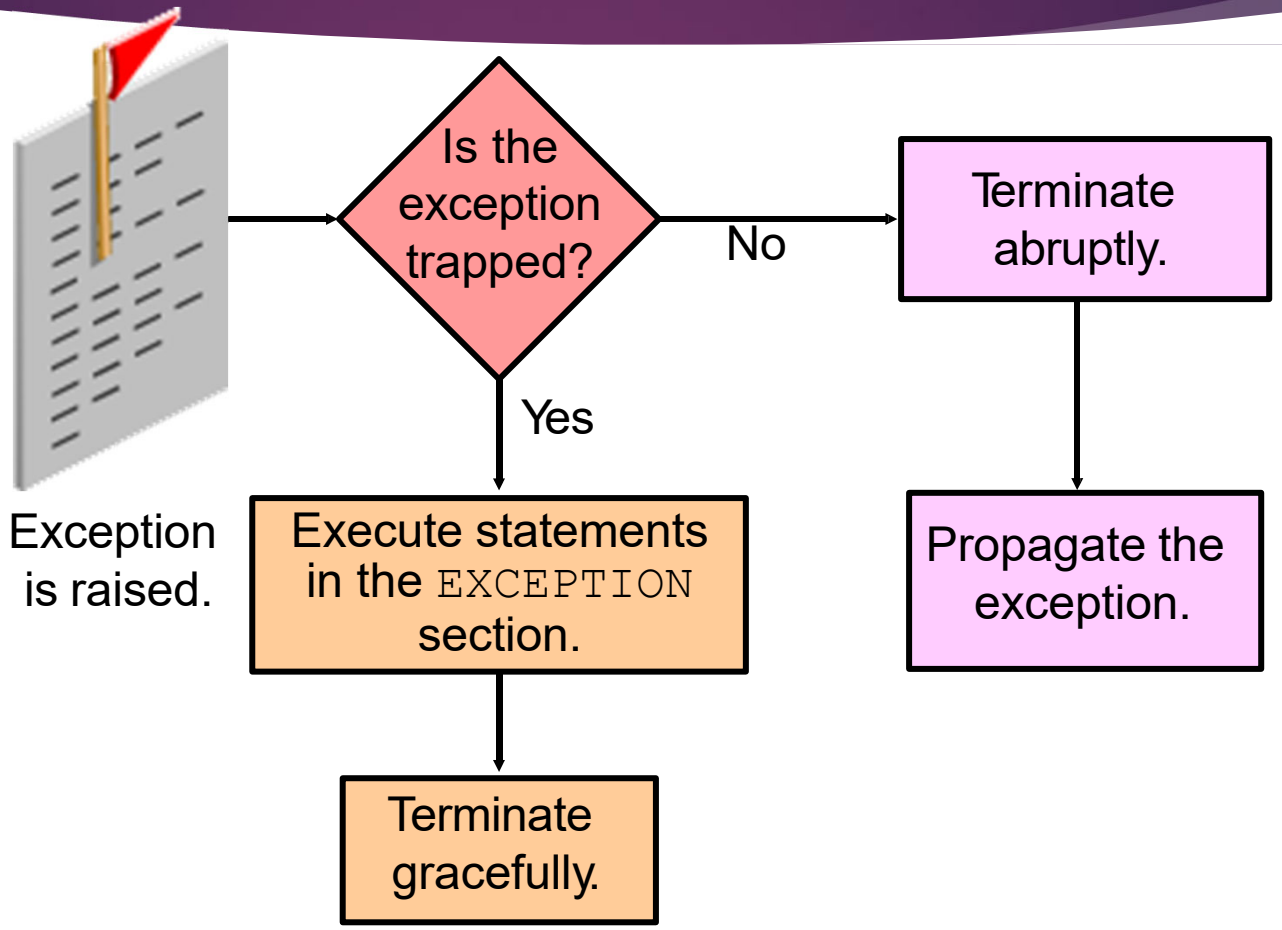
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
Error starting at line 3 in command:
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname FROM employees WHERE
  first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John''s last name is : ' ||v_lname);
END;
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
01422. 00000 -  "exact fetch returns more than requested number of rows"
*Cause:    The number specified in exact fetch is less than the rows returned.
*Action:   Rewrite the query or change number of rows requested
```

Exceptions

- An *exception* is an unusual/erroneous condition encountered during execution:
  - system error (e.g. "out of memory")
  - error caused by user program
  - warning issued by application
- PL/SQL's exception handling allows these to be handled "cleanly" in a central place.

Handling Exceptions



Exception Handler Syntax

