Our View of Databases

PL/SQL

- **Procedural language** extension to SQL.
- Extends the data manipulation power of SQL with the power of procedural programming languages.
- For application development.
A PL/SQL program example 1: Introducing the block structure

DECLARE
  current_salary emp.salary%TYPE;
  salary_missing EXCEPTION;
BEGIN
  SELECT salary INTO current_salary FROM emp WHERE emp_no = 999;
  IF current_salary IS NULL THEN RAISE salary_missing;
  ELSE UPDATE emp SET salary = salary + 10000 WHERE emp_no = 999;
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN INSERT INTO emp_audit VALUES ('No id');
  WHEN salary_missing THEN INSERT INTO emp_audit VALUES ('No salary');
END;
/

PL/SQL program example 2
DECLARE
  qty_on_hand NUMBER(5);
BEGIN
  SELECT quantity INTO qty_on_hand FROM inventory
  WHERE product = 'TENNIS RACKET'
  FOR UPDATE OF quantity;
  IF qty_on_hand > 0 THEN  -- check quantity
    UPDATE inventory SET quantity = quantity - 1
    WHERE product = 'TENNIS RACKET';
    INSERT INTO purchase_record
    VALUES ('Tennis racket purchased', SYSDATE);
  ELSE
    INSERT INTO purchase_record
    VALUES ('Out of tennis rackets', SYSDATE);
  END IF;
  COMMIT;
END;

PL/SQL: Other constructs
Control structures, cursor variables, and records:
  - IF-THEN-ELSE
  - FOR-LOOP, WHILE-LOOP, LOOP-EXIT-WHEN, GOTO
    Example, FOR-LOOP and specifying range:
    SELECT ... INTO upper_limit FROM ...
    FOR i IN 1..upper_limit LOOP ...
    END LOOP;
  - CURSOR c IS SELECT ... FROM ...
    FOR i IN c LOOP ...
    END LOOP;
  - FOR i IN (SELECT ...) LOOP ...
  - table.attribute%TYPE, table%ROW
Iterative control, examples

• DECLARE
  salary_total INTEGER := 0;
  CURSOR c IS SELECT * FROM emp; -- name the sql work area for later access
BEGIN
  FOR emp_rec IN c LOOP
    salary_total := salary_total + emp_rec.salary;
  END LOOP;
END;
/

• DECLARE
  salary_total INTEGER := 0;
BEGIN
  FOR emp_rec IN (SELECT * FROM emp) LOOP
    salary_total := salary_total + emp_rec.salary;
  END LOOP;
END;
/

See more examples in the PL/SQL manual.

Iterative control, cont'd

DECLARE
  v_ename varchar2(10);
  v_empno emp.empno%type;
  CURSOR cur_emp IS
    SELECT ename, empno
    FROM emp
    WHERE deptno=10;
BEGIN
  OPEN cur_emp;
  LOOP
    FETCH cur_emp into v_ename, v_empno;
    EXIT when cur_emp%NOTFOUND;
    INSERT into temp (col1, message)
    values (v_empno, v_ename);
  END LOOP;
  CLOSE cur_emp;
END;

Referring to current row in a cursor

• my_cursor%NOTFOUND, my_cursor%FOUND
gives true/false if earlier FETCH returned one row.

• Refer to current row:
  DECLARE
    v_ename emp.ename%type;
    v_deptno emp.dept%type;
  CURSOR cur_emp IS
    SELECT ename, dept
    FROM emp;
BEGIN
  OPEN cur_emp;
  LOOP
    FETCH cur_emp into v_ename, v_deptno;
    EXIT when cur_emp%NOTFOUND;
    IF v_deptno=10
      DELETE FROM emp WHERE CURRENT OF my_cur;
    END LOOP;
  CLOSE cur_emp;
END;
Cursor with parameters

```sql
DECLARE
    CURSOR emp_cur (sal_value NUMBER) IS
    SELECT ename FROM emp WHERE sal > sal_value;
BEGIN
    OPEN emp_cur(1200);
    ......
END;
```

Same as:

```sql
DECLARE
    sal_value NUMBER;
    CURSOR emp_cur IS
    SELECT ename FROM emp WHERE sal > sal_value;
BEGIN
    sal_value := 1200;
    OPEN emp_cur;
    ......
END;
```

Stored procedures

```sql
CREATE OR REPLACE PROCEDURE raise_salary (emp_id IN INTEGER,
    amount NUMBER) IS
    current_salary NUMBER;
    salary_missing EXCEPTION;
BEGIN
    SELECT salary INTO current_salary FROM emp WHERE emp_no = emp_id;
    IF current_salary IS NULL THEN RAISE salary_missing;
    ELSE UPDATE emp SET salary = salary + amount WHERE emp_no = emp_id;
END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN INSERT INTO emp_audit VALUES (emp_id, 'No id');
    WHEN salary_missing THEN INSERT INTO emp_audit VALUES (emp_id, 'No salary');
END raise_salary;
/ 
execute raise_salary(999,111);
```

