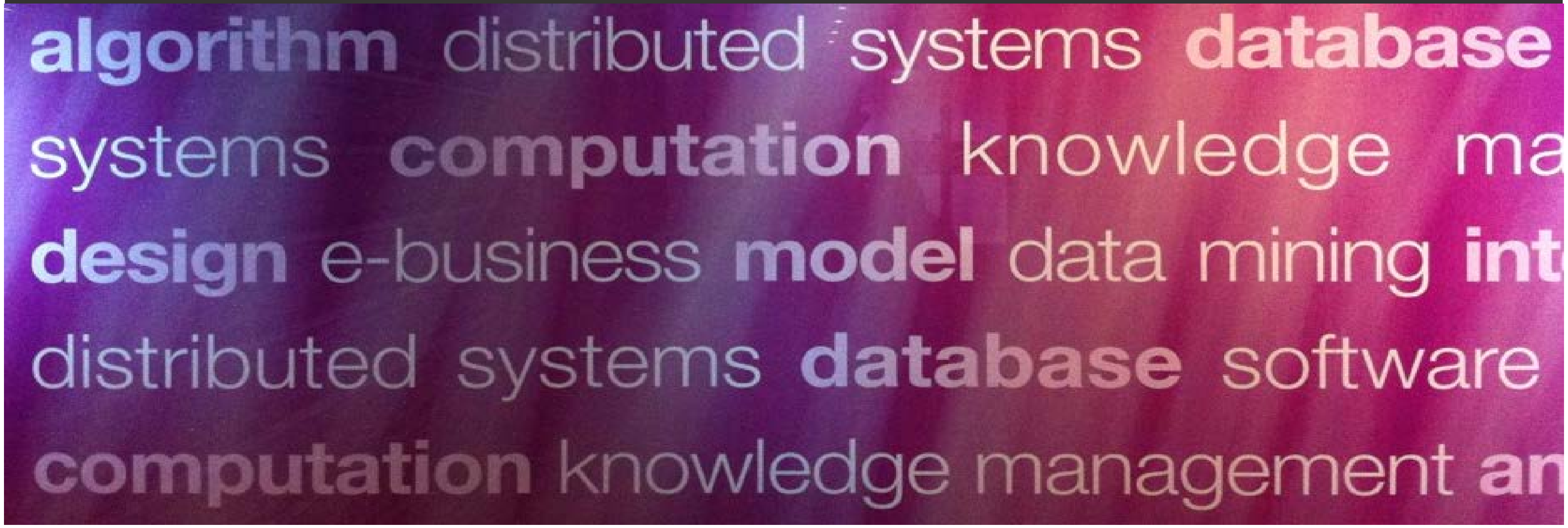
MONASH University

Information Technology

FIT3176 Advanced Database Design

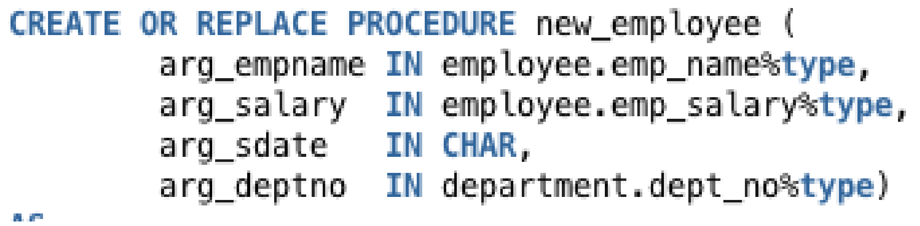
Dr Minh Le
Minh.Le@monash.edu

Topic 6: PL/SQL Packages


algorithm distributed systems database
systems computation knowledge ma
design e-business model data mining int
distributed systems database software
computation knowledge management an