```
BEGIN
    ... Statements ...
    EXCEPTION
        WHEN ExcepName1 THEN Statements1;
        WHEN ExcepName2 THEN Statements2;
            ...
  END;
```

- If an error occurs in *Statements*, control is transferred to:
  - the exception handler in this block
  - the exception handler at the next enclosing block
  - ... and so on out to the system level

Some Predefined Exceptions

PL/SQL provides exceptions for low-level/system errors:

- NO_DATA_FOUND SELECT..INTO returns no results
- INVALID_CURSOR Attempt to use non-open cursor
- INVALID_NUMBER Non-numeric-looking string used in context where number needed
- NOT_LOGGED_ON Attempted SQL operation without being connected to Oracle
- STORAGE_ERROR PL/SQL store runs out or is corrupted
- VALUE_ERROR Arithmetic conversion, truncation, size-constraint error

User-defined

- Exceptions are defined by NAME; used by RAISE.
- **Example:**

```
DECLARE
   outOfStock EXCEPTION;
   qtyOnHand INTEGER;
BEGIN
     ...
    IF qtyOnHand < 1 THEN
       RAISE outOfStock;
    END IF;
    ...
    EXCEPTION WHEN outOfStock THEN
           -- handle the problem
END;
```

- User-defined exceptions are local to a block and its sub-blocks.

PL/SQL Transactions

- A *transaction* is an 'atomic' sequence of SQL/plsql statements to accomplish a single task.
- The first SQL statement begins a transaction.
- **COMMIT** forces any changes made to be written to database.
- **ROLLBACK** restores database to state at start of transaction.
- Finer grain control:
  - Can create **SAVEPOINTs** within a transaction
  - Can rollback to a specific savepoint, etc.

Example

```
BEGIN
     ...
     UPDATE Employees SET ...
         WHERE id# = emp_id;
    DELETE FROM Employees WHERE ... ...
    SAVEPOINT more_changes;
    ... -- make changes to Employees
    -- possibly raise some_exception ...
    COMMIT;
    EXCEPTION
        WHEN some_exception
              THEN ROLLBACK TO more_changes;
END;
```

“

Next Lecture..view

”

PL/SQL Block Types

Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

Anonymous

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

Arguments to Procedures/Functions

- Each argument has a *mode*:
  - IN parameter is used for input only (default)
  - OUT paramter is used to return a result
  - IN OUT returns result, but initial value is used
- Can also specify a DEFAULT value for each argument.

Example

```
PROCEDURE raise(emp# INTEGER, increase REAL) IS
      current_salary REAL;
      salary_missing EXCEPTION;
BEGIN
      SELECT salary INTO current_salary FROM Employees
            WHERE id# = emp#;
      IF current_salary IS NULL THEN
            RAISE salary_missing;
      ELSE
            UPDATE Employees SET salary = salary + increase
                 WHERE id# = emp#;
      END IF;
      EXCEPTION
             WHEN NO_DATA_FOUND THEN
               INSERT INTO Audit VALUES (emp#, "Unknown
              employee");
             WHEN salary_missing THEN
                INSERT INTO Audit VALUES (emp#, "Null salary");
END;
```

Triggers

- Oracle triggers are PL/SQL or Java procedures that are invoked when specified database activity occurs
- Triggers can be used to
  - Enforce a business rule
  - Set complex default values
  - Update a view
  - Perform a referential integrity action
  - Handle exceptions

Summary

- PL/SQL: Procedural Language extension for SQL
- Declarations for Variables,..
- Assignments
- PL/SQL-specific types: Records, Tables, Cursors
- Exception Handling
- Procedures and Functions