Part 18: Application Programming II (Stored Procedures, Triggers)

References:

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Edition, 1999. Section 10.5, "Programming Oracle Applications"
- R. Sunderraman: Oracle Programming A Primer, Addison-Wesley, 1999. Chapter 4, "PL/SQL"
- Michael Gertz: Oracle/SQL Tutorial, 1999. http://www.db.cs.ucdavis.edu/teaching/sqltutorial/
- Oracle8 Application Developer's Guide, Oracle Corporation, 1997, Part No. A58241-01.
- PL/SQL User's Guide and Reference, Oracle Corporation, 1997, Part No. A58236-01.

Warning:

• These slides are old (from 1999). Today, Java can be used as an alternative to PL/SQL.

Objectives

- You should know some advantages of using stored procedures.
- You should have at least some basic impression of the Oracle PL/SQL syntax.
- You should be able to explain what triggers are.
- You should know some applications of triggers.
- You should have some basic impression of the trigger syntax in Oracle.





2. Triggers in Oracle

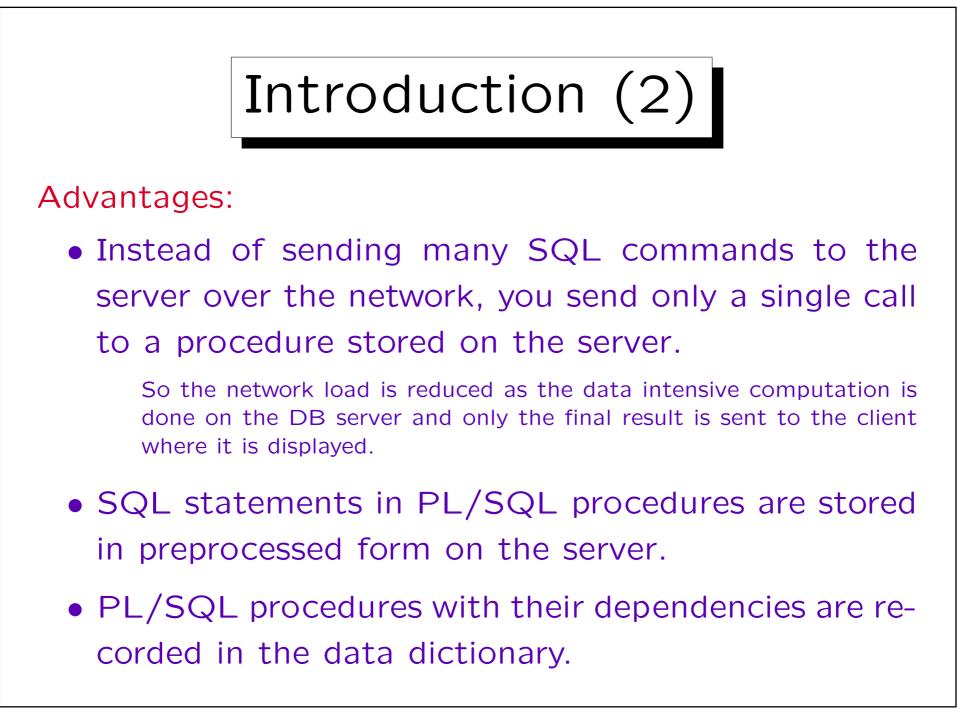
Stefan Brass: Datenbanken I

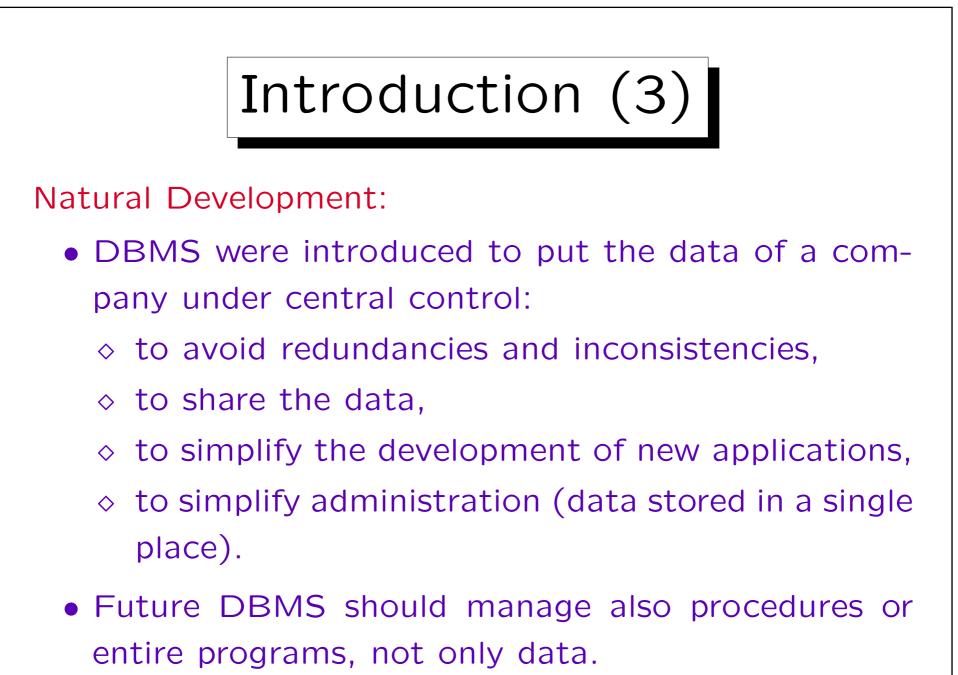


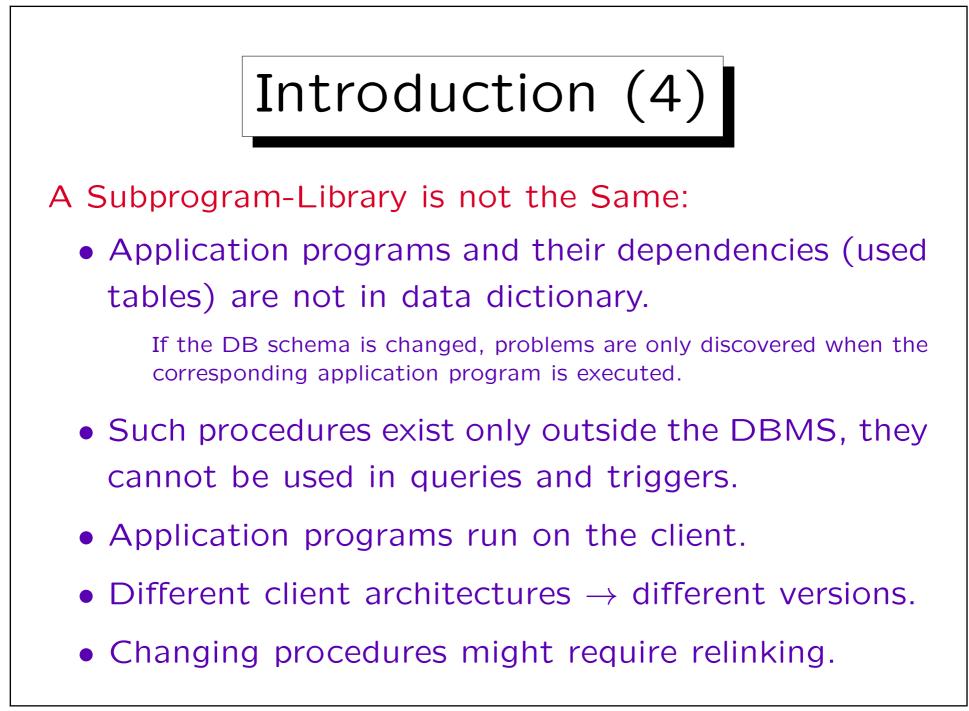
- PL/SQL is an imperative programming language with variables and the usual control structures.
- PL/SQL has an especially good interface to SQL, e.g. the same type system as the Oracle database.
- Procedures written in PL/SQL can be stored and executed on the database server.
- PL/SQL is not meant for creating user interfaces.

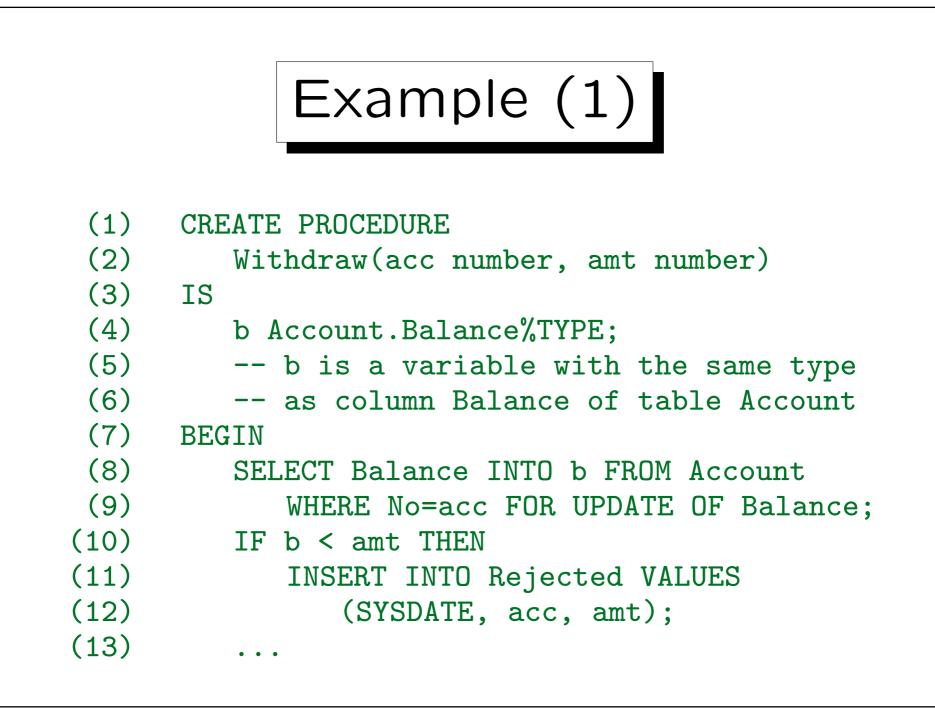
However, it is used in other Oracle user interface products. All input/output must be done with parameters or database tables. E.g., if a web page is created, it is first stored in the DB. It is a "server-side" programming language.

