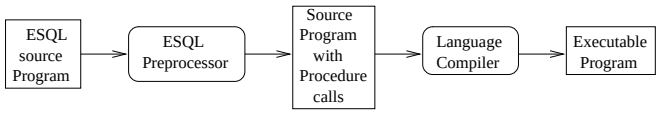
Relational Application Programming

- SQL is not computationally complete
- **Embedded SQL** (ESQL): SQL expressions embedded into programs in traditional algorithmic languages (e.g., C, COBOL): **host language**
- In addition, DBMSs provide proprietary languages that integrate SQL:
  - ◇ Informix-4GL
  - ◇ Informix Stored Procedure Language (SPL)
  - ◇ Oracle PL/SQL
  - ◇ Microsoft's Transact SQL (T-SQL)

Embedded SQL



- Application program = statements in host language interspersed with SQL requests
- Preprocessor replaces relational expressions by calls to compiled modules realizing relational access
- Result of relational requests exploited one tuple at a time in host programming language ⇒ "impedance mismatch"
- **Static embedding:** SQL statements written as part of the source program text
- **Dynamic embedding:** SQL statements composed by DBMS according to data input by users

Programming Oracle Applications

Done in several ways

- Writing interactive SQL queries in the SQL query mode
- Writing programs in a host language and embedding SQL within the program
  - ◇ precompiler (e.g., PRO*COBOL, PRO*C) used to link the application to Oracle
- Writing in PL/SQL, Oracle's procedural language
- Using Oracle Call Interface (OCI) and the Oracle runtime library SQLLIB

Programming in PL/SQL

- Oracle's procedural language extension to SQL with
  - ◇ data encapsulation, information hiding, overloading, exception handling
- Most heavily used technique for application development in Oracle
- Block-structured language
- Basic units: procedures, functions, anonymous blocks
- Can contain any number of nested subblocks
- Declarations are local to the block and cease to exist when the block completes
- Structure of PL/SQL block

```
[ DECLARE
   --- declarations ]
BEGIN
   --- statements
[ EXCEPTION
   --- handlers ]
END;
```

- Declaration part
  - ◇ optional
  - ◇ variables and objects are declared
  - ◇ variables can have any SQL data type as well as additional PL/SQL data types
  - ◇ variables can be assigned values
- Executable part
  - ◇ objects are manipulated
  - ◇ only required part
  - ◇ data processed using conditional, iterative, and sequential flow-of-control statements: IF-THEN-ELSE, FOR-LOOP, WHILE-LOOP, EXIT-WHEN, GO-TO
- Exception part
  - ◇ exceptions or errors raised during execution can be handled
  - ◇ user-defined and database exceptions or errors
  - ◇ When an error or exception occurs, an exception is raised and the normal execution stops, control transfers to the exception-handling part of the PL/SQL block or subprogram

Programming in PL/SQL: Example

```
DECLARE
   v_fname   employee.fname%TYPE;
   v_minit   employee.minit%TYPE;
   v_lname   employee.lname%TYPE;
   v_address employee.address%TYPE;
   v_salary  employee.salary%TYPE;
BEGIN
  SELECT fname, minit, lname, address, salary
  INTO   v_fname, v_minit, v_lname, v_address, v_salary
  FROM employee
  WHERE salary = (select max(salary) from employee );
  DBMS_OUTPUT.PUT_LINE(v_fname, v_minit, v_lname,
    v_address, v_salary);
EXCEPTION
  WHEN OTHERS
  DBMS_OUTPUT.PUT_LINE('Error Detected');
END;
```

5

- It is necessary to declare program variables to match the types of the database attributes that the program will process
- `%TYPE` means that the variable is of the same type as the corresponding column in the table
- `DBMS_OUTPUT.PUT_LINE`: PL/SQL's print function
- Error message printed if error is detected while executing the SQL: in this case if more than one employee is selected
- `INTO` clause specifies the program variables into which attribute values from the DB are retrieved

Programming in PL/SQL: Example

```
DECLARE
   avg_salary NUMBER;
BEGIN
  SELECT avg(salary) INTO avg_salary
  FROM employee;
  UPDATE employee
  SET salary = salary * 1.1
  WHERE salary < avg_salary;
  SELECT avg(salary) INTO avg_salary
  FROM employee;
  IF avg_salary > 50000 THEN
    DBMS_OUTPUT.PUT_LINE('Average Salary is ' || avg_salary);
  END IF;
  COMMIT;
EXCEPTION
  WHEN OTHERS
  DBMS_OUTPUT.PUT_LINE('Error in Salary update');
  ROLLBACK;
END;
```

6

- `avg_salary` is defined as a variable and gets the value of the average of the employee's salary from the first SELECT statement
- this value is used to choose which of the employees will have their salaries updated
- `EXCEPTION` part rolls back the whole transaction (removes any effect of the transaction on the DB) if an error of any type occurs during execution

Cursors in PL/SQL

- Multirow queries handled in two stages
  - ◇ query is started ("opened")
  - ◇ rows are requested one at a time
- These operations performed with a **cursor** = data structure to hold current state of query execution
- Similar to a file variable or file pointer
  - ◇ points to a single tuple from the result of a query
- Sequence of operations needed in application program
  - ◇ **declare** the cursor and its associated SELECT statement
  - ◇ **open** the cursor, starting execution of associated SELECT statement
  - ◇ iteratively **fetch** and process rows of data one at a time into host variables
  - ◇ **close** the cursor after last row is fetched

Cursors in PL/SQL, cont.