*Adapted from slides developed by Lindsay Smith

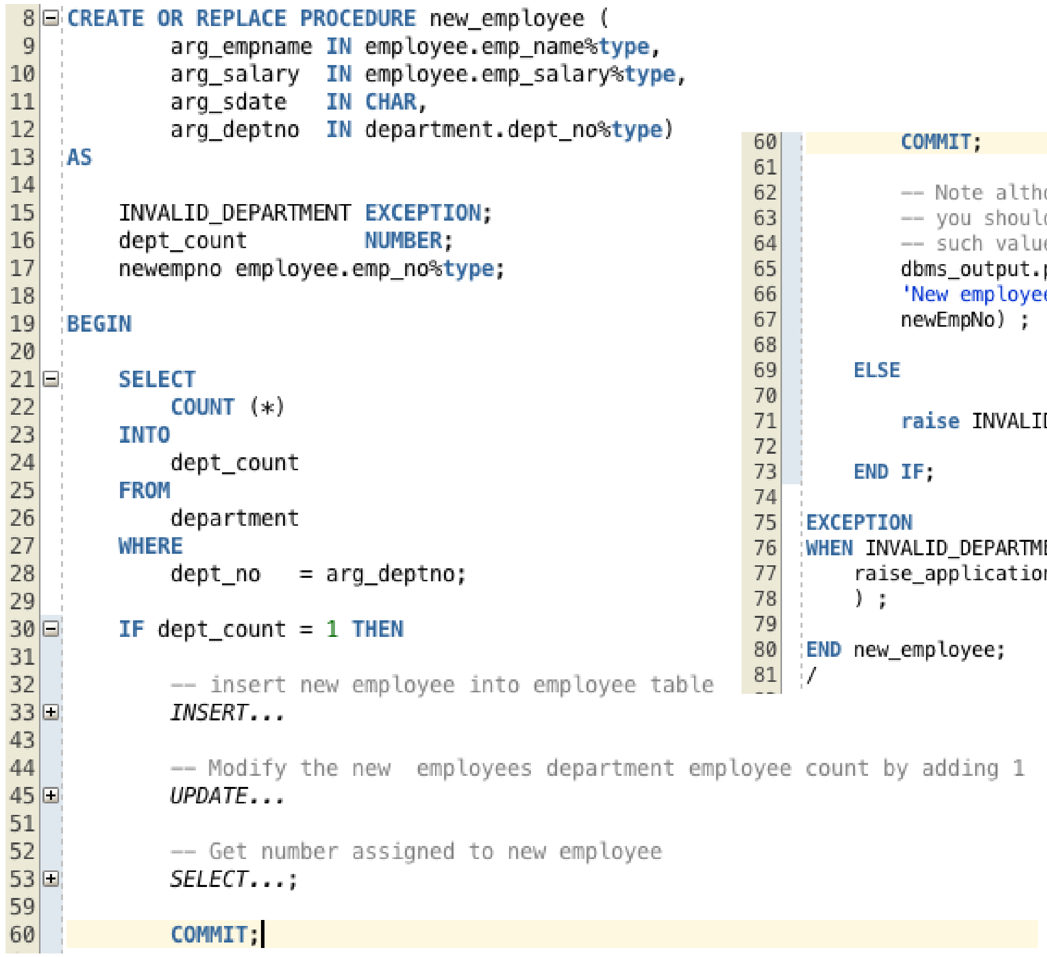
Week 5 New Employee procedure



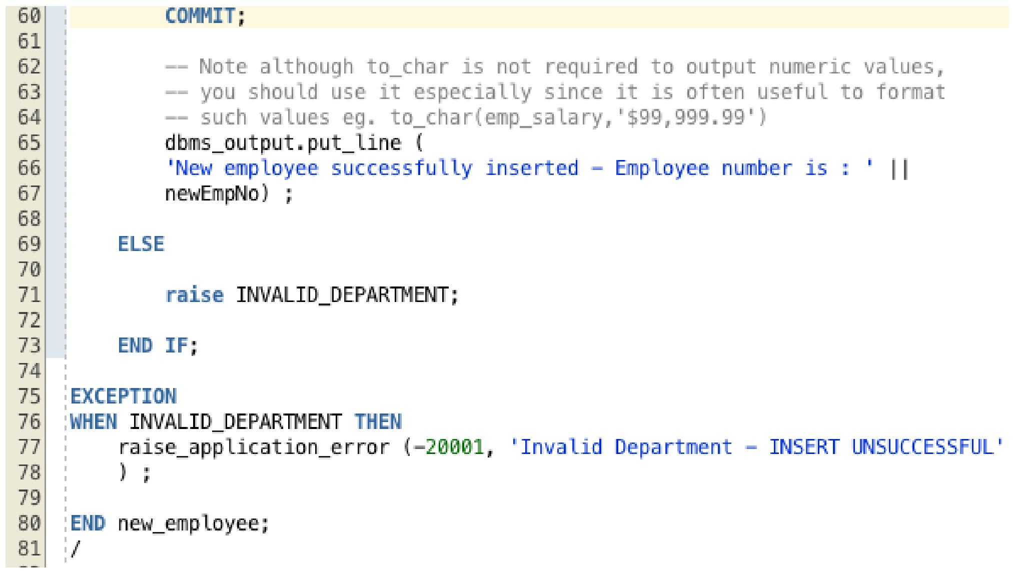
```
CREATE OR REPLACE PROCEDURE new_employee (
        arg_empname IN employee.emp_name%type,
        arg_salary  IN employee.emp_salary%type,
        arg_sdate   IN CHAR,
        arg_deptno  IN department.dept_no%type)
```

- Handed in department number may be incorrect
  - Declare
    - EXCEPTION INVALID_DEPARTMENT
  - Check if department exists
    - how? ***– carry out a select count(*) on deptno = arg_deptno***
    - if not
      - raise INVALID_DEPARTMENT
      - calls raise_application_error
        - » Number - ***negative integer in the range -20000 .. -20999***
        - » Message

Week 5 New Employee procedure



```
CREATE OR REPLACE PROCEDURE new_employee (
        arg_empname IN employee.emp_name%type,
        arg_salary  IN employee.emp_salary%type,
        arg_sdate   IN CHAR,
        arg_deptno  IN department.dept_no%type)
AS

    INVALID_DEPARTMENT EXCEPTION;
    dept_count         NUMBER;
    newempno employee.emp_no%type;

BEGIN

    SELECT
        COUNT (*)
    INTO
        dept_count
    FROM
        department
    WHERE
        dept_no    = arg_deptno;

    IF dept_count = 1 THEN

        -- insert new employee into employee table
        INSERT...

        -- Modify the new  employees department employee count by adding 1
        UPDATE...

        -- Get number assigned to new employee
        SELECT...;

        COMMIT;
```



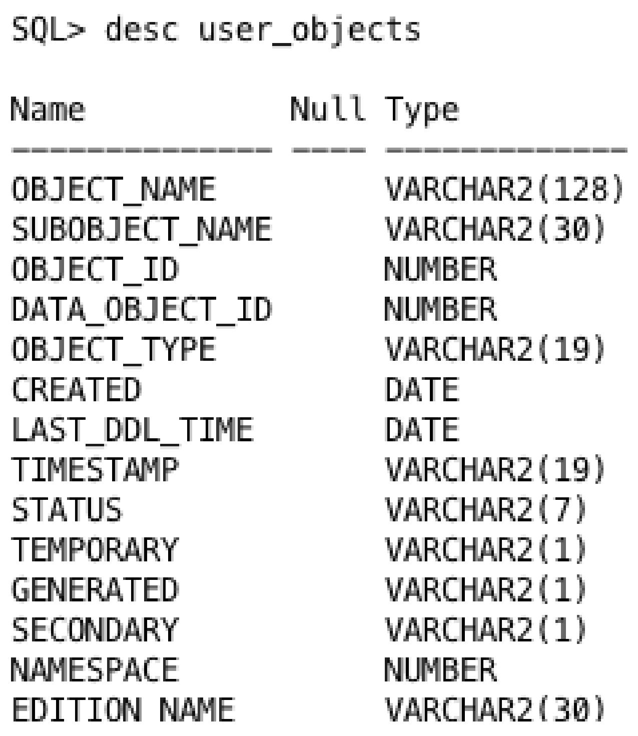
```
        COMMIT;

        -- Note although to_char is not required to output numeric values,
        -- you should use it especially since it is often useful to format
        -- such values eg. to_char(emp_salary,'$99,999.99')
        dbms_output.put_line (
        'New employee successfully inserted - Employee number is : ' ||
        newEmpNo) ;

    ELSE

        raise INVALID_DEPARTMENT;

    END IF;

EXCEPTION
WHEN INVALID_DEPARTMENT THEN
    raise_application_error (-20001, 'Invalid Department - INSERT UNSUCCESSFUL'
    ) ;

END new_employee;
/
```