18-4



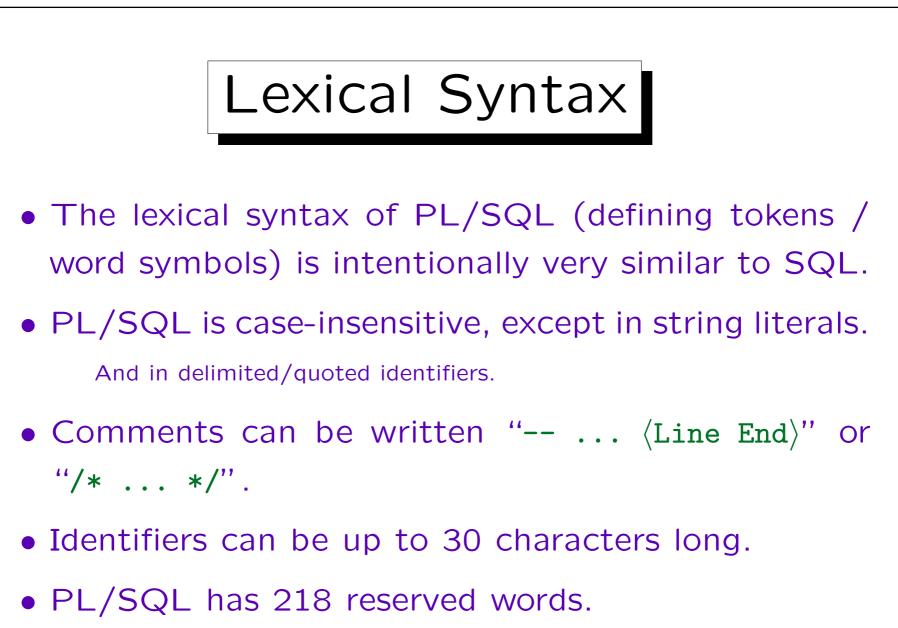




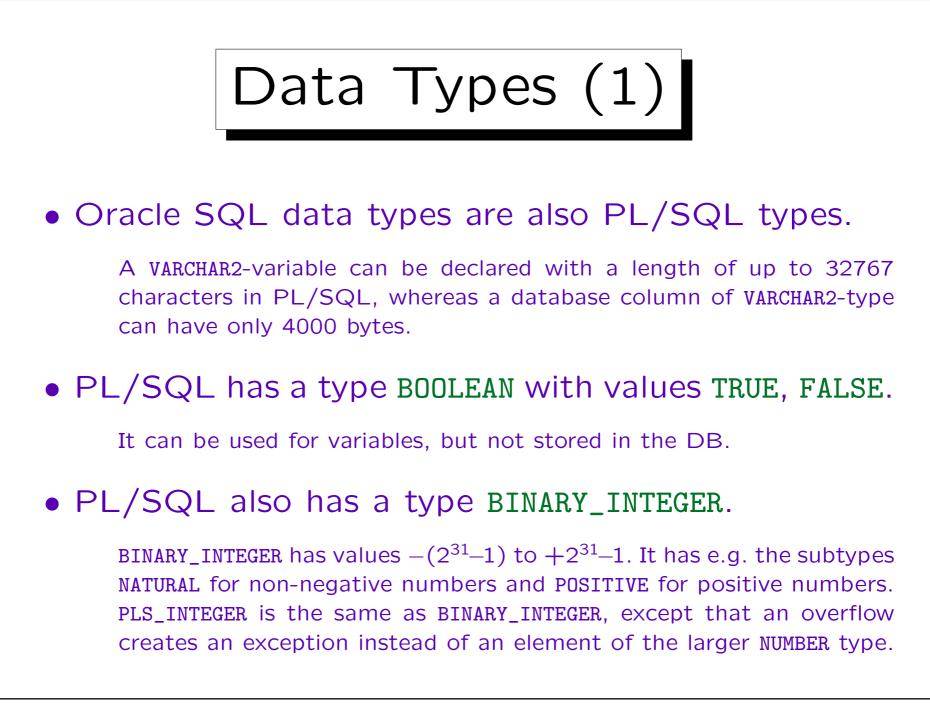


Example (2)

```
(13)
         ELSE -- Current Balance is sufficient.
(14)
             b := b - amt;
(15)
             UPDATE Account SET Balance = b
(16)
                WHERE No = acc;
(17)
             INSERT INTO Done
(18)
                VALUES (SYSDATE, acc, amt);
(19)
         END IF;
(20)
         COMMIT;
(21) END;
```



This includes SQL reserved words, plus about 100 new ones.

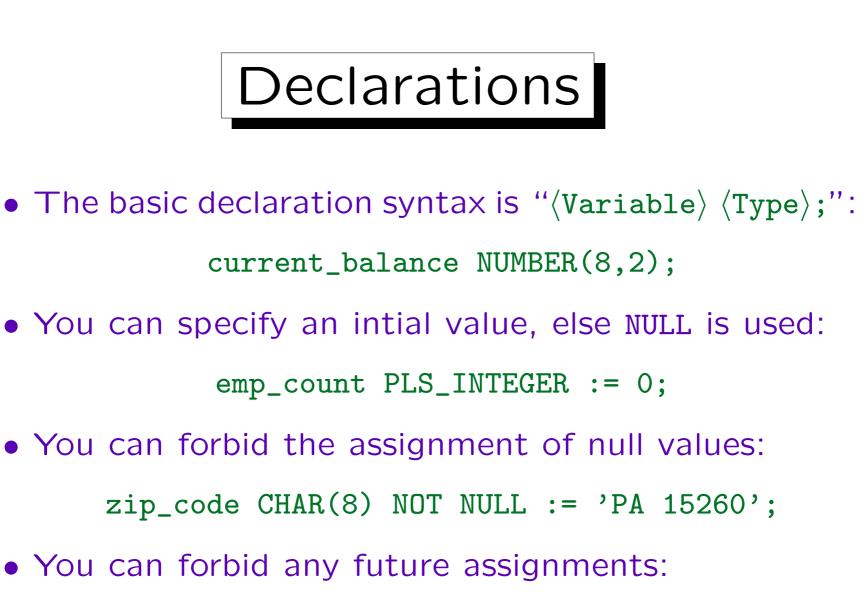


Data Types (2)

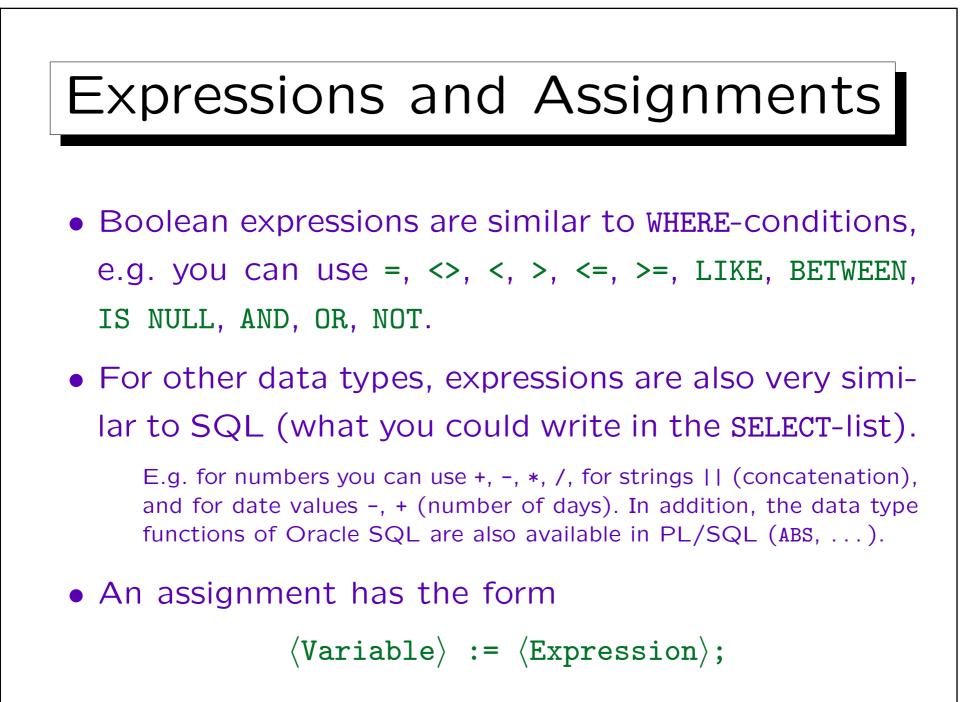
- Procedure parameters can have only unconstrained data types, e.g. "NUMBER", but not "NUMBER(4)".
- PL/SQL has also record types, so you can store an entire database row into a single variable.
- PL/SQL also has collection types
 - ◊ VARRAY (array of varying size) and
 - ◇ TABLE (something between a set and an array).
- PL/SQL allows you to define your own data types.

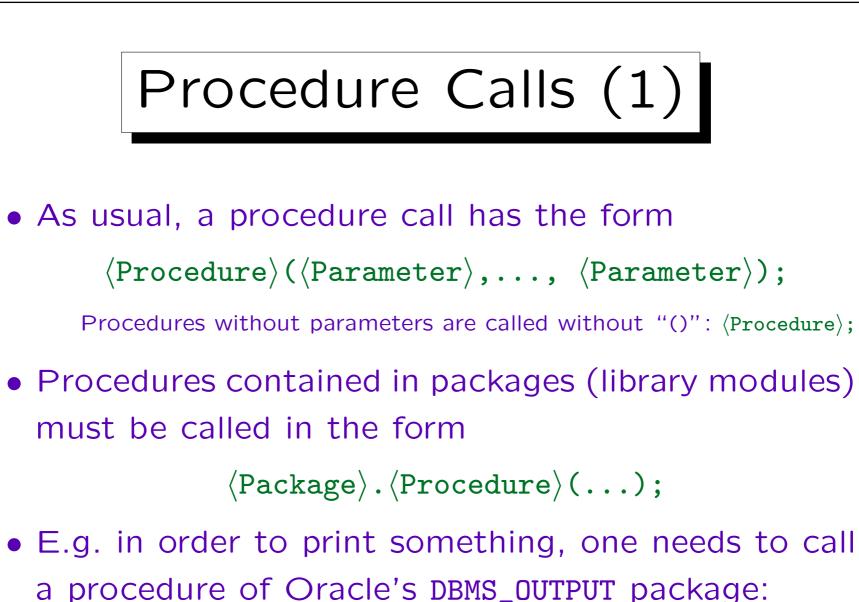
Data Types (3)

- Instead of explicitly specifying a type, one can also use the following special constructs:
 - ◇ TABLE.COLUMN%TYPE: the type of a DB column,
 - ◇ TABLE%ROWTYPE: the record-type corresponding to a DB row, and
 - ◇ VARIABLE%TYPE: the type of another variable.

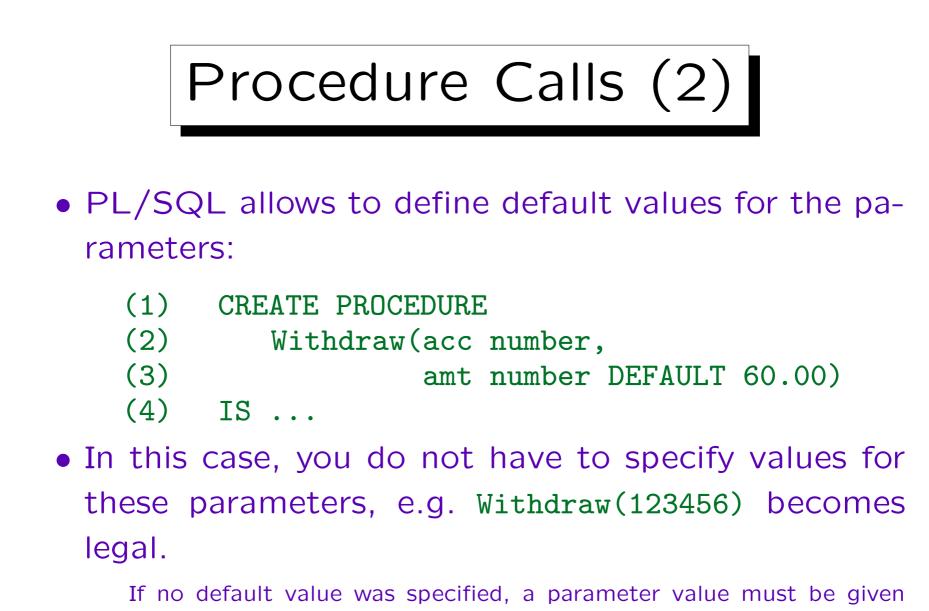


credit_limit CONSTANT NUMBER := 100.00;

