Why Use Stored Procedures?
- **Performance**
  - compiled and optimized code
  - Save communication overhead
- **Security**
  - Access control
  - Less data transferred over the wire
- **Simplify application code**
- **Triggers for data integrity**

Why Not To Use Stored Procedures?
- **Portability**
- **PL** are generally more difficult to develop and maintain than conventional programming languages
  - Less language features
  - Less tool support

Procedures and Functions in Oracle
- **Procedure**
  - No return value
  - Usually called by other procedures, functions, triggers, and/or programs.
- **Function**
  - Returns a value
  - Usually used in SQL statements like the system built-in functions

Example: hello()
```sql
create or replace procedure hello as
begin
    dbms_output.put_line( 'Hello World!' );
    end hello;
/
```
Note that `hello` does not have a parameter list, not even `()`

Create Procedures
```sql
CREATE [OR REPLACE] PROCEDURE procedure_name
    ((parameter_name [IN | OUT | IN OUT] type[, ...]))
(IN | AS)
BEGIN
    procedure_body
END procedure_name;
```
Use Procedures

- call hello();
- show errors
- user_procedures
  - describe user_procedures
  - select object_name from user_procedures;
- drop procedure hello;

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()

```sql
create or replace procedure sum2p
  (a in integer, b in integer, s out integer) as
begin
  s := a+b;
end sum2p;
```

Example: sum2f()

```sql
create or replace function sum2f (a in integer, b in integer) return integer as
begin
  sum2p( a, b, s);
  return s;
end sum2f;
```

More Examples

- **Factorial**

Packages

- A package is a collection of PL/SQL objects group together under one package name.
  - Procedures and functions
  - Cursors, variables, and types
- **Package**
  - **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
- Auditing
- Constraints
- 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 | view_name}
{FOR EACH ROW [WHEN trigger_condition]}]
BEGIN
  trigger_body
END trigger_name;
Triggering Events

- INSERT
- DELETE
- UPDATE [OF column1,column2,...]

Three predicates available in a trigger body to determine triggering event type:
- INSERTING
- DELETING
- UPDATING

Before or After

- BEFORE: trigger fires before the triggering event
- AFTER: trigger fires after the event

Instead-Of Triggers

- Execute the trigger procedure instead of the triggering event (statement)
- Can only be used on views; and views can only use INSTEAD OF triggers.

Statement Trigger vs. Row Trigger

- Statement Trigger
  - Fires once per statement
- Row Trigger
  - FOR EACH ROW
  - 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
  - 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
  - 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?
  - Default: no allowed
    - ON DELETE CASCADE
    - 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 pseudocolumns
- 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