See code on Moodle

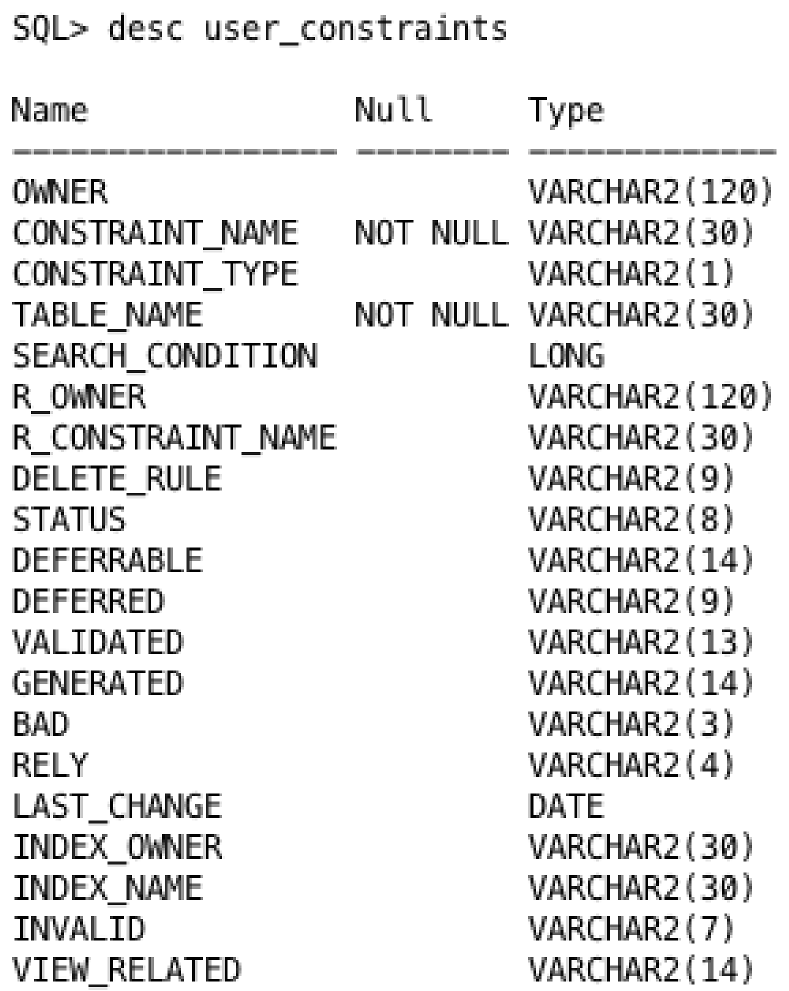
Our account has heaps of tables, procedures, …

How do we get rid of them?
What are the problems in removing objects?

Oracle system tables – object and constraints *you* own



```
SQL> desc user_objects

Name           Null Type
-------------- ---- -------------
OBJECT_NAME         VARCHAR2(128)
SUBOBJECT_NAME      VARCHAR2(30)
OBJECT_ID           NUMBER
DATA_OBJECT_ID      NUMBER
OBJECT_TYPE         VARCHAR2(19)
CREATED             DATE
LAST_DDL_TIME       DATE
TIMESTAMP           VARCHAR2(19)
STATUS              VARCHAR2(7)
TEMPORARY           VARCHAR2(1)
GENERATED           VARCHAR2(1)
SECONDARY           VARCHAR2(1)
NAMESPACE           NUMBER
EDITION NAME        VARCHAR2(30)
```



```
SQL> desc user_constraints

Name                 Null     Type
-------------------- -------- -------------
OWNER                         VARCHAR2(120)
CONSTRAINT_NAME      NOT NULL VARCHAR2(30)
CONSTRAINT_TYPE               VARCHAR2(1)
TABLE_NAME           NOT NULL VARCHAR2(30)
SEARCH_CONDITION              LONG
R_OWNER                       VARCHAR2(120)
R_CONSTRAINT_NAME             VARCHAR2(30)
DELETE_RULE                   VARCHAR2(9)
STATUS                        VARCHAR2(8)
DEFERRABLE                    VARCHAR2(14)
DEFERRED                      VARCHAR2(9)
VALIDATED                     VARCHAR2(13)
GENERATED                     VARCHAR2(14)
BAD                           VARCHAR2(3)
RELY                          VARCHAR2(4)
LAST_CHANGE                   DATE
INDEX_OWNER                   VARCHAR2(30)
INDEX_NAME                    VARCHAR2(30)
INVALID                       VARCHAR2(7)
VIEW_RELATED                  VARCHAR2(14)
```

User objects in the database

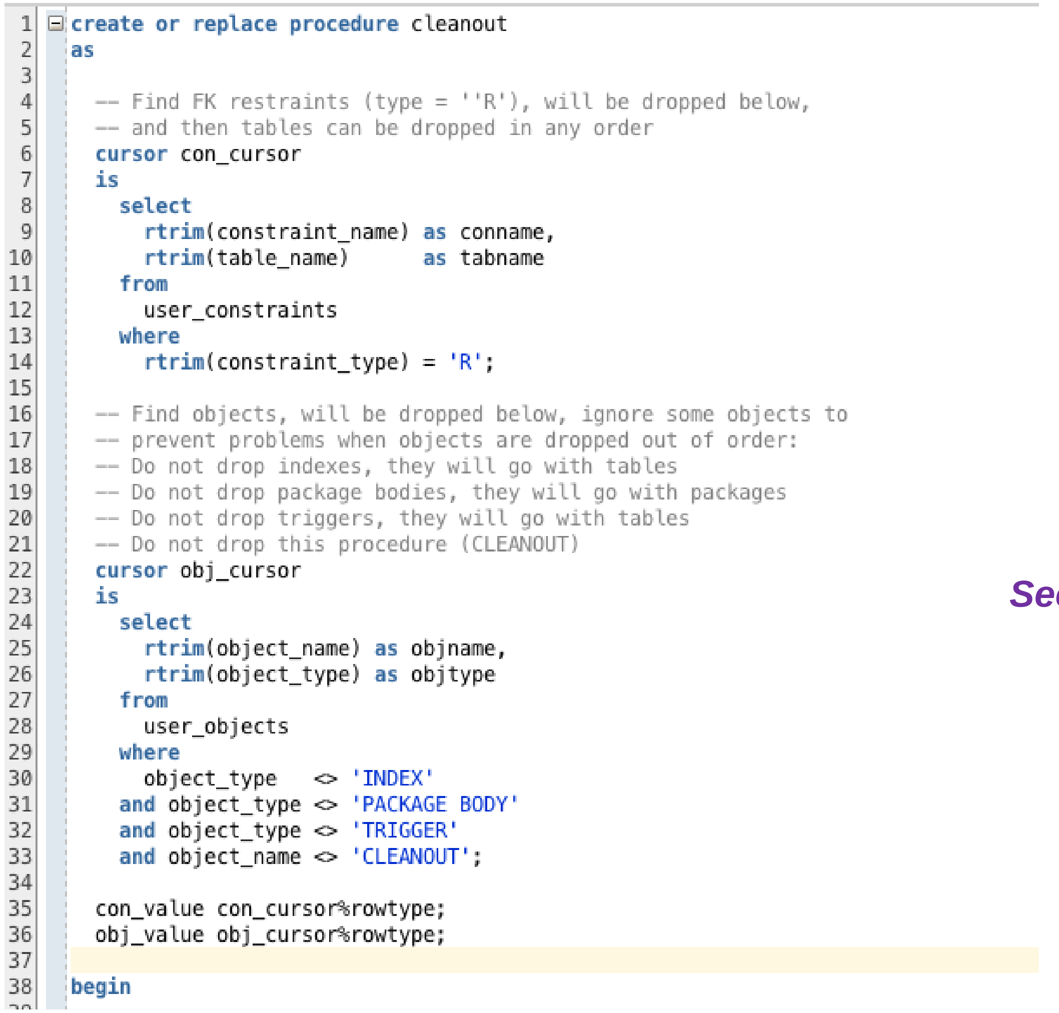
- Drop objects – tables, indexes, procedures, functions, packages, etc.
  - List all the objects a user owns:

```
select rtrim(object_name) as objname,
       rtrim(object_type) as objtype
from user_objects;
```