Stored functions

```sql
CREATE OR REPLACE FUNCTION raise_salary (emp_id IN INTEGER, amount IN NUMBER)
    RETURN INTEGER IS
    current_salary NUMBER;
    salary_missing EXCEPTION;
BEGIN
    SELECT salary INTO current_salary FROM emp WHERE emp_no = emp_id;
    IF current_salary IS NULL THEN RAISE salary_missing;
    ELSE UPDATE emp SET salary = salary + amount WHERE emp_no = emp_id;
END IF;
RETURN current_salary + amount;
EXCEPTION
    WHEN NO_DATA_FOUND THEN INSERT INTO emp_audit VALUES (emp_id, 'No id');
    WHEN salary_missing THEN INSERT INTO emp_audit VALUES (emp_id, 'No salary');
END raise_salary;
/ 
execute raise_salary(999,111); 
IF raise_salary(999,111) > 15000 THEN ...
No data found

begin
  select year into year_exists ... where year = v_year;
  if year_exists is NULL then insert into ...
end our_procedure;

• The above check for NULL is never executed. Instead, use exception handling:

begin
  begin
    select year into year_exists ... where year = v_year;
    exception when no_data_found then insert into ...
  end;
end our_procedure;

"Warning: Procedure created with compilation errors": How to debug

set serveroutput on
begin
  v_line := 'Hello World';
  dbms_output.put_line (v_line);
end;
/

SHOW ERRORS PROCEDURE your_procedure_name_here;
or show the most recent error:
sho err

 Assertions

• CREATE TABLE TEACHER ( PNum CHAR(11), FName VARCHAR(20), LName VARCHAR(20), Office CHAR(10), Phone CHAR(4) NOT NULL CHECK (Phone>'0000' AND Phone<'9999'), CONSTRAINT pk_TEACHER PRIMARY KEY (PNum), CONSTRAINT fk_TEACHER FOREIGN KEY (Office) REFERENCES OFFICE(ID), CONSTRAINT chk_TEACHER CHECK (Phone<Office));

DBMS only checks when inserting/updating TEACHER.

• CREATE ASSERTION ass_names CHECK( NOT EXISTS ( SELECT * FROM TEACHER T, OFFICE O WHERE T.LName=O.Name));
Triggers

- Checks and assertions only offer one option as action: abort the operation.
- **Triggers** give greater flexibility by enabling the user to decide on the action to take.
- Trigger = event + condition + action.

When to consider the trigger, e.g., updating
When to execute the action; SQL condition
PL/SQL block or stored procedure

- **CREATE OR REPLACE TRIGGER TRIGGER_TOTAL_SALARY AFTER INSERT ON EMPLOYEE FOR EACH ROW WHEN (NEW.DNO IS NOT NULL) BEGIN UPDATE DEPARTMENT SET TOTAL_SALARY = TOTAL_SALARY + :NEW.SALARY WHERE DNO = :NEW.DNO; END;**

• Other constructs:
  - AFTER, BEFORE, INSTEAD OF
  - INSERT, DELETE, UPDATE (attribute, attribute, ...)
  - (FOR EACH ROW)
  - WHEN
  - NEW.attribute, OLD.attribute (Use NEW and OLD in WHEN clause and NEW and OLD in the body, mind the colons)
  - Call a stored procedure within the trigger, e.g., raise_salary(999,111); Any other scenario where TOTAL_SALARY needs updating? Create triggers to guarantee its consistency in those scenarios!
Triggers

- When is the condition evaluated?
  - Immediate.
  - Before, after, or instead of the triggering event.
  - Deferred.
  - Detached.

- When is the action executed?
  - Immediate, deferred, or detached.

Main uses of triggers: Constraint checking, notification, maintenance of derived attributes, and duplicated tables.

Commit and rollback

- Transaction: SQL statements that are treated as an atomic unit
  - A transaction begins with the first SQL statement issued
  - A transaction ends with commit (COMMIT; or DDL statement or user disconnection) or rollback (ROLLBACK; or unhandled exception or abnormal termination)

- Commit: Make permanent the changes made by the transaction

- Rollback: Undo the changes made by the transaction
  - up to previous commit or savepoint?

- Savepoint: Intermediate marker.

```sql
UPDATE emp
SET sal = 2000 WHERE ename = 'BLAKE';
SAVEPOINT blake_sal;
UPDATE emp
SET sal = 1500 WHERE ename = 'CLARK';
SAVEPOINT clark_sal;
SELECT SUM(sal) FROM emp;
ROLLBACK TO SAVEPOINT blake_sal;
UPDATE emp
SET sal = 1200 WHERE ename = 'CLARK';
COMMIT;
```