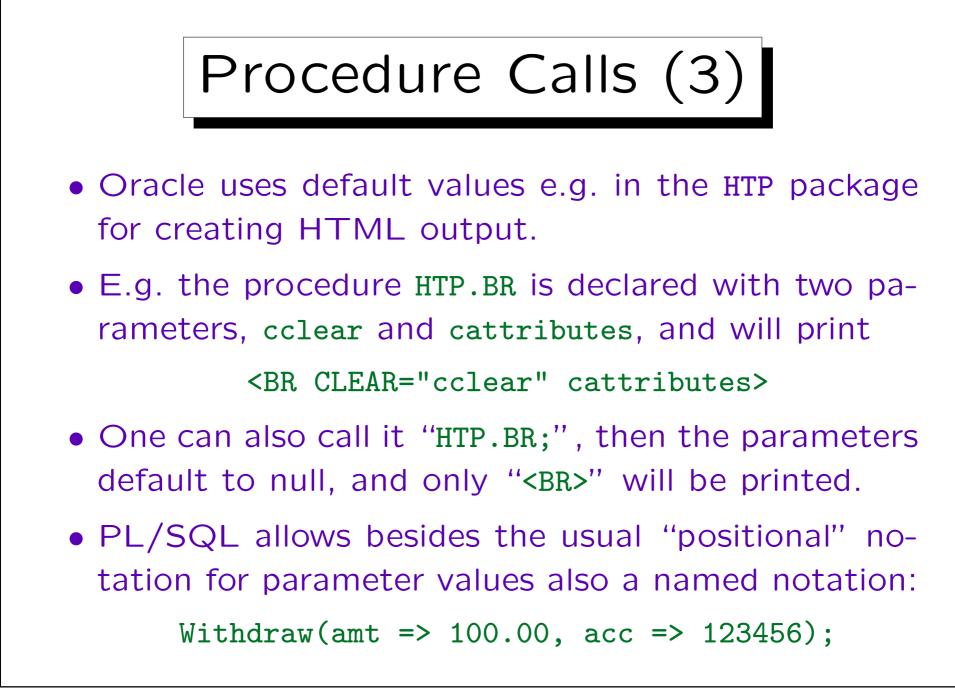


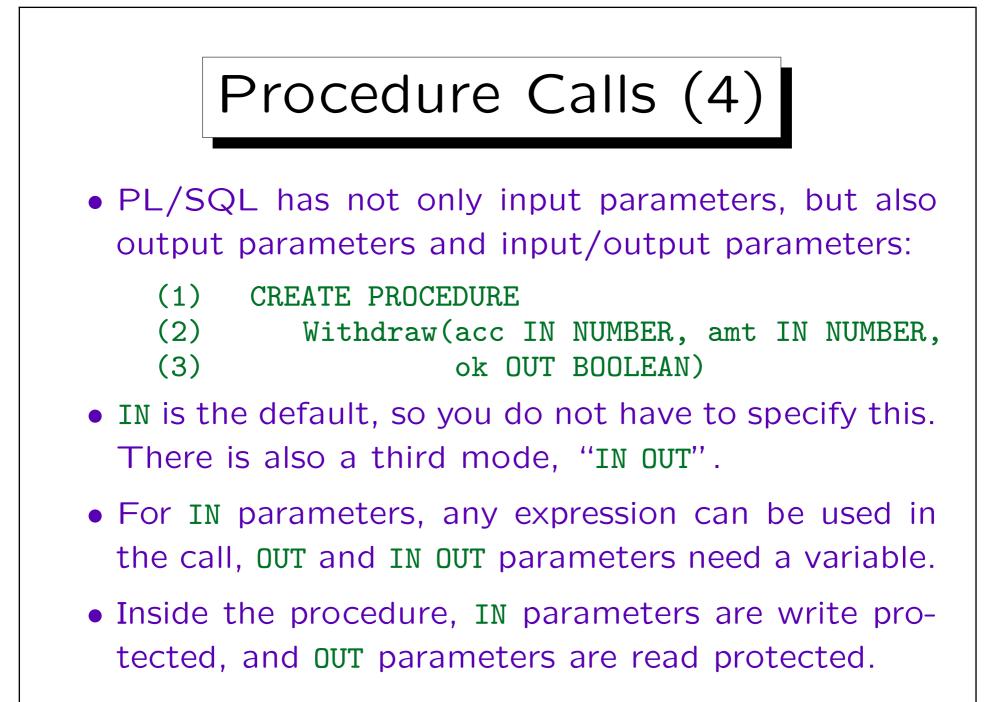


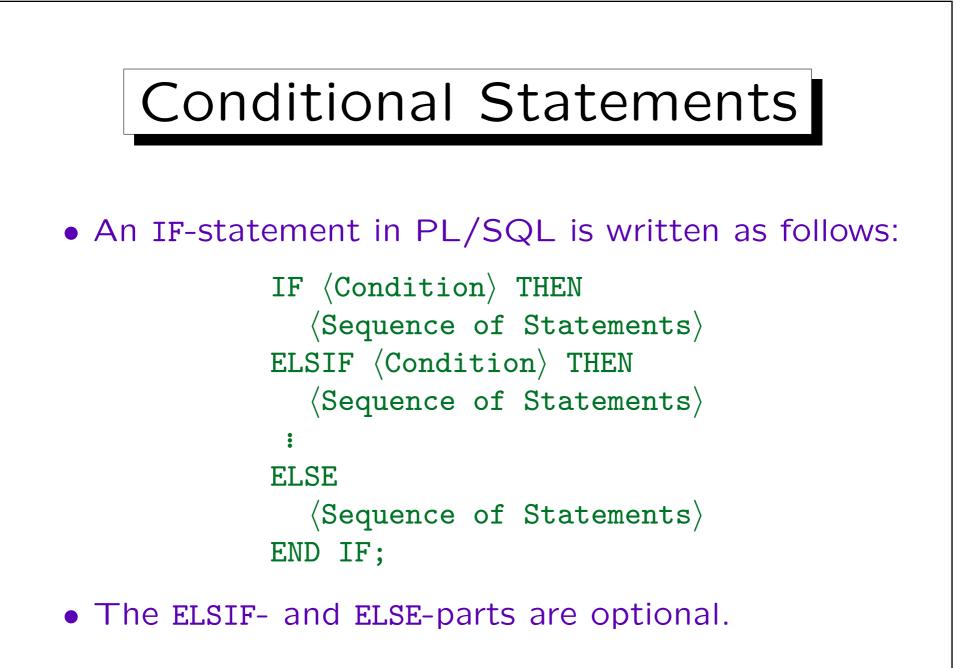
DBMS_OUTPUT.PUT_LINE('Wrong Employee name');

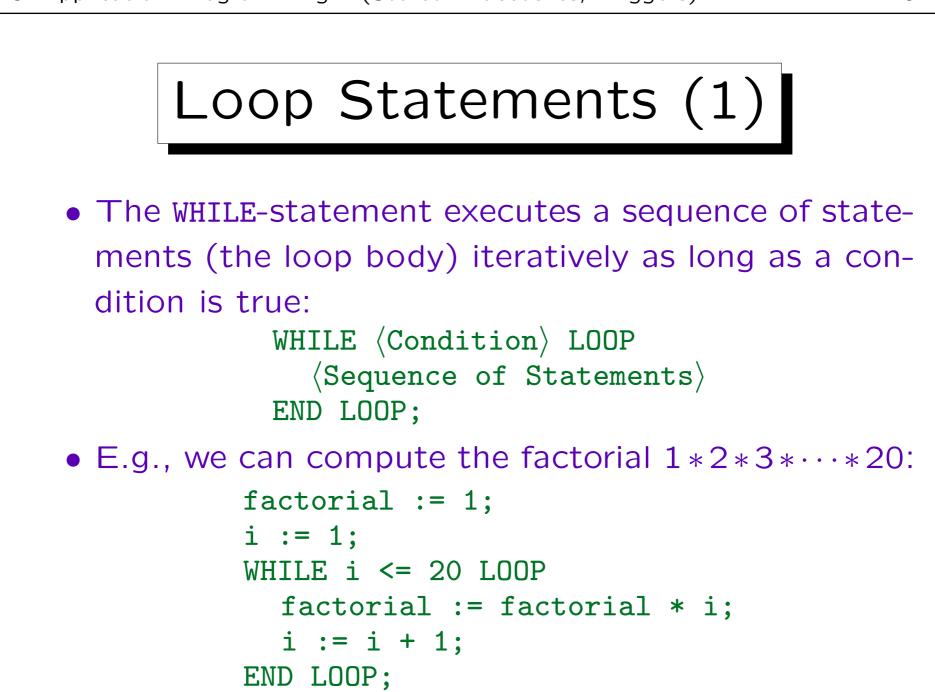


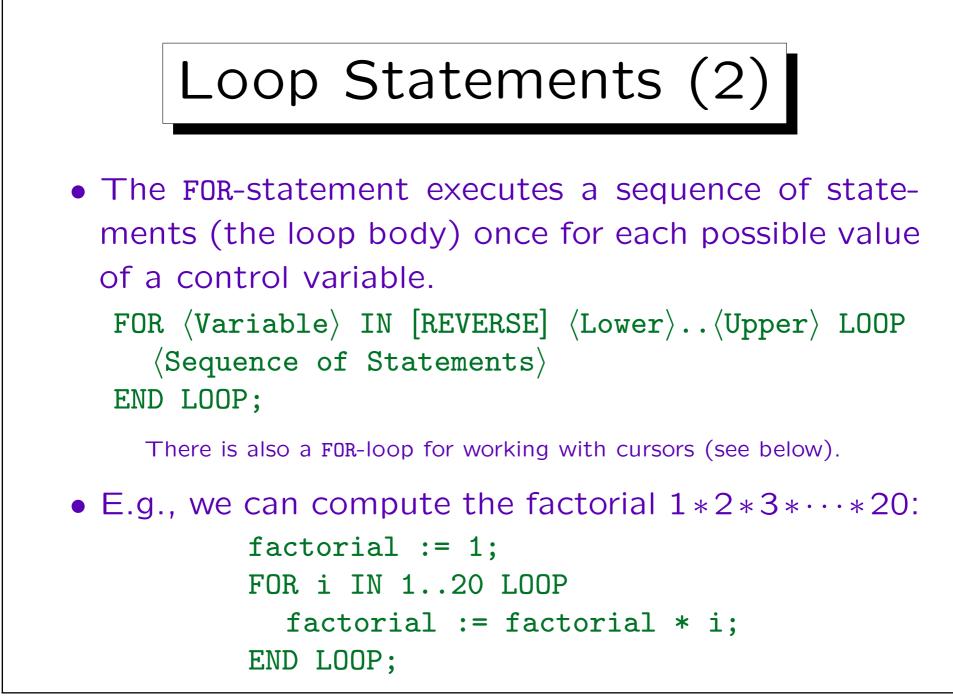
in the call. If the declared default value is "NULL", the procedure can detect whether a value was actually specified.