  - When dropping a table, the drop may be refused by Oracle due to FK constraints (type 'R' = restraints)
    - *Drop constraints first, then objects*
    - List FK constraints a user has in place:

```
select rtrim(constraint_name) as conname,
       rtrim(table_name) as tabname
       from user_constraints
       where rtrim(constraint_type) = 'R';
```

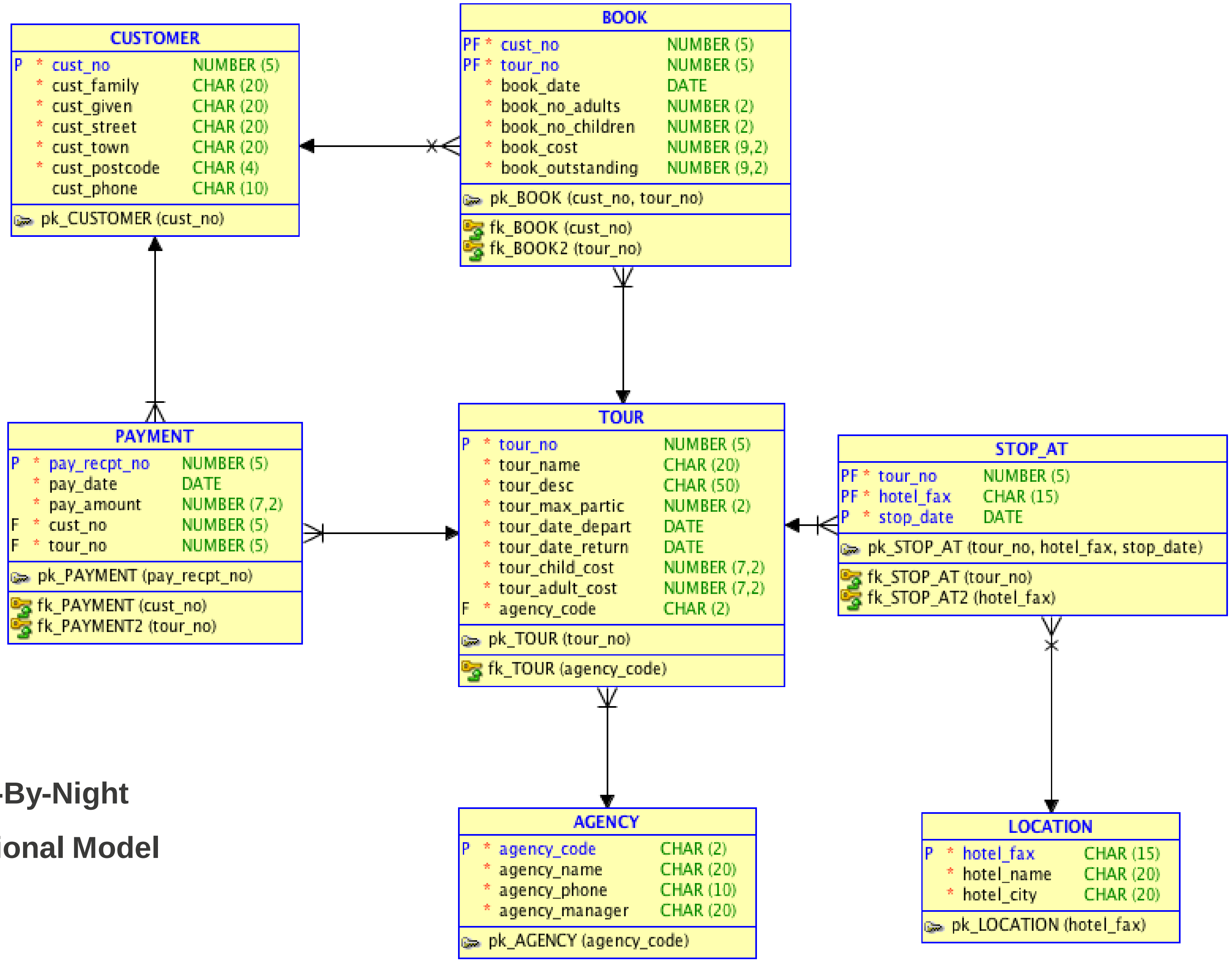
CLEANOUT



```
create or replace procedure cleanout
as

  -- Find FK restraints (type = ''R'), will be dropped below,
  -- and then tables can be dropped in any order
  cursor con_cursor
  is
    select
      rtrim(constraint_name) as conname,
      rtrim(table_name)      as tabname
    from
      user_constraints
    where
      rtrim(constraint_type) = 'R';

  -- Find objects, will be dropped below, ignore some objects to
  -- prevent problems when objects are dropped out of order:
  -- Do not drop indexes, they will go with tables
  -- Do not drop package bodies, they will go with packages
  -- Do not drop triggers, they will go with tables
  -- Do not drop this procedure (CLEANOUT)
  cursor obj_cursor
  is
    select
      rtrim(object_name) as objname,
      rtrim(object_type) as objtype
    from
      user_objects
    where
      object_type    <> 'INDEX'
    and object_type <> 'PACKAGE BODY'
    and object_type <> 'TRIGGER'
    and object_name <> 'CLEANOUT';

  con_value con_cursor%rowtype;
  obj_value obj_cursor%rowtype;

begin
```

