Introduction to PL/SQL

Kristian Torp

Department of Computer Science
Aalborg University
www.cs.aau.dk/˜torp
torp@cs.aau.dk

December 2, 2011

daisy
Center for Data-intensive Systems

daisy.aau.dk

Outline

1. Introduction

2. Stored Procedures
   - An Overview
   - Data Types
   - Parameter Parsing
   - Cursors

3. Packages
   - Case Study: Table API

4. Summary

Outline

1. Introduction

2. Stored Procedures
   - An Overview
   - Data Types
   - Parameter Parsing
   - Cursors

3. Packages
   - Case Study: Table API

4. Summary

Learning Outcomes

Learning Outcomes

- Understand how code can be executed within a DBMS
- Be able to design stored procedures in general
- Be able to construct and execute stored procedures on Oracle
- Knowledge about the pros and cons of stored procedures

Note That

- Concepts are *fairly* DBMS independent
- All code examples are Oracle specific

Prerequisites

SQL

- Knowledge about the SQL select statement
- Knowledge about SQL modification statements, e.g., insert and delete
- Knowledge about transaction management, e.g., commit and rollback
- Knowledge about tables, views, and integrity constraints

Procedural Programming Language

- Knowledge about another programming language of the C family
- Knowledge about data types, e.g., int, long, string, and Boolean
- Knowledge of control structures, e.g., if, while, for, and case
- Knowledge of functions, procedures, parameters, and return values

Motivation

Purpose

Move processing into the DBMS from client programs (database applications)

Advantages

- Code accessible to all applications
  - Access from different programming languages
- Very efficient for data intensive processing
  - Process large data set, small result returned
- Enhance the security of database applications
  - Avoid SQL injection attacks `http://en.wikipedia.org/wiki/SQL_injection`

Motivation

Purpose

Move processing into the DBMS from client programs (database applications)

Advantages

- Code accessible to all applications
  - Access from different programming languages
- Very efficient for data intensive processing
  - Process large data set, small result returned
- Enhance the security of database applications
  - Avoid SQL injection attacks `http://en.wikipedia.org/wiki/SQL_injection`

Missing Standard

Unfortunately, the major DBMS vendors each have their own SQL dialect

Overview

Functionality

- SQL extended with control structures
  - Control structures like if and loop statements
- Used for
  - Stored procedures (and functions)
  - Package (Oracle specific)
  - Triggers
  - Types a.k.a. classes (Oracle specific)
- In very widely used in the industry
  - see `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`
- In the SQL standard called SQL/PSM
  - PSM = Persistent Storage Model

Focus

The focus is here on stored procedures and packages!

Outline

1 Introduction

2 Stored Procedures
- An Overview
- Data Types
- Parameter Parsing
- Cursors

3 Packages
- Case Study: Table API

4 Summary

Motivation for Stored Procedures

The Big Four Benefits

- Abstraction
  - Increases readability
- Implementation hiding
  - Can change internals without effecting clients
- Modular programs
  - More manageable and easier to understand
- Library
  - Reuse, reuse, and reuse!

Note

This is not different from any other procedural programming language!

Outline

1. Introduction

2. Stored Procedures
   - An Overview
   - Data Types
   - Parameter Parsing
   - Cursors

3. Packages
   - Case Study: Table API

4. Summary

A Procedure: Hello, World!

Example (The Start Program)

```
create or replace procedure hello_world is
begin
    dbms_output.put_line('Hello, World!');
end;
```

Note

- It is a procedure, i.e., not a function
  - Both a procedure and a function is called a stored procedure
- It is a begin and end language, not curly brackets: { and }
- It uses a built-in library dbms_output.put_line
  - The package dbms_output has the procedure put_line
  - Uses the dot notation for invoking functions myPackage.myProcedure

A Function: Calculating Your BMI

Example (A BMI Function)

```
create or replace function bmi(height int, weight float)
return float is
begin
    if height <= 0.3 or height > 3.0 then
        dbms_output.put_line('height must be in [0.3, 3.0] meters');
    end if;
    if weight <= 0 then
        dbms_output.put_line('weight must be positive');
    elsif weight > 500 then
        dbms_output.put_line('No human''s weight is 500 kg');
    end if;
    return weight/height**2;
end;
```