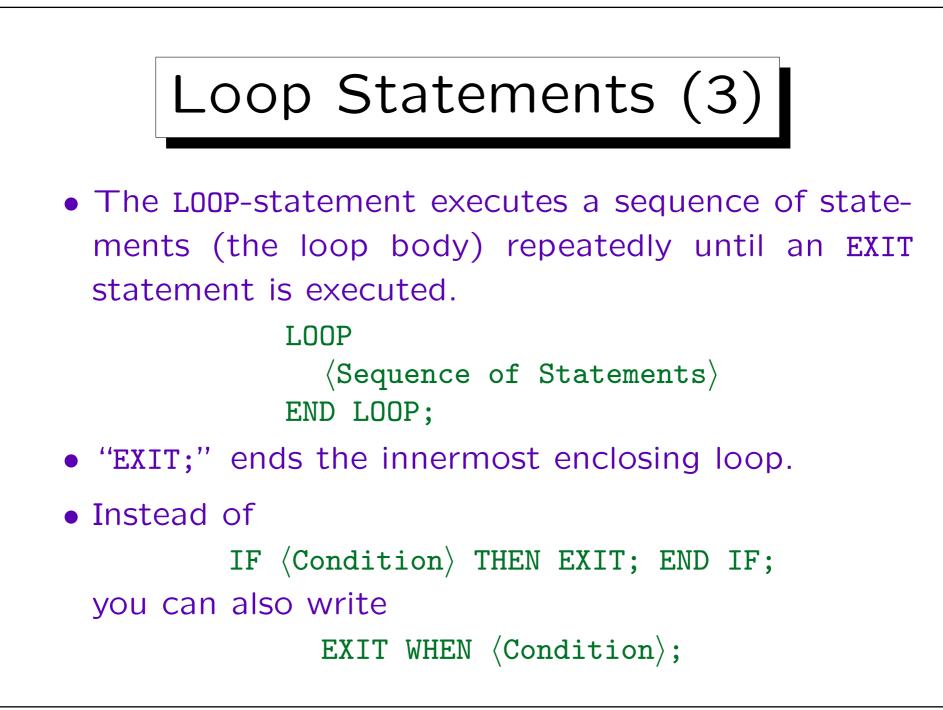


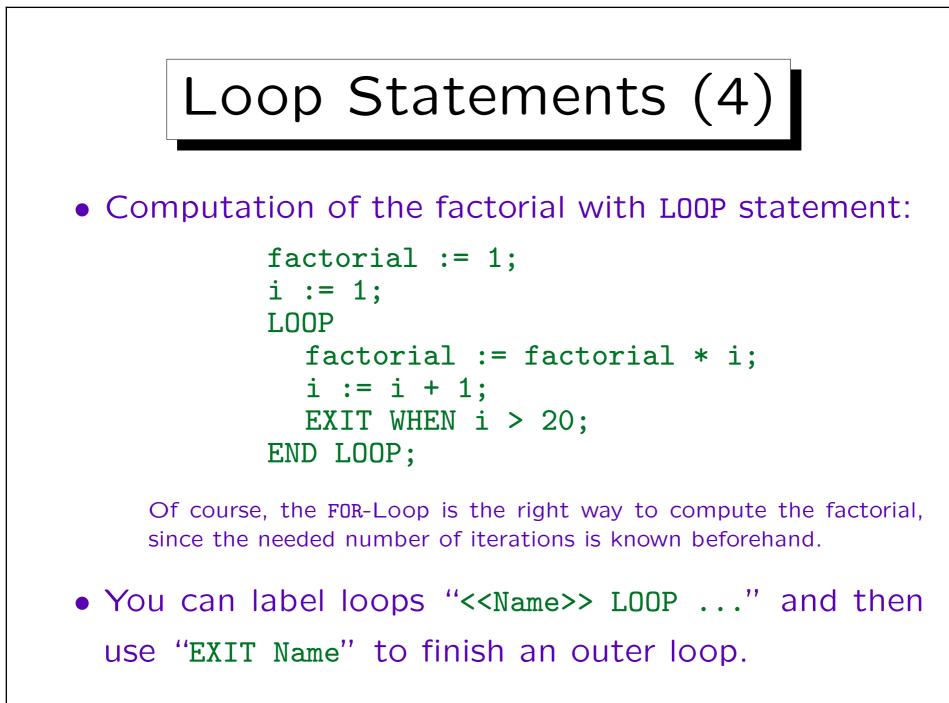


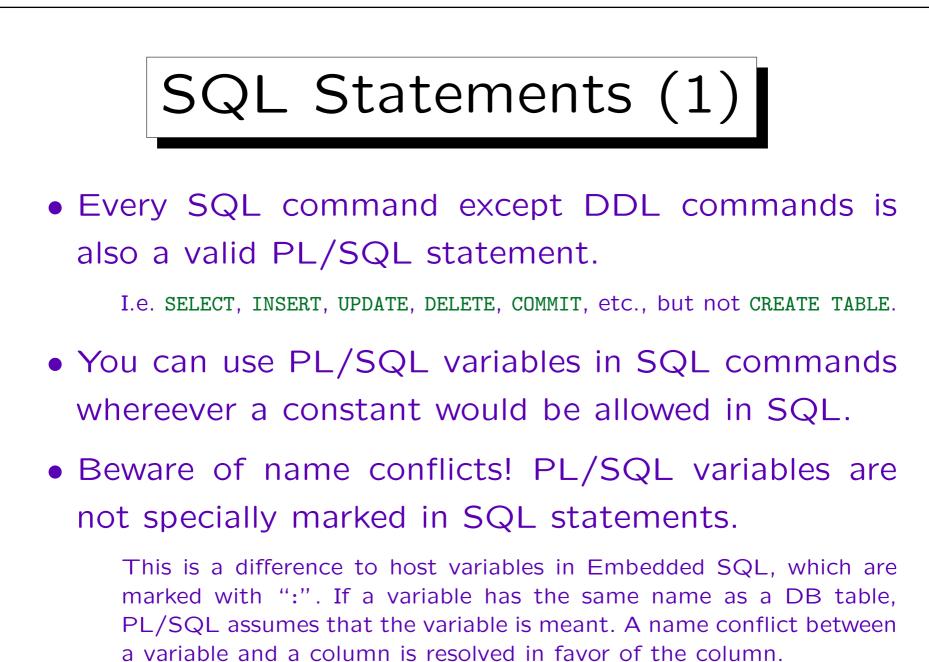


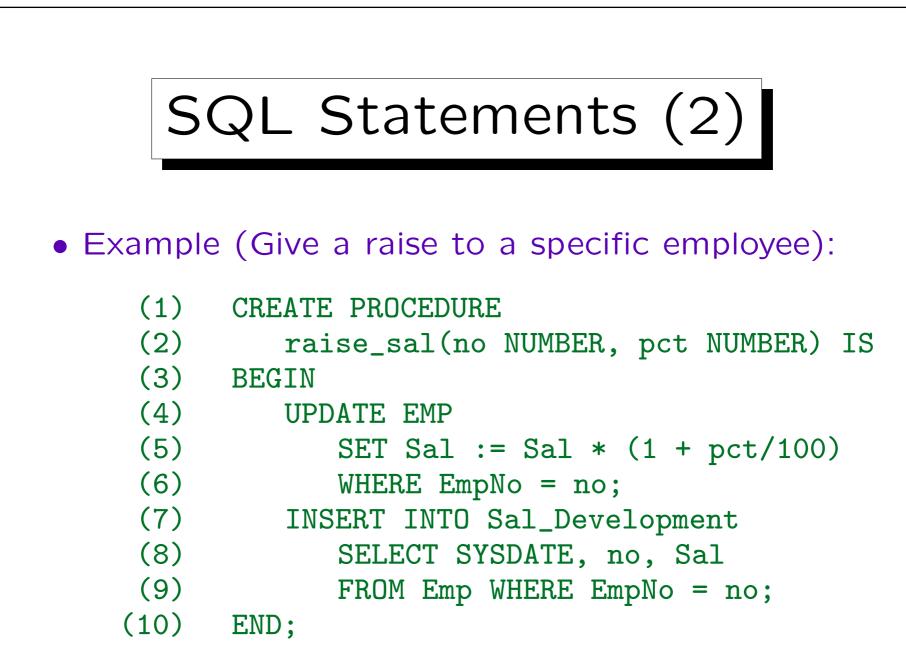


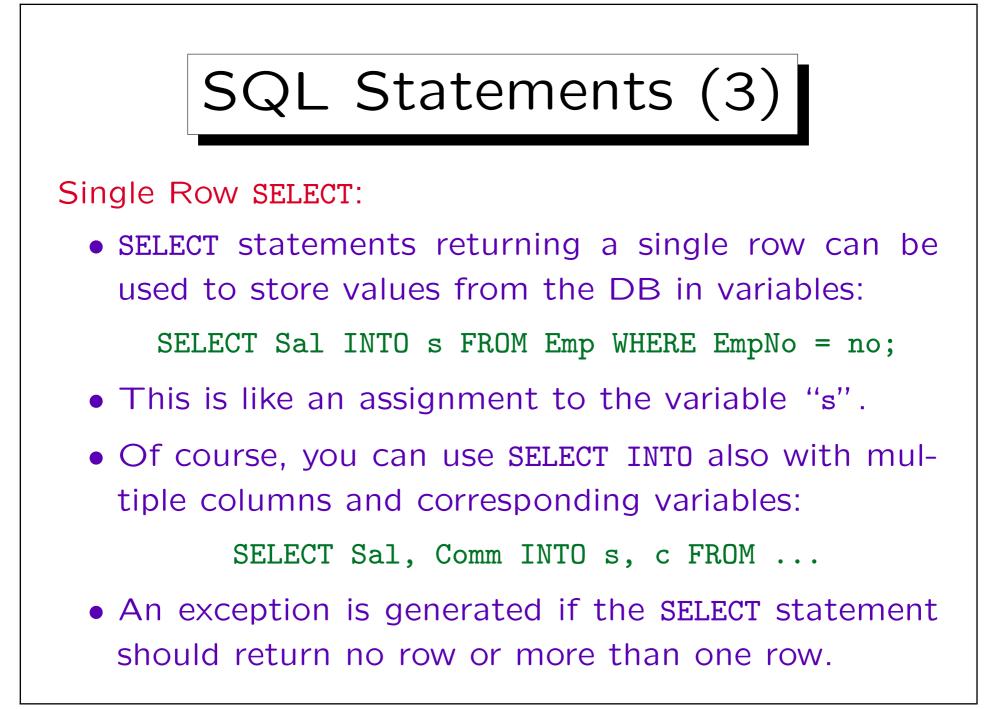
Stefan Brass: Datenbanken I









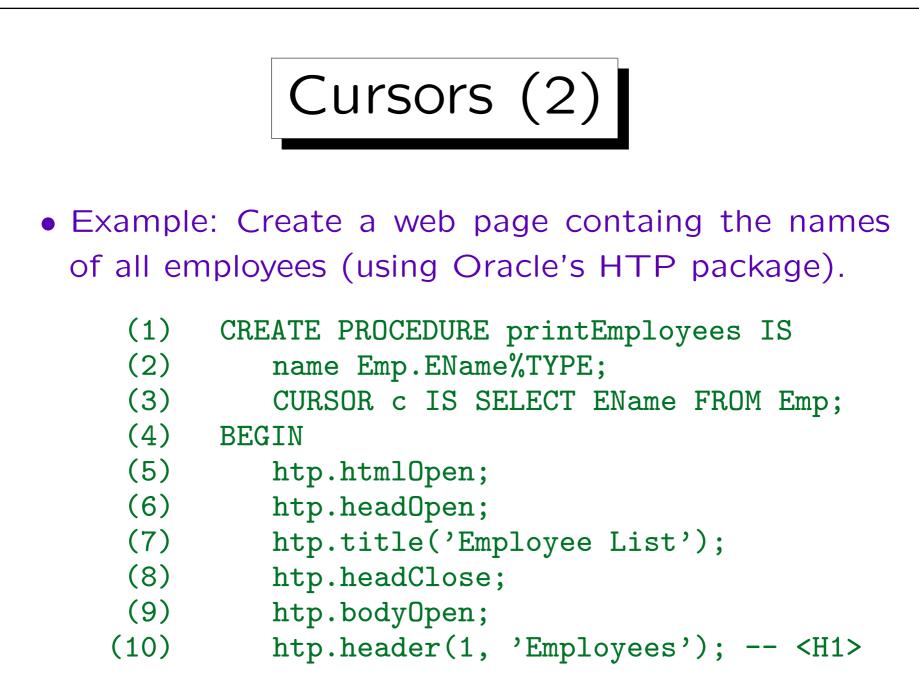


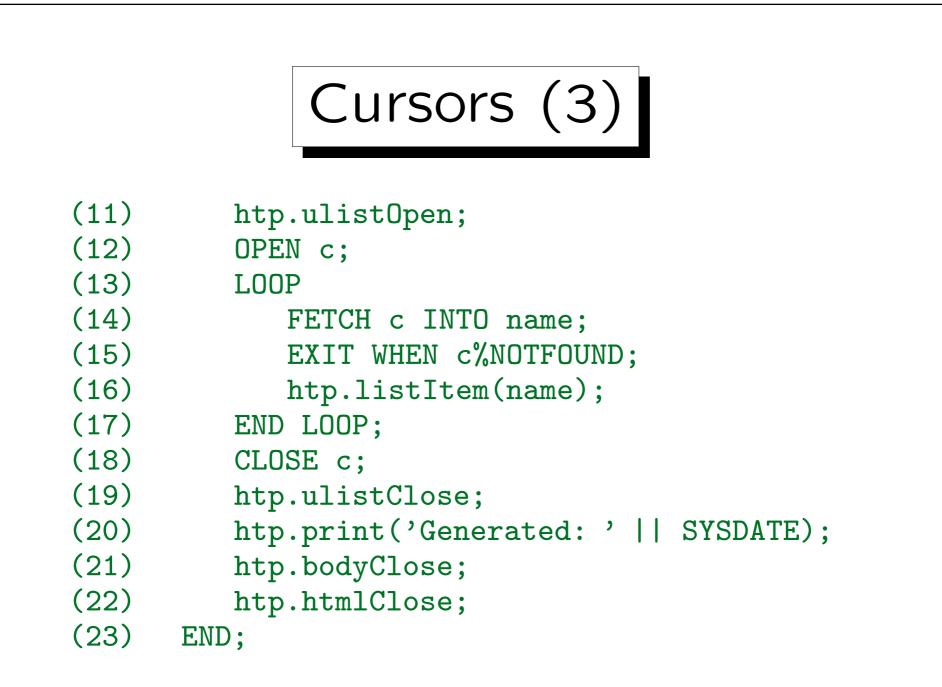


- If a SELECT statement can return more than one row, you need a cursor (or a special kind of FOR-Loop).
- The cursor must be declared in the declaration section (the part before the BEGIN).