See code available on Moodle

Packages



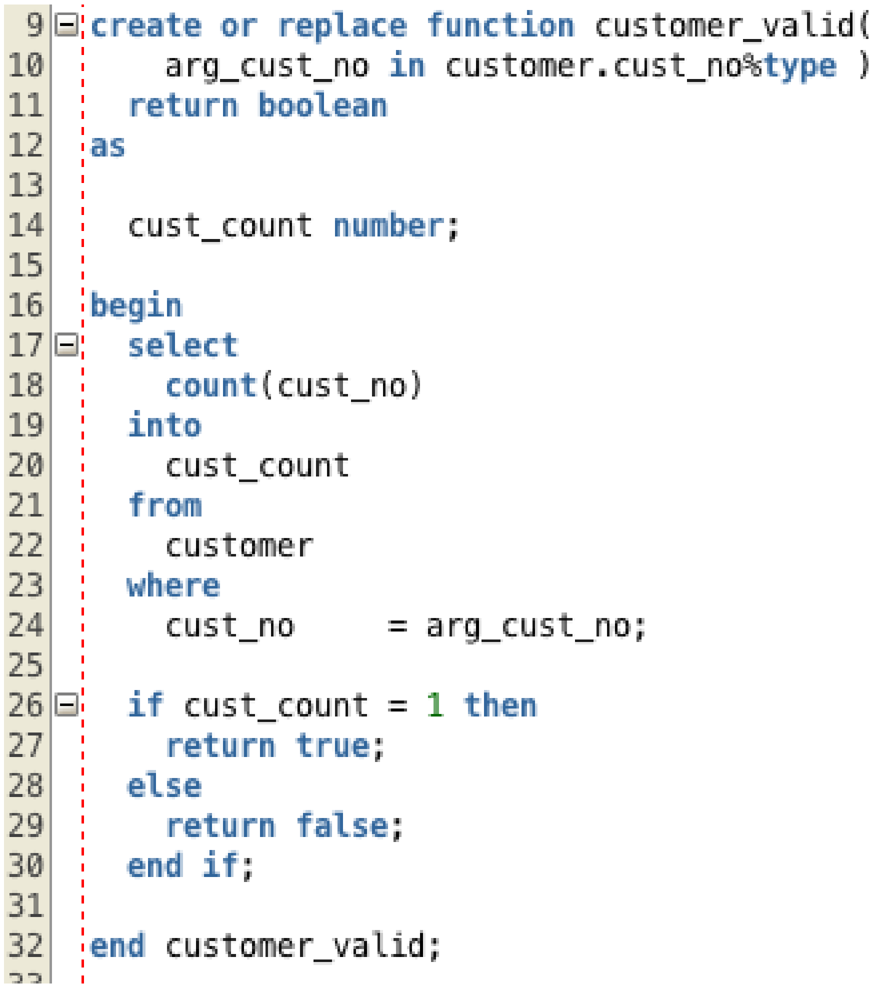
Fly-By-Night

Relational Model

PL/SQL Functions

- A customer will make BOOKings and PAYMENTs
  - Active tables compared with tour details and AGENCY
- One important validation when accepting bookings/payments
  - Is this a valid customer? (we wish to *avoid raising Oracle database errors*, difficult to automatically handle in calling program)
  - Result in repeated validation code in our procedures
    - Problems with such an approach? ***– problems with maintenance***
    - Create a function to complete the validation and call it repeatedly:
      - Algorithm? ***– select count(*) …***
      - Parameters? ***– value to check (custno or tourno)***
      - Return? **– is the value valid (PL/SQL use boolean)**

Function to validate customer



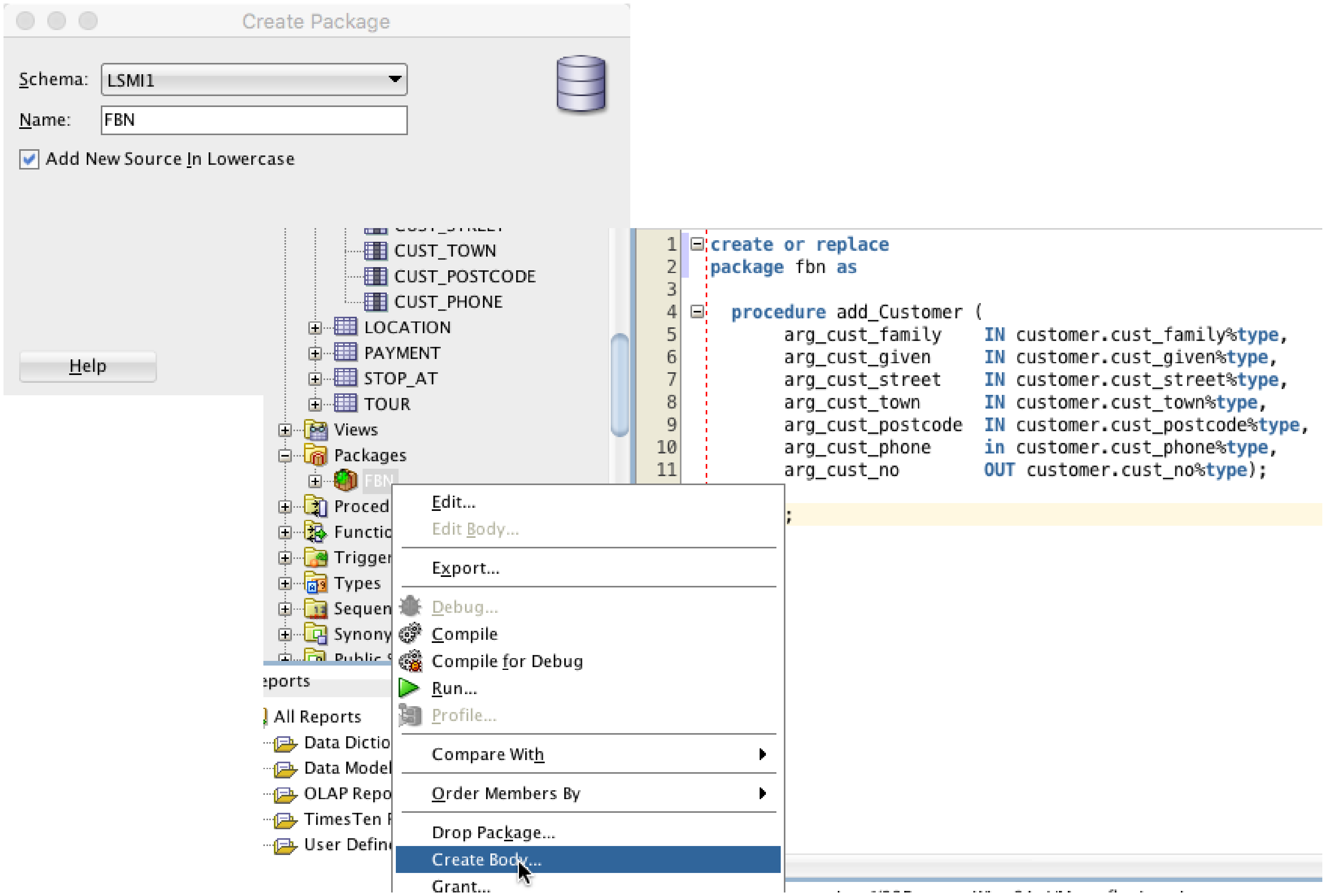
```
create or replace function customer_valid(
    arg_cust_no in customer.cust_no%type )
  return boolean
as

  cust_count number;

begin
  select
    count(cust_no)
  into
    cust_count
  from
    customer
  where
    cust_no     = arg_cust_no;

  if cust_count = 1 then
    return true;
  else
    return false;
  end if;

end customer_valid;
```

