Our View of Databases

Real World

Model

Database management system

Physical database

Processing of queries and updates

Access to stored data

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 ... END 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.

Stored procedures

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

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 ...
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:
    sh 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));

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

DBMS only checks when inserting/updating TEACHER.
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.

Triggers

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

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

Triggers

• Other constructs:
  - AFTER, BEFORE, INSTEAD OF
  - INSERT, DELETE, UPDATE [attribute, attribute, ...]
  - [FOR EACH ROW]
  - [WHEN]
  - NEW attribute, OLD attribute [less NEW and OLD in WHEN clause and
    when used with SQL:2019, avoid the colon]
  - 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.