You specify an SQL query when you declare the cursor.

- Then you OPEN the cursor (executes query).
- Next, you use a FETCH-statement in a loop to get the column values for each output row.
- Finally, you CLOSE the cursor.

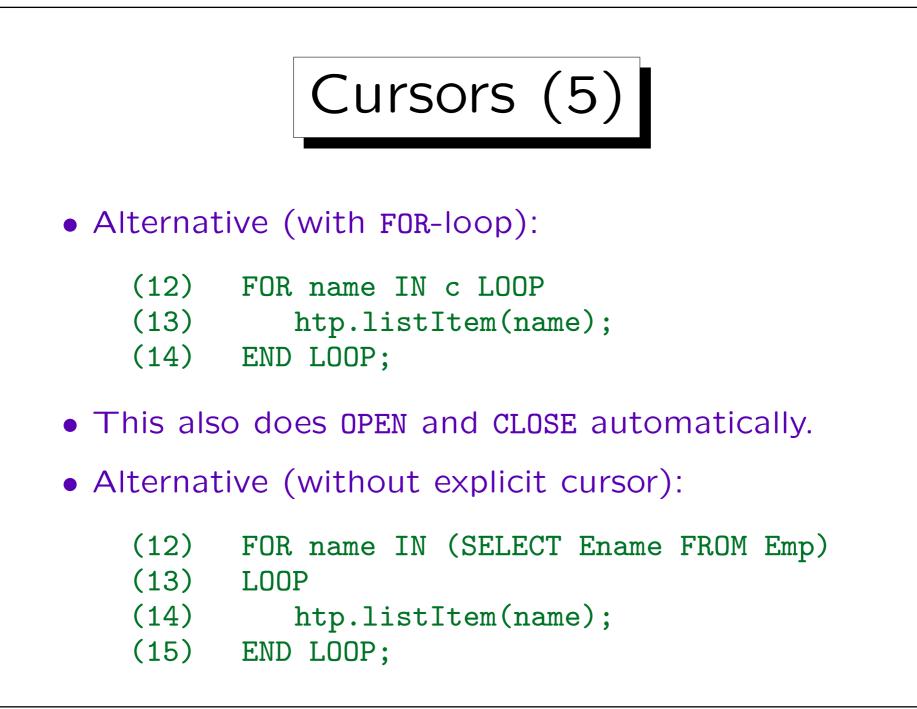




Cursors (4)

• Alternative (with WHILE-loop):

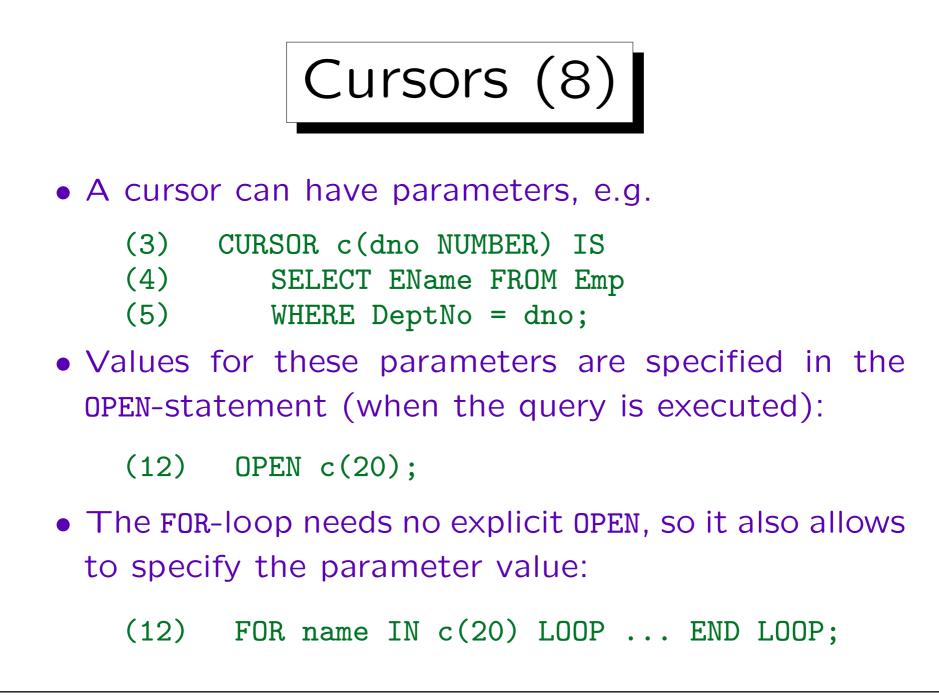
(12)	OPEN c;
(13)	FETCH c INTO name;
(14)	WHILE c%FOUND LOOP
(15)	<pre>htp.listItem(name);</pre>
(16)	FETCH c INTO name;
(17)	END LOOP;
(18)	CLOSE c;

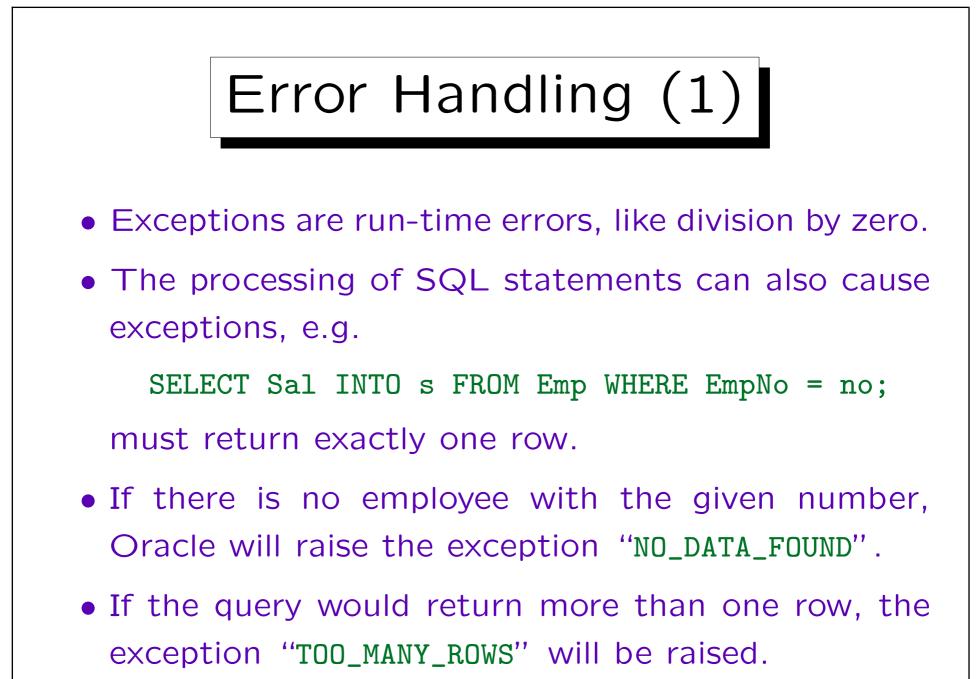


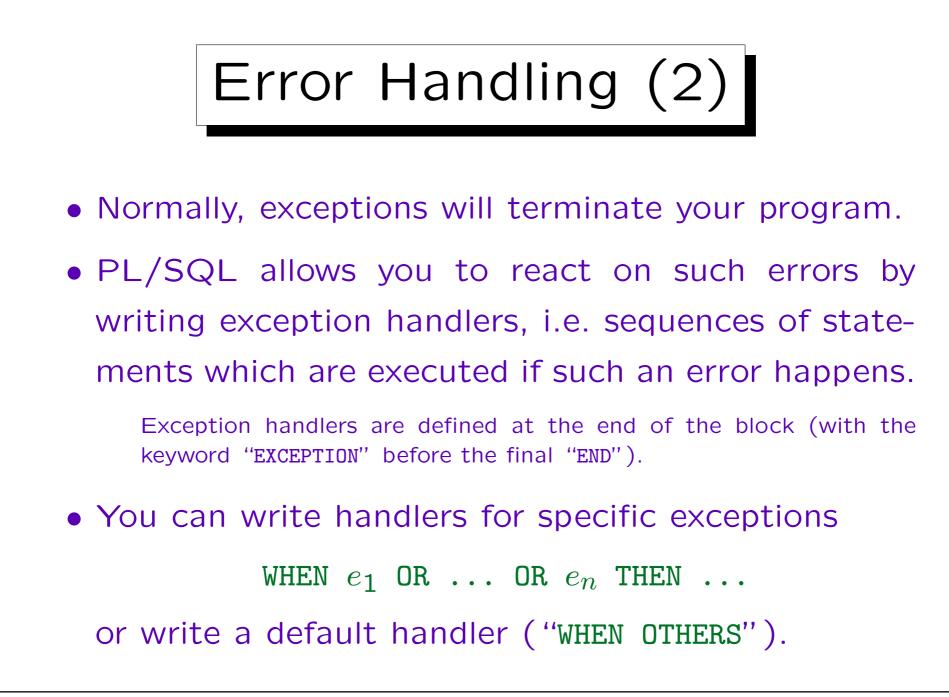


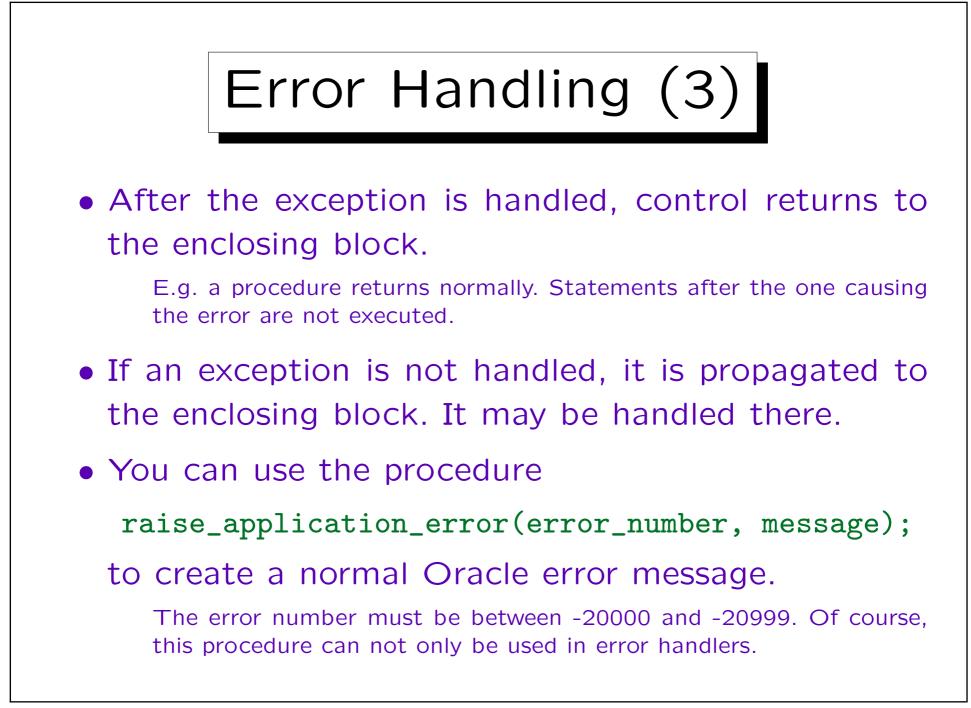
- For updates in loops, one can refer to "CURRENT OF (Cursor)" in WHERE-conditions.
- This requires a "SELECT ... FOR UPDATE" query.
- Example: Suppose we want to assign a unique number to every instructor. We have used ALTER TABLE to add a column No to the table Instructor, but it first contains null values.
- The following program also uses PL/SQL's recordtypes (which allow to store entire rows in a single variable).