PL/SQL Packages

- Related PL/SQL procedures and functions can be grouped into one package
  - Keeps all related PL/SQL objects together
  - Consists of
    - Package Specification
      - List of procedure and function headers
    - Package Body
      - Implementation of procedures and functions
- To execute a procedure or function of a package use:
  - package_name.procedure_name (parameters)
- The creator of a package would normally ***grant execute rights*** on the package to allow other users to run it:
  - **grant execute on my_package to** user/public;
  - users then run package with the same rights as the package creator

Create a package

- Can be created from:
  - the SQL Window, or
  - the SQL Developer GUI
- Creation from the GUI
  - Right click package, select “New Package” and name
    - Enter procedure declaration
  - Must be compiled before it is saved to the database
  - Creates package specification (spec)
  - Right click package – select “Create Body …”
- Note not all procedures in the package need to be declared in the spec
  - Used to create “hidden” internal code to prevent duplication of PL/SQL code in body



Schema: LSMI1

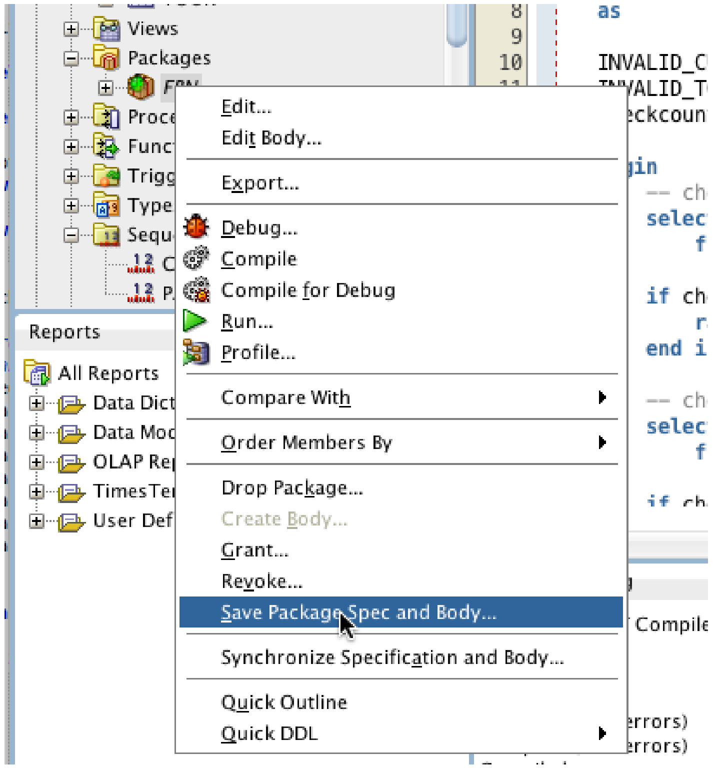
Name: FBN

[✓] Add New Source In Lowercase

```
create or replace
package fbn as

  procedure add_Customer (
       arg_cust_family     IN customer.cust_family%type,
       arg_cust_given      IN customer.cust_given%type,
       arg_cust_street     IN customer.cust_street%type,
       arg_cust_town       IN customer.cust_town%type,
       arg_cust_postcode   IN customer.cust_postcode%type,
       arg_cust_phone      in customer.cust_phone%type,
       arg_cust_no         OUT customer.cust_no%type);

;
```

Saving a package from the database

- Export package
  - Right click package, select “Save Package Spec and Body”
    - Saves to a .sql file based on the name of the package, save to ***SVN folder***



Call procedure in package

To call the procedure add_customer in the package fbn, we would use:

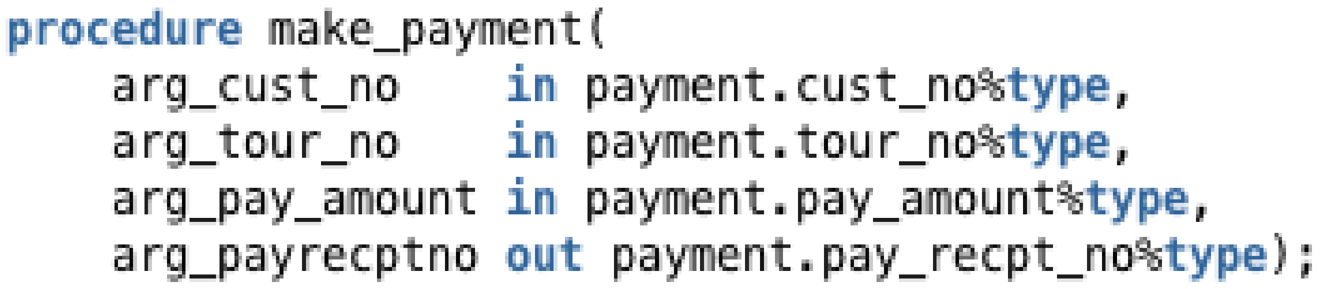
A. exec add_customer (….);
B. exec add_customer.fbn (….);
C. exec fbn.add_customer (….);
D. exec fbn(add_customer (…));