Note

- It takes two parameters height and weight
- It is a function, i.e., has a return statement
- It is strongly typed language, i.e., parameters and the return value

Executing Stored Procedures

Example (Execute on Oracle server)

```
-- to enable output from the server
SQL>set serveroutput on
```

Executing Stored Procedures

Example (Execute on Oracle server)

```
-- to enable output from the server
SQL>set serveroutput on

-- execute the procedure
SQL>execute hello_world;
```

Executing Stored Procedures

Example (Execute on Oracle server)

```
-- to enable output from the server
SQL>set serveroutput on

-- execute the procedure
SQL>execute hello_world;

-- execute the function
SQL>exec bmi(1.87, 90);
-- results in an error, value returned by function must be used!
```

Executing Stored Procedures

Example (Execute on Oracle server)

```
-- to enable output from the server
SQL>set serveroutput on
-- execute the procedure
SQL>execute hello_world;
-- execute the function
SQL>exec bmi(1.87, 90);
-- results in an error, value returned by function must be used!
-- Wrap the function call
SQL>exec dbms_output.put_line(bmi(1.87, 90));
```

Executing Stored Procedures

Example (Execute on Oracle server)

```
-- to enable output from the server
SQL>set serveroutput on

-- execute the procedure
SQL>execute hello_world;

-- execute the function
SQL>exec bmi(1.87, 90);
-- results in an error, value returned by function must be used!

-- Wrap the function call
SQL>exec dbms_output.put_line(bmi(1.87, 90));
```

Note

- Output from server is *not* enabled by default in a session!
- Return value of a function *cannot* be ignored!

Using SQL in Stored Procedures

Example (Use the Data Stored)

```
create or replace function get_status(student_id number)
return varchar2 is
    v_status varchar2(50);
begin
    select sta.dsc
    into   v_status
    from   student stu, status sta
    where  stu.stat_id = sta.stat_id
    and    stu.sid = student_id;
    return v_status;
end;
```

Note

- The declaration of the variable v_status
- The usage of the into keyword in the select statement
- The usage of the parameter student_id in the select statement

Calling Other Procedures

Example (Callee)

```
create or replace procedure p (st varchar2) as
begin
    dbms_output.put_line(st);
end;
```

Calling Other Procedures

Example (Callee)

```
create or replace procedure p (st varchar2) as
begin
    dbms_output.put_line(st);
end;
```

Example (Caller)

```
create or replace procedure call_p is
begin
    p('Hello'); p('World!');
end;
```

Calling Other Procedures

Example (Callee)

```
create or replace procedure p (st varchar2) as
begin
    dbms_output.put_line(st);
end;
```

Example (Caller)

```
create or replace procedure call_p is
begin
    p('Hello'); p('World!');
end;
```

Note

- Can call own and built-in stored procedures
- Will use the procedure p instead of dbms_output.put_line
- You are now officially a PL/SQL library builder!!!

Control Structures: A Crash Course I

Example (The If Statement)

```
create or replace procedure pb(val boolean) is
begin
    if val = true then
        dbms_output.put_line('true');
    elsif val = false then
        dbms_output.put_line('false');
    else
        dbms_output.put_line('null');
    end if;
end;
```

Note

- Is this stupid?
- Recall three-valued logic the root of all evil!
- We will use the procedure pb in the code that follows!

Control Structures: A Crash Course II

Example (The While Statement)

```
create or replace procedure count_10 is
   i int:= 1;
begin
    while i < 10 loop
        dbms_output.put_line(i);
        i := i + 1;
    end loop;
```

Note

- What is printed 1 to 9 or 1 to 10?
- PL/SQL also has a for statement, very different from C
- PL/SQL does *not* have increment/decrement operators, e.g., i-- or ++j
- PL/SQL does *not* have compound assignments, e.g., i+=7 or j*=2

Surprises: In General

Surprises

- The code is stored in the DBMS!
- { has been replaced by begin and } by end
- SQL and programming logic blends very nicely!
  - A strong-point of PL/SQL
- Procedures are different from functions
- The assignment operator is := and not =
- The comparison operator is = and not ==
- Control structures are quite different from the C world
- Three-valued logic will time and again surprise you!
- Server output is not enabled by default
  - Which is a big surprise