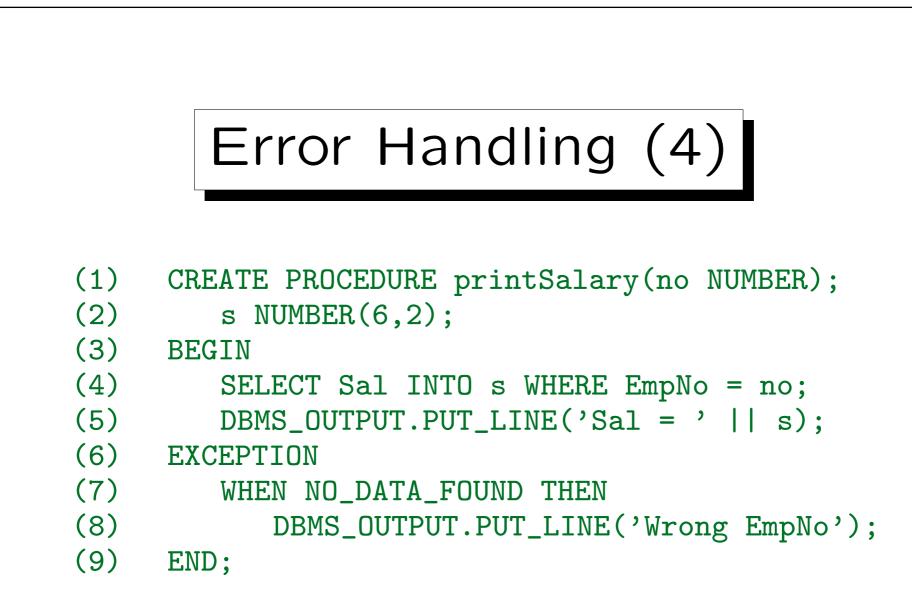
```
Cursors (7)
(1) CREATE PROCEDURE Number_Instructors IS
(2)
         inst Instructor%ROWTYPE;
(3)
         i NUMBER(4) := 1;
(4)
         CURSOR c IS SELECT * FROM Instructor
(5)
                     FOR UPDATE;
(6)
      BEGIN
(7)
         FOR inst IN c LOOP
(8)
            UPDATE Instructor SET no = i
(9)
               WHERE CURRENT OF c;
(10)
        i := i + 1;
(11)
         END LOOP;
(12) END;
```





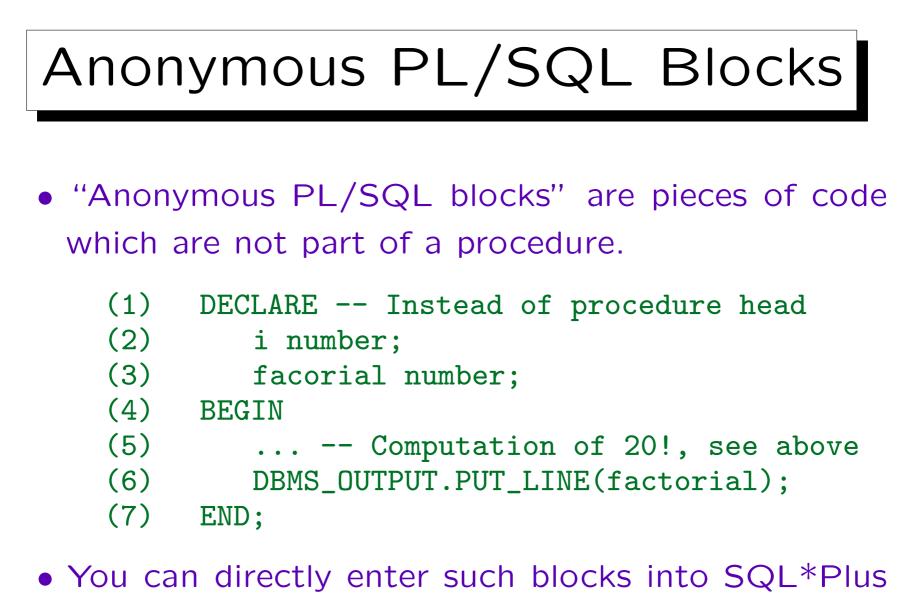


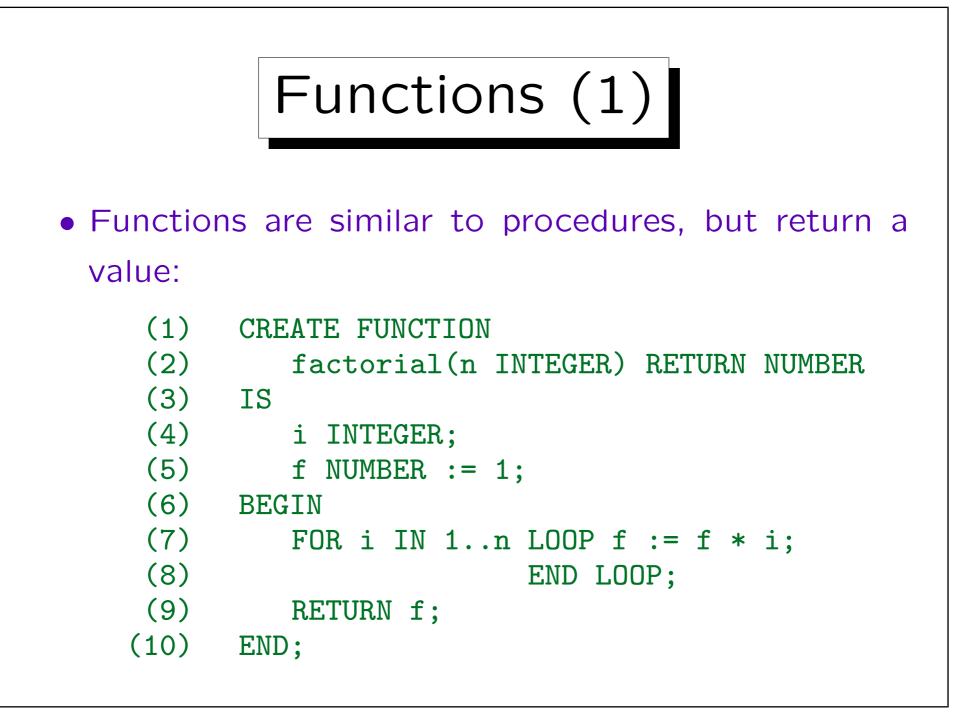




Error Handling (5)

- You can declare your own exceptions: You define them like a variable with type "EXCEPTION".
- Then you use the statement "RAISE $\langle \texttt{Exception} \rangle$ " to cause this exception.
- This allows, e.g., to have non-local jumps if you have many nested procedures and want to get back to some outer level.







• Functions can be used in PL/SQL expressions, e.g.

x := factorial(20) / 1000;

- Functions can also be used in SQL queries (!): SELECT n, factorial(n) FROM test_inputs
- Functions are not allowed to have side effects.
- Functions must execute a RETURN statement, or the exception "PROGRAM_ERROR" is raised.
- RETURN; (without value) can be used in procedures to transfer the control back to the caller.

Packages (1)

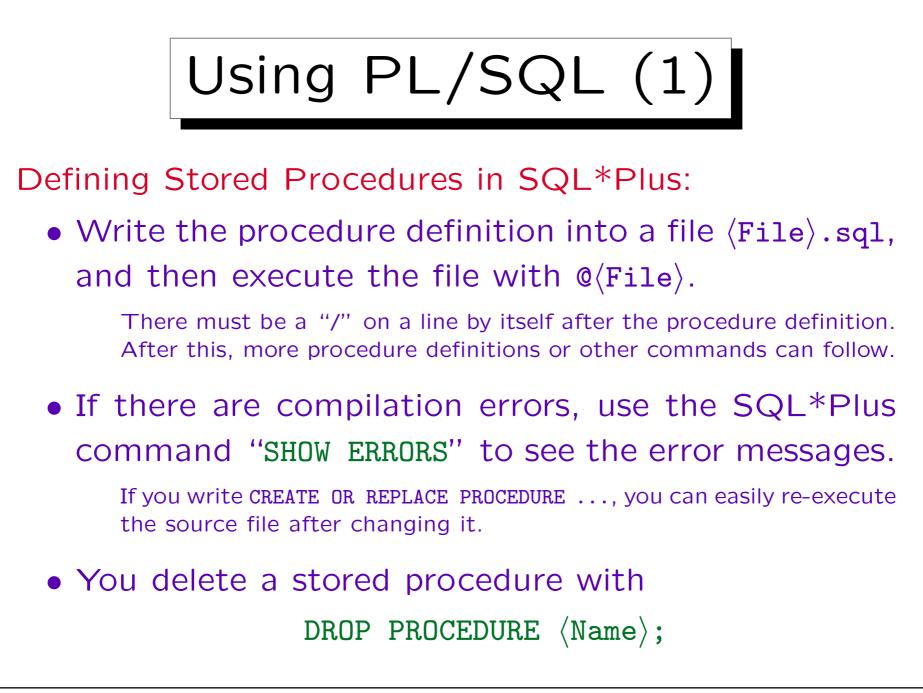
- PL/SQL has a module mechanism.
- You can define "Packages" which can contain declarations of types, procedures, functions, global variables, constants, cursors, and exceptions.
- The interface (public part) and the implementation (private part) are defined separately.
- Global variables persist for the duration of a session. Across transactions. But remote calls are restricted.
- When you refer to a packaged procedure etc. from outside a package, you must write "(Package).(Name)".