Procedure to accept a new payment

- From schema, sequence available:
  - CREATE SEQUENCE payrecptno START WITH 100;
- Parameters/validation
  - pay_recpt_no
    - **OUT parameter**
  - pay_date
    - **No parameter required, sysdate**
  - pay_amount
    - **IN parameter, > 0 and <= the amount outstanding**
  - cust_no
    - **IN parameter, valid (existing) customer**
  - tour_no
    - **IN parameter, check valid (existing) tour**
  - cust_no & tour_no
    - **validation – is this customer booked on this tour?**
- Tasks: if valid cust_no, tour_no and is booked on this tour:
  - **insert payment**
  - **reduce book_outstanding**

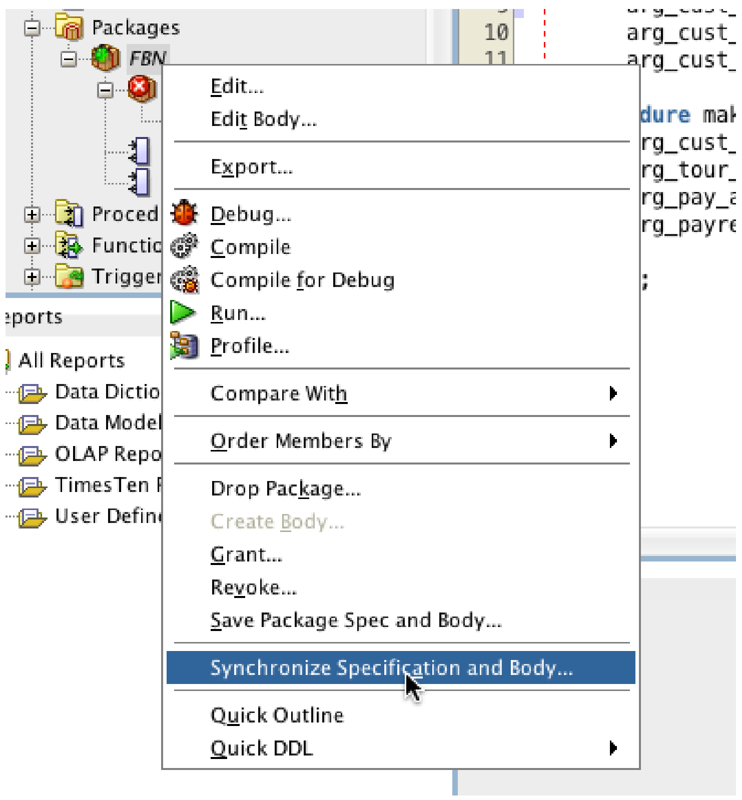
Procedure header to make a payment

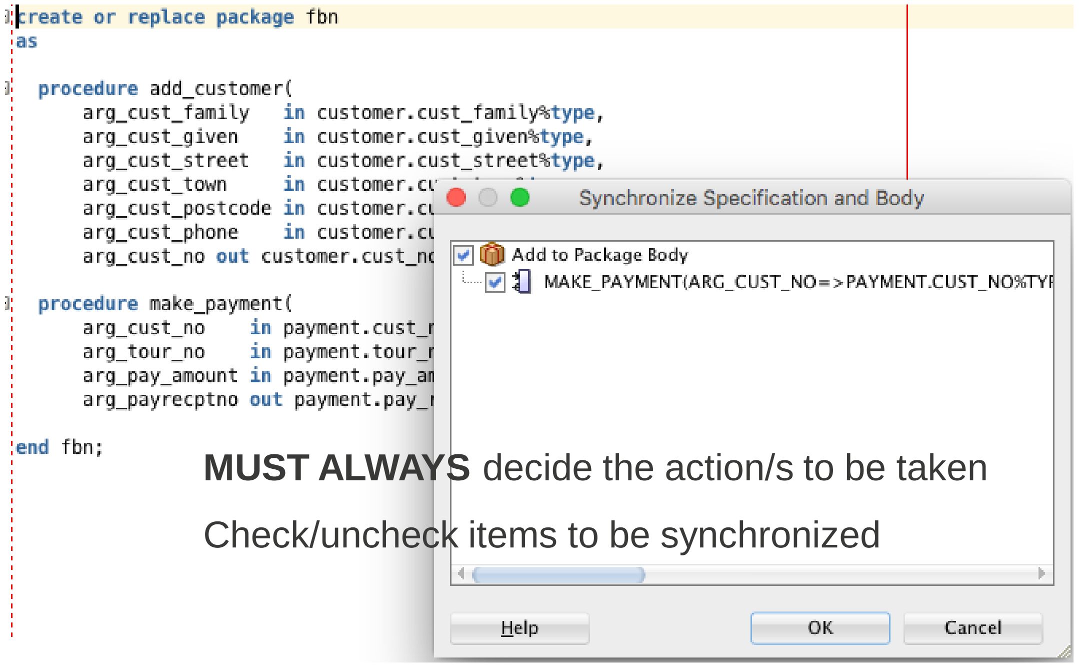
- Add to Package Spec:



```
procedure make_payment(
    arg_cust_no    in payment.cust_no%type,
    arg_tour_no    in payment.tour_no%type,
    arg_pay_amount in payment.pay_amount%type,
    arg_payrecptno out payment.pay_recpt_no%type);
```

Updating package body after adding a new procedure





```
create or replace package fbn
as

  procedure add_customer(
      arg_cust_family   in customer.cust_family%type,
      arg_cust_given    in customer.cust_given%type,
      arg_cust_street   in customer.cust_street%type,
      arg_cust_town     in customer.cu
      arg_cust_postcode in customer.cu
      arg_cust_phone    in customer.cu
      arg_cust_no out customer.cust_nc

  procedure make_payment(
      arg_cust_no    in payment.cust_r
      arg_tour_no    in payment.tour_r
      arg_pay_amount in payment.pay_an
      arg_payrecptno out payment.pay_r

end fbn;
```