Outline

1. Introduction

2. Stored Procedures
   - An Overview
   - Data Types
   - Parameter Parsing
   - Cursors

3. Packages
   - Case Study: Table API

4. Summary

An Example

Example (Various Data Types)

```
create or replace procedure use_basic_types is
    v_str varchar2(30) := 'A string';
    v_int int := 2**65;
    c_pi  constant float := 3.14159265358979323846264338327950288419;
    v_float float := v_int * c_pi;
begin
    p(v_str); p(v_int); p(v_float);
end;
```

Output

```
A string
36893488147419103232
115904311329233965478,149216911761758199
```

Note

- Forget what you think of data types and size!
- Very high precision on all number types in both SQL and PL/SQL
- The size of strings must be defined

Overview: Scalar Data Types

Scalar Data Types

Description	Type	Examples
Integers	smallint	-100, 0, 100
	int/integer	-1000, 0, 1000
	positive	0, 1, 2, 3
Floats	number	10.3
	dec/decimal	123.456, 3.4
	real	123456.7890
Strings	varchar2	Hello, Theta-Join
	nvarchar2	Tøger, Dæmon
	char	World, Noise
Boolean	Boolean	True, False
Date/time	date	2007-09-09
	timestamp	2009-09-09 12:34:56

Note

- Not all of these data types are available from within SQL!

Quiz: The Decimal Data Type

Example (Rouding)

```
create or replace procedure using_decimal is
    v_dec decimal(4,2);
begin
    v_dec := 12.34;
    dbms_output.put_line(v_dec);
    v_dec := 12.344;
    dbms_output.put_line(v_dec);
    v_dec := 12.347;
    dbms_output.put_line(v_dec);
end;
```

Questions

- Will this compile, note that it is decimal(4,2)?
- What will be printed (if it compiles)?
- Are you surprised?

Overview: Other Data Types

Special Data Types

Description	Type
Composite	Record
	Varray
	Table
Large Objects	BLOB
	CLOB
	BFILE
Reference Types	REF
	REF CURSOR

Note

- We will only use records in this lecture.

Anchored Data Types: Type

Example (Anchor for a Column)

```
create or replace function get_status_anchored(
    student_id student.sid%type)
return status.dsc%type is
    v_status status.dsc%type;
begin
    select sta.dsc
    into   v_status
    from   student stu, status sta
    where  stu.stat_id = sta.stat_id
    and    stu.sid = student_id;
    return v_status;
end;
```

Note

- The anchored type using the %type
- Very convenient of maintenance reasons (avoid “hard-wiring” types)
  - Widely used, you are encouraged to use it!

Anchored Data Types: Rowtype

Example (Anchor for a Table)

```
create or replace procedure get_course_rowtype(
   course_id course.cid%type)
is
   v_row course%rowtype;
   v_tmp varchar2(500);
begin
   select *
   into   v_row
   from   course c
   where  c.cid = course_id;
   v_tmp := v_row.cname || ': ' || v_row.dsc;
   p(v_tmp);
end;
```

Note

- The anchored type using the rowtype
  - Creates a record
- The dot notation for access elements of the record

Surprises: Data Type

Note

- Strings are a basic type, not an object like in Java or C#
  - A maximum size must be specified
- The sizes of the basic data type are very different from C and Java
- Date and time are basic data types!
  - This is very handy
- The anchored types is something new compared to C and Java
- Booleans are not a basic data type in SQL but in PL/SQL
  - This sometimes leads to very annoying problems
- Support for composite data type is not very good in PL/SQL compared to C and Java
- LOB objects are plain stupid
  - But sometimes necessary

Outline

1 Introduction

2 Stored Procedures

- An Overview
- Data Types
- Parameter Parsing
- Cursors

3 Packages

- Case Study: Table API

4 Summary

Overview

Example

```
create or replace procedure p_in(val in int) is
    v_tmp int;
begin
    v_tmp := val + 5;
    --val := val + 5; /*illegal val is read-only */
end;
```

Example

```
create or replace procedure p_in_out(val in out int) is
begin
    val := val + 5;
end;
```

Example

```
create or replace procedure call_ps is
    v_in     int := 10;
    v_in_out int := 10;
begin
    p_in(v_in); p(v_in);
    p_in_out(v_in_out); p(v_in_out);
end;
```

- When execute call_ps prints 10 and 15, why?