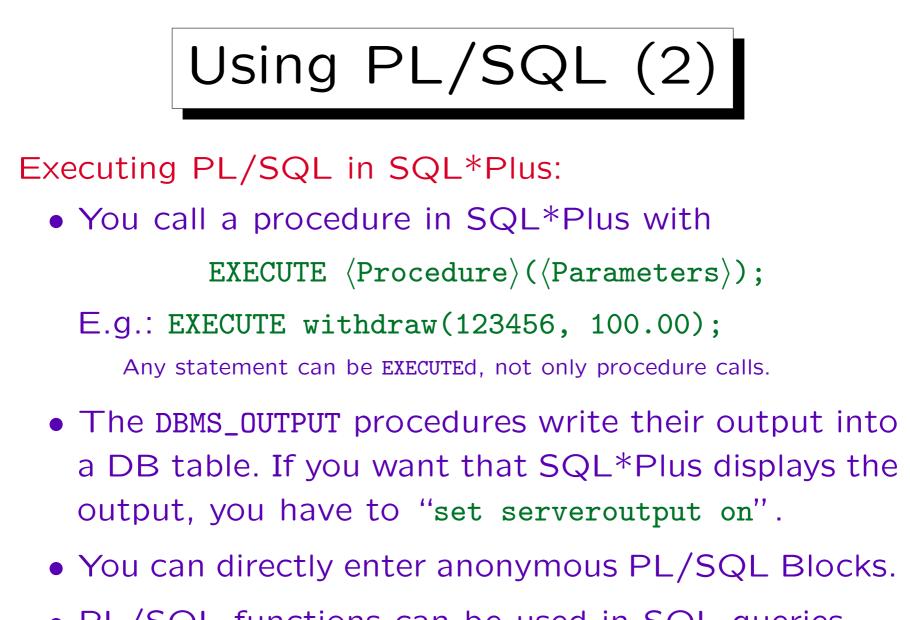
```
Packages (2)
(1) CREATE PACKAGE Com_Fun AS
(2)
        FUNCTION fac(n INTEGER) RETURN NUMBER;
(3)
        FUNCTION bino(n INTEGER) RETURN NUMBER;
(4) END Com_Fun;
(5) CREATE PACKAGE BODY Com_Fun AS
(6)
        FUNCTION fac(n INTEGER) RETURN NUMBER IS
(7)
           i INTEGER; f NUMBER;
(8)
       BEGIN
```

```
(10)
         END;
(11)
```

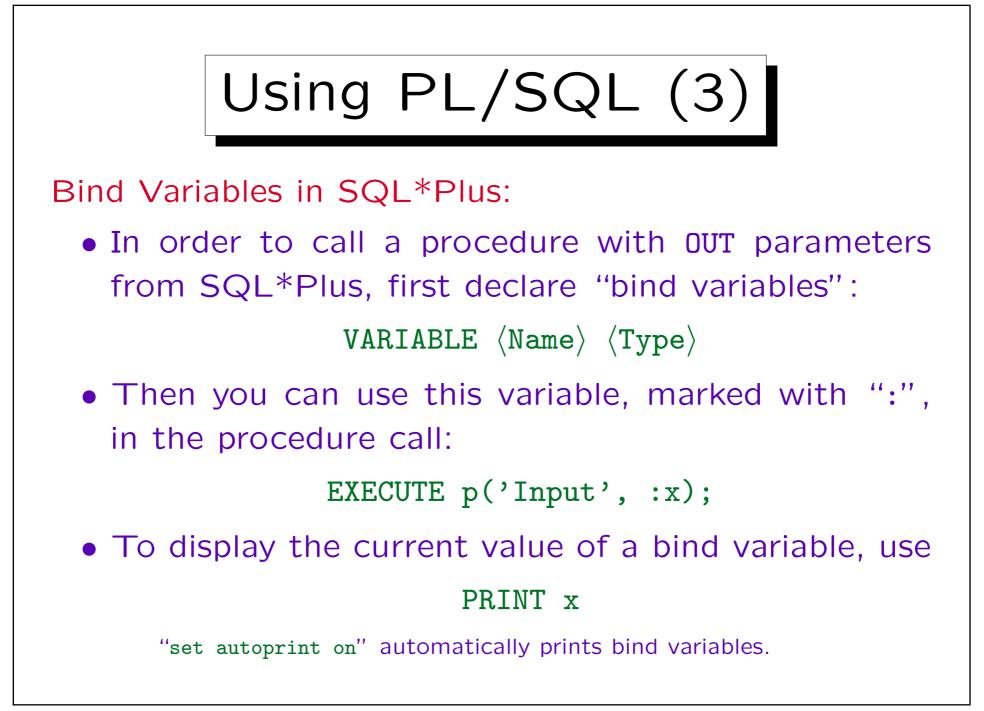
(9)

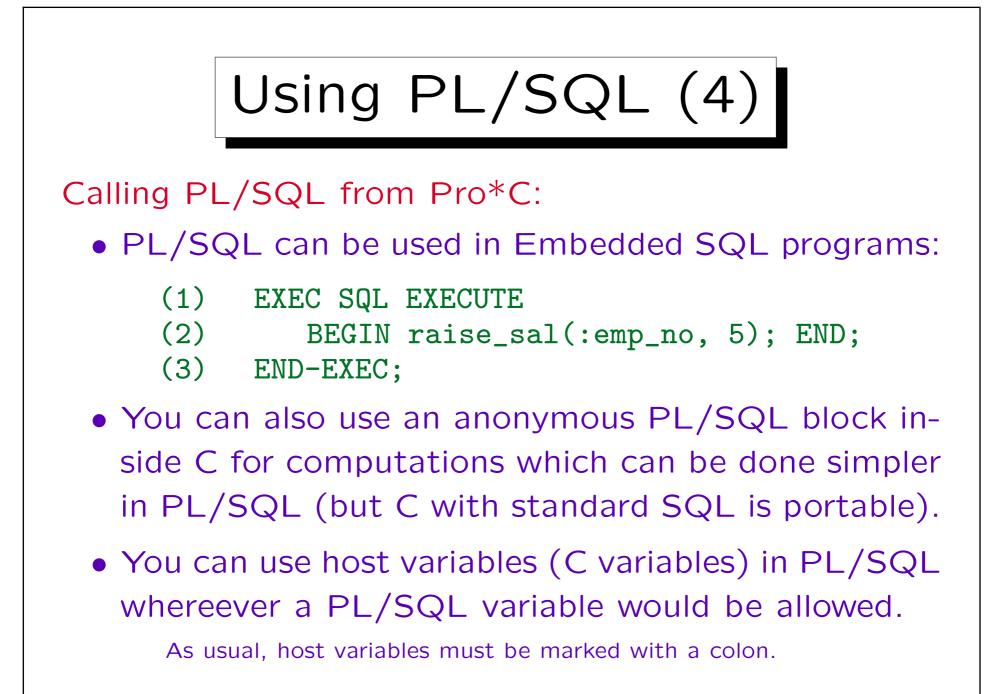
```
(12) END Com_Fun;
```

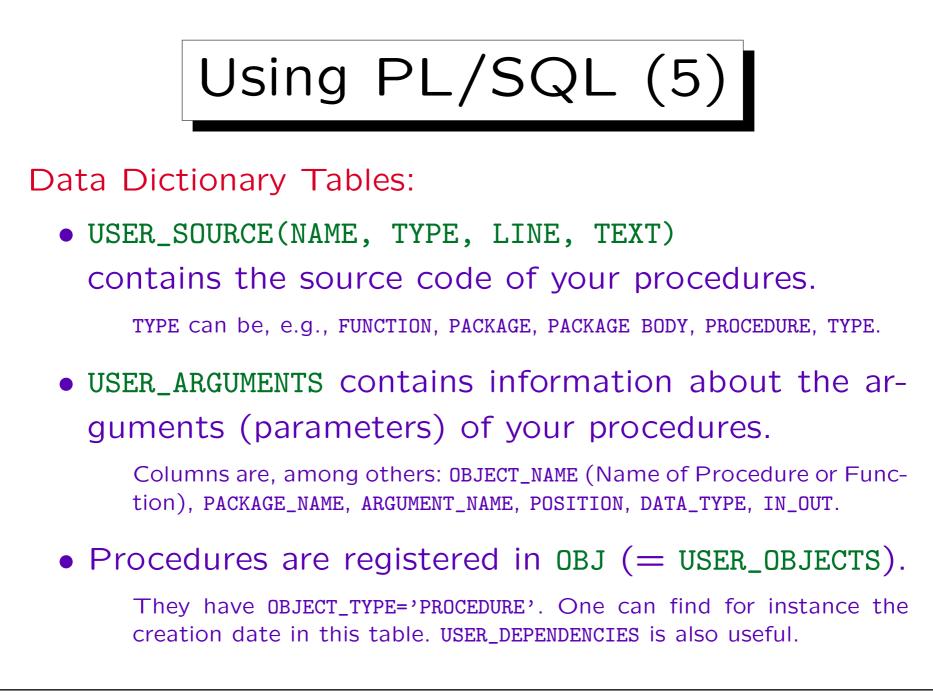




• PL/SQL functions can be used in SQL queries.