Synchronize Specification and Body
Add to Package Body
MAKE_PAYMENT(ARG_CUST_NO=>PAYMENT.CUST_NO%TYP
Help
OK
Cancel

MUST ALWAYS decide the action/s to be taken

Check/uncheck items to be synchronized

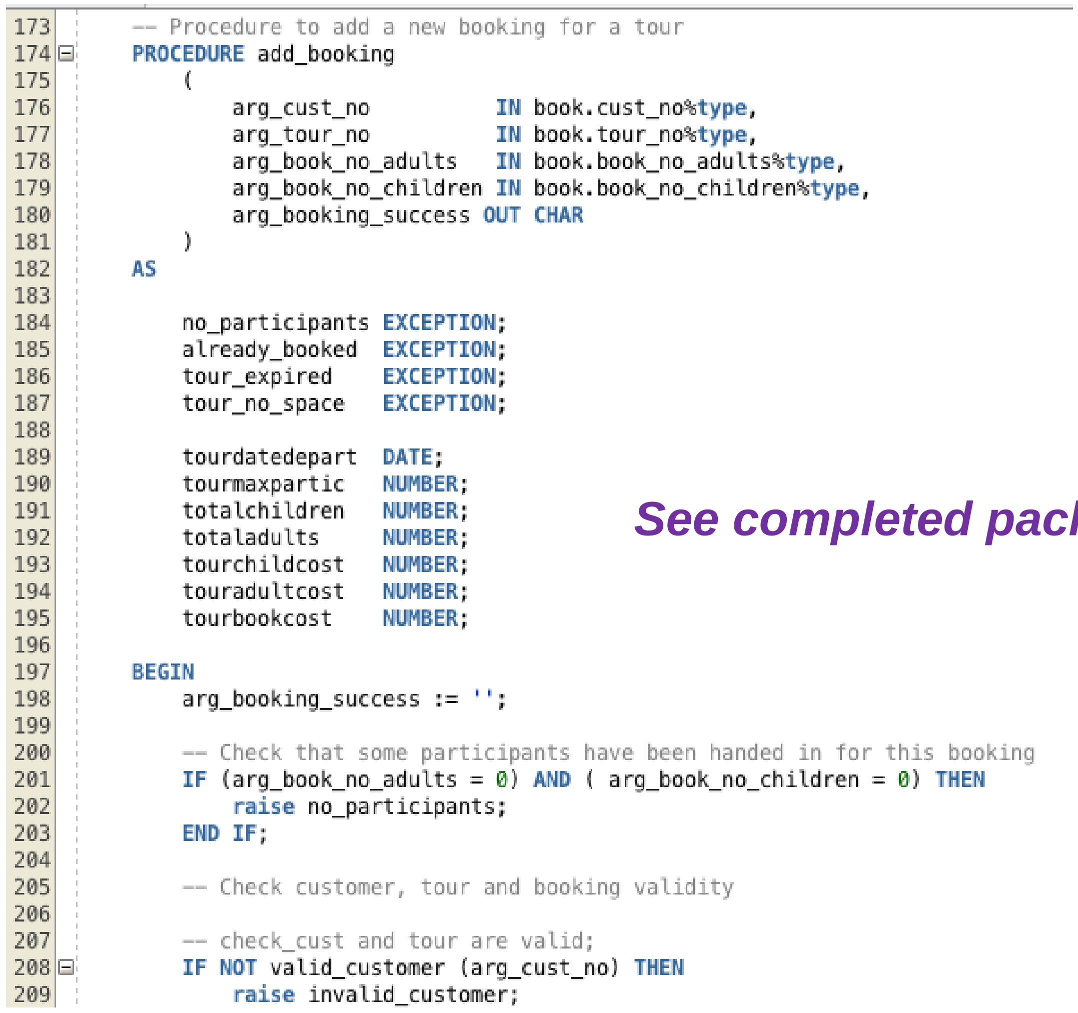
Workflow to follow:

Step 1: add procedure header to package spec

Step 2: sync to create entry in body

Step 3: Code PL/SQL body

Add new PL/SQL code to package body



```
-- Procedure to add a new booking for a tour
PROCEDURE add_booking
    (
        arg_cust_no          IN book.cust_no%type,
        arg_tour_no          IN book.tour_no%type,
        arg_book_no_adults   IN book.book_no_adults%type,
        arg_book_no_children IN book.book_no_children%type,
        arg_booking_success OUT CHAR
    )
AS

    no_participants EXCEPTION;
    already_booked  EXCEPTION;
    tour_expired    EXCEPTION;
    tour_no_space   EXCEPTION;

    tourdatedepart  DATE;
    tourmaxpartic   NUMBER;
    totalchildren   NUMBER;
    totaladults     NUMBER;
    tourchildcost   NUMBER;
    touradultcost   NUMBER;
    tourbookcost    NUMBER;

BEGIN
    arg_booking_success := '';

    -- Check that some participants have been handed in for this booking
    IF (arg_book_no_adults = 0) AND ( arg_book_no_children = 0) THEN
        raise no_participants;
    END IF;

    -- Check customer, tour and booking validity

    -- check_cust and tour are valid;
    IF NOT valid_customer (arg_cust_no) THEN
        raise invalid_customer;
```

See completed package code on Moodle

Package Testing

- Bind variables can be used in PL/SQL and SQL (in the SQL Developer SQL Worksheet)
- Test Harness for package
  - A single SQL file which tests the ***full*** functionality of your package
  - Include a spool command on the first line to capture output into a text file

```
-- null phone number
var new_custno number
exec  fbn.add_customer('Smith',...........,null,:new_custno);
print new_custno

select * from customer where cust_no = :new_custno;
```

Summary

- Discussed Oracle system tables: user_objects and user_constraints, which are a part of Oracle Data Dictionary.
- Oracle data dictionary refers to a read-only set of tables that provides information about the Oracle database.
- The user_objects and user_constraints tables record all objects and constraints created/owned by the user.
- Explained the “cleanout” procedure which is used to remove all Oracle objects and constraints owned by the user.
- Discussed the purposes of having a PL/SQL package
- Demonstrated how to create, execute and save a package for future use
- Demonstrated how to create and run a test harness for a package, which is creating a single SQL file to test the ***full*** functionality of the package