

# Why Not To Use Stored Procedures?

Portability

- PL are generally more difficult to develop and maintain than conventional programming languages
  - n Less language features
  - n Less tool support

# Procedures and Functions in Oracle

#### Procedure

- n No return value
- <sup>n</sup> Usually called by other procedures, functions, triggers, and/or programs.
- Function
  - n Returns a value
  - <sup>n</sup> Usually used in SQL statements like the system built-in functions

# Example: hello()

create or replace procedure hello as begin dbms\_output.put\_line( 'Hello World!' ); end hello;

ena /

Note that *hello* does not have a parameter list, not even ()

## **Create Procedures**

CREATE [OR REPLACE] PROCEDURE *procedure\_name* [(*parameter\_name* [IN | OUT | IN OUT] *type* [, ...])] {IS | AS} BEGIN *procedure\_body* 

END procedure\_name;

### **Use Procedures**

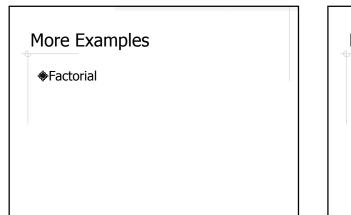
@call hello();

- show errors
- wuser\_procedures
  - $_{\rm n}$  describe user\_procedures
  - n select object\_name from user\_procedures;
- In the second second

## Parameter Mode

- IN: the parameter already has a value when the procedure starts, and the value cannot be changed in the procedure body; *default mode*.
- OUT: the parameter value will set in the procedure body.
- ♦ IN OUT: the parameter has a value when the procedure start, and the value may be changed in the procedure body.

#### Example: sum2p() Example: sum2f() create or replace function sum2f (a in integer, b in integer) return integer as create or replace procedure sum2p s integer default 0; (a in integer, b in integer, s out integer) as begin begin sum2p( a, b, s ); s := a+b; return s; end sum2p; end sum2f; Note that the declaration block is between CREATE...AS and BEGIN, and the DECLARE keyword is not needed any more.



### Packages

A package is a collection of PL/SQL objects group together under one package name.

- <sup>n</sup> Procedures and functions
- <sup>n</sup> Cursors, variables, and types
- Package
  - <sup>n</sup> Specification declarations
  - ... Body implementations

### **Create Packages**

CREATE [OR REPLACE] PACKAGE *package\_name* {IS | AS} *package\_specification* END *package\_name*;

CREATE [OR REPLACE] PACKAGE BODY *package\_name* {IS | AS} *package\_body* END *package\_name*;

# Package Specification Example

create or replace package cs422stu31 as procedure hello; procedure sum2p (a in integer, b in integer, s out integer); function sum2f (a in integer, b in integer) return integer; end cs422stu31;

### **Use Packages**

- @call cs422stu31.hello();
- select cs422stu31.sum2f(100,5) from daul;
- select object\_name, procedure\_name from user\_procedures;
- drop package cs422stu31;

## Triggers

- Procedures that are automatically invoked when data is *changed*, e.g. INSERT, DELETE, and UPDATE.
- Common use of triggers
  - n Auditing
  - n Constraints
  - n Replication

# Example: Change Logger

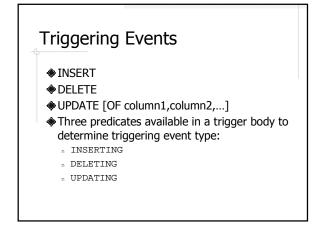
create or replace trigger *change\_logger* before insert or update or delete on *items* begin if *inserting* then insert into log1 (operation) values ('insert'); elsif *deleting* then

- insert into log1 (operation) values ('delete'); else
- insert into log1 (operation) values ('update'); end if;

#### end;

# Create Trigger

CREATE [OR REPLACE] TRIGGER *trigger\_name* {BEFORE | AFTER | INSTEAD OF} *triggering\_event* ON *table\_name* [FOR EACH ROW [WHEN *trigger\_condition*]] BEGIN *trigger\_body* END *trigger\_name*,



## Before or After

- BEFORE: trigger fires before the triggering event
- ♦AFTER: trigger fires after the event
- INSTEAD OF: execute the trigger procedure *instead of* the triggering event (statement)

# Statement Trigger vs. Row Trigger

- Statement Trigger
   Fires once per statement
- Row Trigger
  - n FOR EACH ROW
  - $_{\rm n}\,$  Fires once per row

# Example: Price Logger

- Log the price changes where the new price is more than 20% higher or lower than the old price.
- ♦:OLD and :NEW

## **Use Triggers**

Information about triggers is in the user\_triggers table.
drop trigger *trigger\_name*

# Oracle Restrictions on Triggers

Avoid infinite triggering

- Assume the triggering event is on R
  - n R *cannot* be changed in the trigger body
  - Any relation linked to R by a chain of foreign key constraints *cannot* be changed in the trigger body

# More Restriction on Row Triggers

- A row trigger cannot even query a mutating table, which is
  - <sup>n</sup> either the table being modified, or
  - the table could be modified due to a CASCADE foreign key policy
- Get around the "mutating table error" is fairly tricky
  - (http://asktom.oracle.com/~tkyte/Mutate/)
- However, most of the time you can use a statement trigger instead.

## **Constraints Revisited**

- ♦NOT NULL
- DEFAULT
- ♦UNIQUE
- PRIMARY KEY
- Foreign key
- Check

# Foreign Key Constraint

- Parent and child tables
- What happens if a tuple in the parent table is deleted?
  - n Default: no allowed
  - n ON DELETE CASCADE
  - n ON DELETE SET NULL
- ♦How about ON UPDATE??

## Limitations of the Check Constraint

- The condition must be a boolean expression that can be evaluated using the row being inserted or updated
- The condition cannot contain subqueries
- The condition cannot contain certain SQL functions or peudocolumns
- The condition cannot contain user-defined functions

# Implement Constraints using Triggers

Students( sid, sname ) Assignments( aid, aname, due ) Turnins( sid, aid, filename )

- A new tuple cannot be inserted into *Turnins* if current time is past the due date.
- NOTE: use raise\_application\_error (*error\_code*, *error\_msg*) to raise an error
  - *error\_code* is between -20,000 and -20,999
  - *error\_msg* is up to 2048 characters long