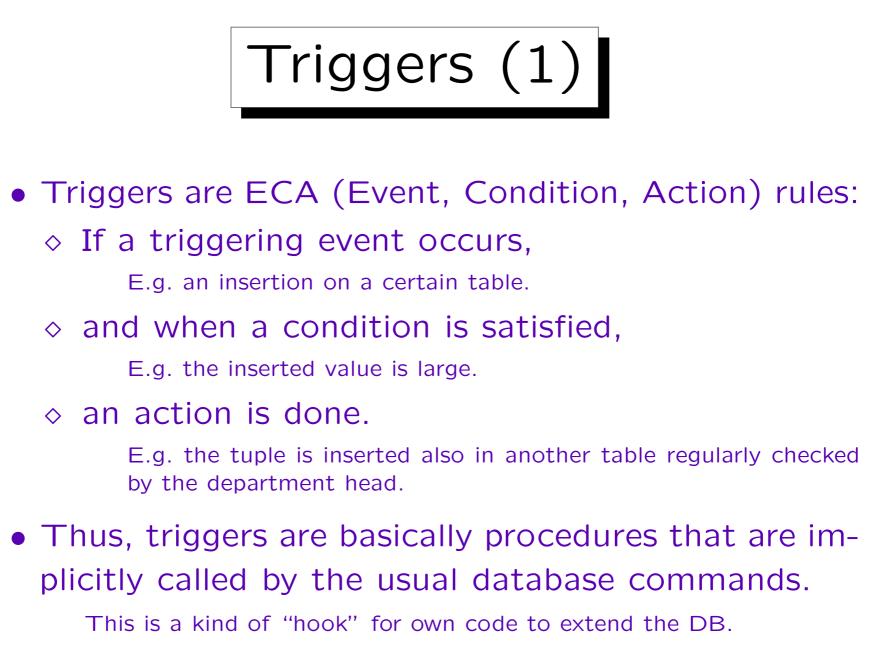


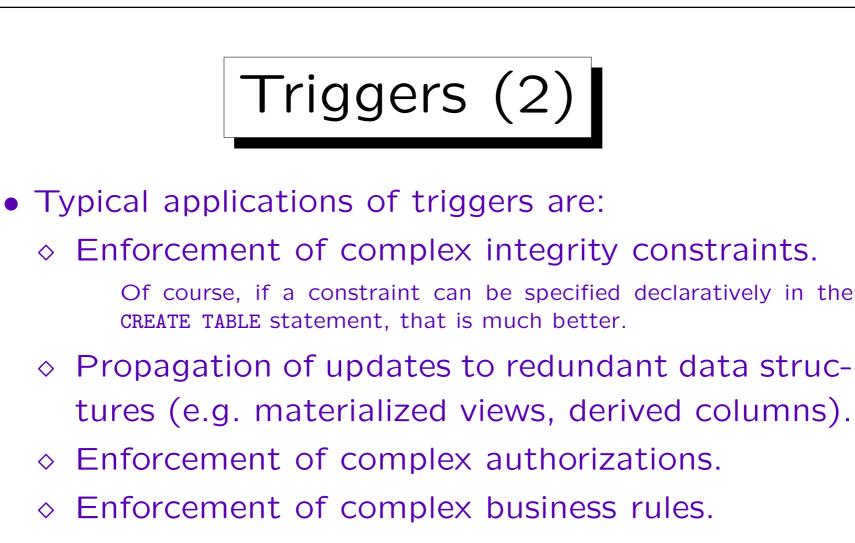


1. PL/SQL

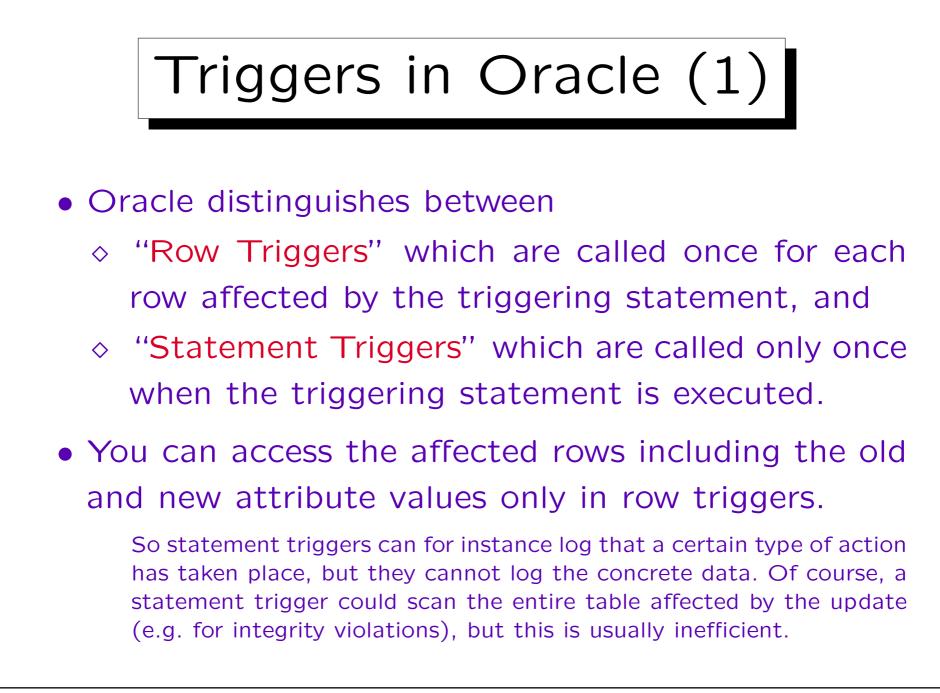
2. Triggers in Oracle

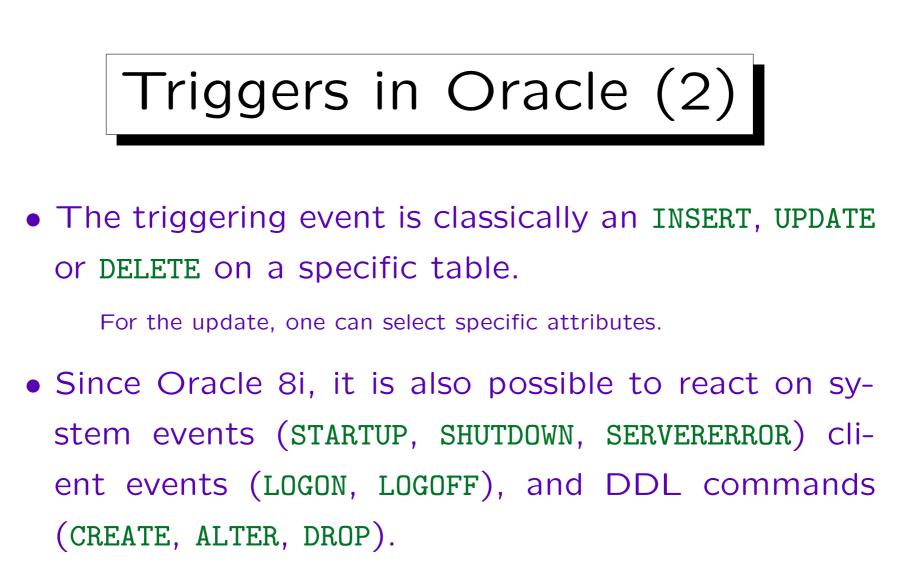
Stefan Brass: Datenbanken I



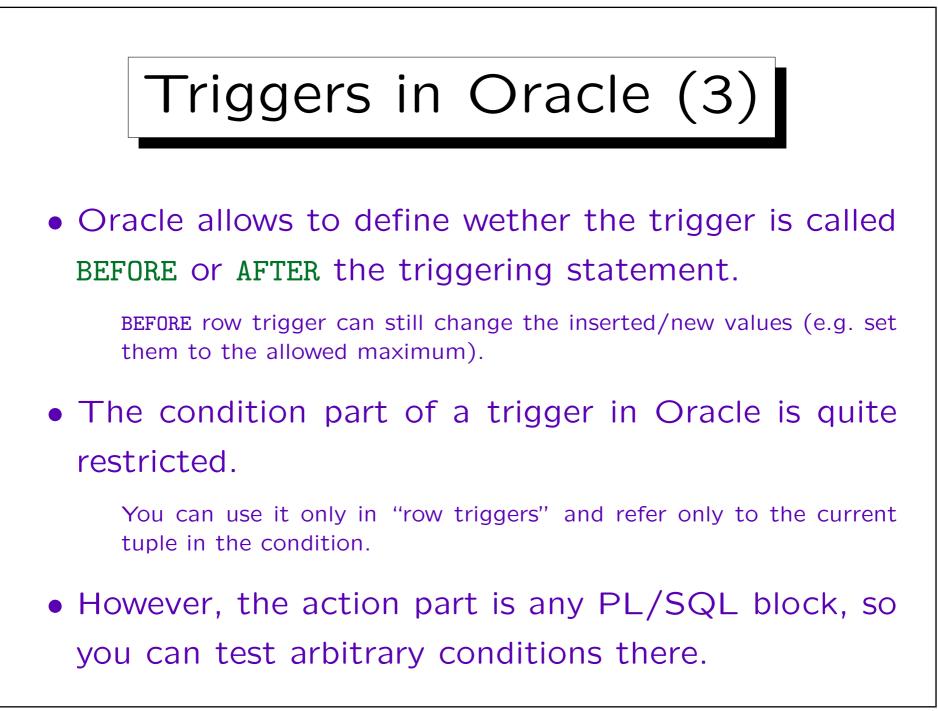


• Oracle also has "INSTEAD OF" trigger to define what should happen if a user tries to updates a view that would normally not be updatable.





There are "event attribute functions" to access more data for these events.



Example (1)

• Suppose we have a table

Inventory				
ItemNo	ItemName	Stock	MinStock	Reordered
•	•			:

- Our application programs change this table by UPDATE Inventory SET Stock = Stock - :taken WHERE ItemNo = :no
- If the Stock becomes smaller than MinStock, we want to automatically reorder the part.

Example (2)



Syntax (1)

Event:

- The triggering event is specified by
 - (2) AFTER UPDATE OF Stock ON Inventory
 - (3) FOR EACH ROW
- So this trigger will be executed after a command UPDATE Inventory SET Stock = ...
- It is executed once for each changed row.
- Triggers can fire on different events (but each trigger is only for one table):
 - (2) AFTER UPDATE OF Stock, MinStock OR INSERT
 - (3) ON Inventory



Condition:

- When the triggering event occurs, the condition is checked:
 - (4) WHEN (new.Stock < new.MinStock
 - (5) AND new.Reordered IS NULL)
- The condition is any WHERE condition without subqueries.
- The tuple variable "new" refers to the tuple after the update or insertion, "old" is the version before the update/deletion.



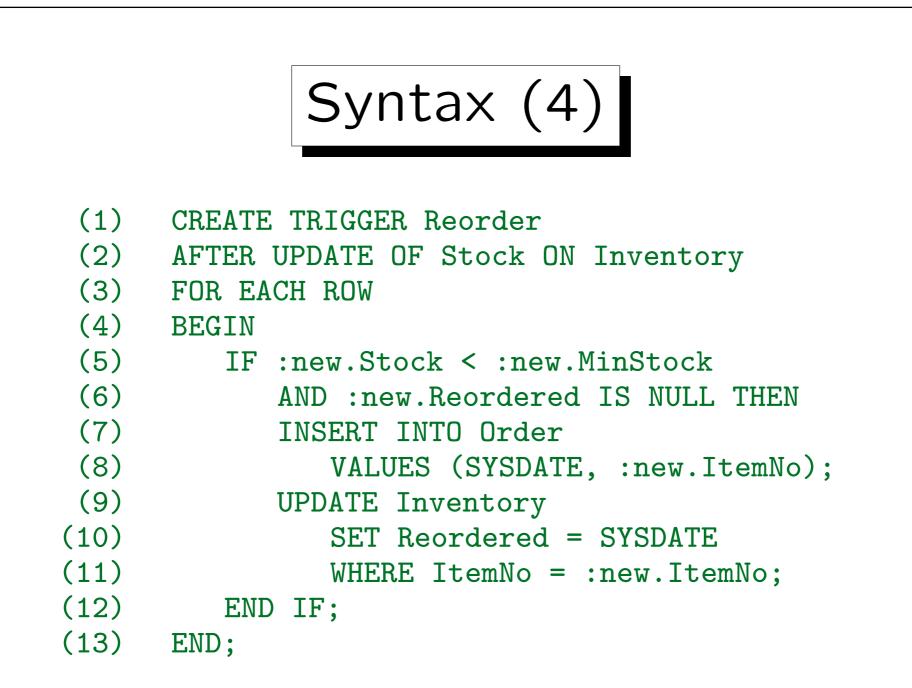
Condition, continued:

 If there is a name conflict with the tuple variables new and old, one can use

REFERENCING new AS myNew

This must be written before "FOR EACH ROW".

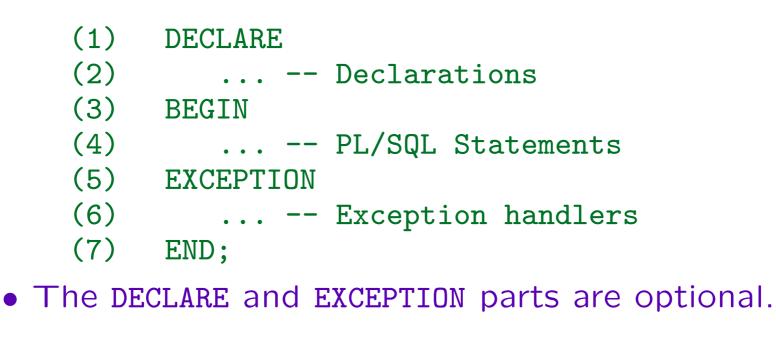
- The condition can alternatively be tested in the action (see example on next slide).
- This is probably slightly less efficient, but allows more general conditions to be tested.
- "WHEN" conditions can only be used in row triggers.





Action:

- The action part is any PL/SQL block.
- A PL/SQL block has the general form



18 - 64



Action, continued:

- COMMIT or ROLLBACK is not allowed in the PL/SQL block (trigger action).
- However, one can call the procedure raise_application_error(error_number, message) to abort the SQL command which fired the trigger. One can select any error number between -20000 and -20999.
- The conditions (predicates)
 INSERTING, DELETING, UPDATING, UPDATING(A)
 can be used to react on the type of update.



Action, continued:

- In row triggers, you can use the variables :new and :old, which contain the tuple before and after the command.
 - You cannot access the affected tuples in statement triggers. Note that in the WHEN condition, old and new are written without ":".
- In row triggers, you are not allowed to access other tuples (besides the changed one) from the updated table.

Such triggers are executed in the middle of changing this table, so the state of the table is undefined.

```
Second Example
(1)
      CREATE TRIGGER Course_Change_Restriction
(2)
      -- Changes are allowed only on working days.
(3)
      BEFORE INSERT OR UPDATE OR DELETE ON Course
(4)
      DECLARE weekday VARCHAR(3);
(5)
      BEGIN
(6)
         weekday := TO_CHAR(SYSDATE, 'DY');
(7)
         IF weekday='SAT' OR weekday='SUN' THEN
(8)
            raise_application_error(-20000,
(9)
               'No changes allowed today.');
(10)
         END IF;
(11)
      END;
```