- Cursor attributes
  - ◇ `%ISOPEN` returns TRUE if the cursor is already open
  - ◇ `%FOUND` returns TRUE if the last FETCH returned a row, returns FALSE if the last FETCH failed to return a row
  - ◇ `%NOTFOUND` is the logical opposite of `%FOUND`
  - ◇ `%ROWCOUNT` yields the number of rows fetched

Cursors in PL/SQL: Example (1)

```
DECLARE
   emp_salary NUMBER;      emp_super_salary NUMBER;
   emp_ssn CHAR(9);        emp_super_ssn CHAR(9);
   CURSOR salary_cursor IS SELECT ssn, salary, superssn FROM employee;
BEGIN
  OPEN salary_cursor
  LOOP
    FETCH salary_cursor INTO emp_ssn, emp_salary, emp_superssn;
    EXIT WHEN salary_cursor%NOTFOUND;
    IF emp_superssn IS NOT NULL THEN
      SELECT salary INTO emp_super_salary FROM employee WHERE ssn=emp_superssn;
      IF emp_salary > emp_super_salary THEN
        DBMS_OUTPUT.PUT_LINE(emp_ssn);
      END IF;
    END IF;
  END LOOP;
  IF salary_cursor%ISOPEN THEN CLOSE salary_cursor;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('Errors with ssn ' || emp_ssn);
  IF salary_cursor%ISOPEN THEN CLOSE salary_cursor;
END;
```


Cursors in PL/SQL: Example (2)

```
DECLARE
   v_fname   employee.fname%TYPE;
   v_minit   employee.minit%TYPE;
   v_lname   employee.lname%TYPE;
   v_address employee.address%TYPE;
   v_salary  employee.salary%TYPE;
   CURSOR EMP IS SELECT ssn, fname, minit, lname, salary FROM employee;
BEGIN
  OPEN EMP;
  LOOP
    FETCH EMP INTO v_ssn, v_fname, v_minit, v_lname, v_salary;
    EXIT WHEN EMP%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('SSN: ' || v_ssn || 'Old salary: ' || v_salary );
    UPDATE employee SET salary = salary * 1.1 WHERE ssn = v_ssn;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('SSN: ' || v_ssn || 'New salary: ' || v_salary*1.1 );
  END LOOP;
  CLOSE EMP;
EXCEPTION
  WHEN OTHERS
  DBMS_OUTPUT.PUT_LINE('Error detected');
END;
```

Programming in PRO*C

- Precompiler: programming tool that allows to embed SQL statements in a source program of some PL
  - ◇ accepts the source program as input
  - ◇ translates the embedded SQL statements into Oracle runtime library calls
  - ◇ generates a modified source program that an be compiled linked and executed
- PRO*C provides automatic conversion between Oracle and C data types
- SQL statements and PL/SQL blocks can be embedded in a C host program

11

Programming in PRO*C: Example (1)

```
#include <stdio.h>
#include <string.h>
VARCHAR username[30];      VARCHAR passwd[10];
VARCHAR v_fname;           VARCHAR v_minit;
VARCHAR v_lname;           VARCHAR v_address;
char v_ssn[9];             float f_salary;
main () {
strcpy(username.arr, "Scott");   username.len = strlen(username.arr);
strcpy(passwd.arr, "TIGER");     passwd.len = strlen(passwd.arr);
EXEC SQL WHENEVER SQLERROR DO sql_error();
EXEC SQL CONNECT :username IDENTIFIED BY :passwd;
EXEC SQL SELECT fname, minit, lname, address, salary
  INTO :v_fname, :v_minit, :v_lname, :v_address, :f_salary
  FROM employee
  WHERE salary=(select max(salary) from employee);
  printf( "%s %s %s %s %f \n" v_fname.arr,
    v_minit.arr, v_lname.arr, v_address.arr, f_salary);
}
sql_error() {
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf('Error Detected\n');
}
```

12

Programming in PRO*C: Example (2)

```
// Same include statements and variable declarations as in previous example
main () {
strcpy(username.arr, "Scott");  username.len = strlen(username.arr);
strcpy(passwd.arr, "TIGER");    passwd.len = strlen(passwd.arr);
EXEC SQL WHENEVER SQLERROR DO sql_error();
EXEC SQL CONNECT :username IDENTIFIED BY :passwd;
EXEC SQL DECLARE EMP CURSOR FOR
     SELECT ssn, fname, minit, lname, salary FROM employee;
EXEC SQL OPEN EMP;
EXEC SQL WHENEVER NOTFOUND DO BREAK;
for (;;) {
  EXEC SQL FETCH EMP INTO :v_ssn, :v_fname, :v_minit, :v_lname, :f_salary;
  printf('SSN: %d, Old Salary %f', v_ssn, f_salary );
  EXEC SQL UPDATE employee SET salary = salary * 1.1 WHERE ssn = v_ssn;
  EXEC SQL COMMIT;
  printf('SSN: %d, New Salary %f', v_ssn, f_salary*1.1 );
  }
EXEC SQL CLOSE EMP;
}
sql_error() {
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf('Error Detected\n');
}
```

13