Quiz

Example

```
create or replace procedure p_in_out(val in out int) is
begin
    val := val + 5;
end;
```

Example

```
create or replace procedure call_ps is
    v_in     int := 10;
    v_in_out int := 10;
begin
    p_in(v_in); p(v_in);
    p_in_out(v_in_out); p(v_in_out);
end;
```

Questions

- What are the formal parameter(s)?
- What are the actual parameter(s)?
- Is it call-by-value or call-by-reference?

Out Parameters

Example

```
create or replace procedure get_x_y_coor(
    coor_id in int, x_coor out int, y_coor out int)
is
begin
    x_coor := round(coor_id/4.2);  -- stupid calculations
    y_coor := round(coor_id/7.5);
end;
```

Note

- in and out parameters can both be used in same procedure
- The out parameters are write-only
- More than one value is “returned” by the procedure
- The calculation is naturally plain stupid
- round is the built-in rounding function

Parameter Mode

Mode

Mode	Description
in	Formal parameter is read-only
out	Formal parameter is write-only
in out	Formal parameter is read-write

Note

- in is the default parameter mode if the mode is not specified
- Stored procedures cannot be overloaded on the parameter signature
- There is a nocopy compiler hint for in out parameters

What is Wrong Here?

```
create procedure proc_1(i int)
is
begin
   -- snip complicated stuff
   return i;
end;
```

A

```
create function func_1(i int)
return int is
begin
   -- snip complicated stuff
   return 'hello';
end;
```

B

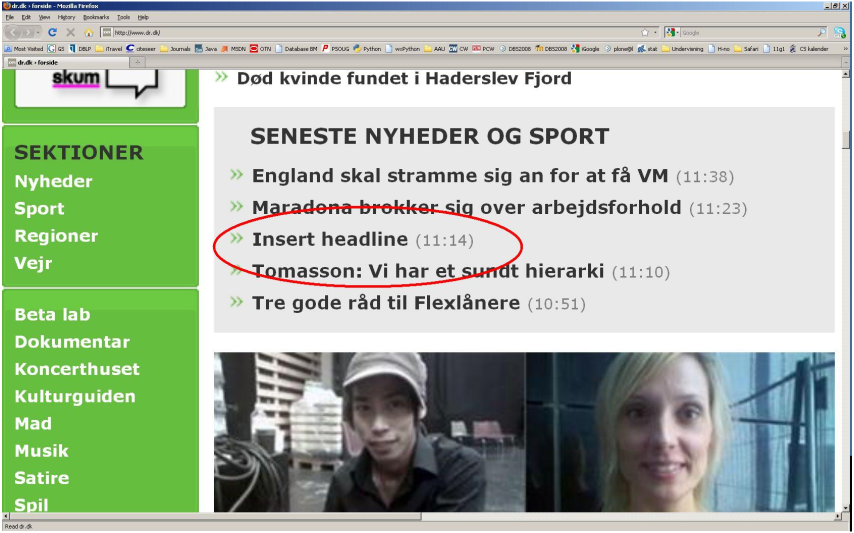
```
create function func_2(i int)
return int is
begin
   -- snip complicated stuff
   return i*2;
   p('hello world');
end;
```

C

```
create function func_3(i int)
return int is
begin
   -- snip complicated stuff
   p('hello world');
end;
```

D

Avoid This


dr.dk › forside - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.dr.dk/
Google
Most Visited GS DBLP iTravel citeseer Journals Java MSDN OTN Database BM PSOUG Python wxPython AAU CW PCW DBS2008 DBS2008 iGoogle plone@l stat Undervisning H-no Safari 11g1 CS kalender
dr.dk › forside
skum
Read dr.dk

» Død kvinde fundet i Haderslev Fjord

SEKTIONER
Nyheder
Sport
Regioner
Vejr

Beta lab
Dokumentar
Koncerthuset
Kulturguiden
Mad
Musik
Satire
Spil

SENESTE NYHEDER OG SPORT

» England skal stramme sig an for at få VM (11:38)
» Maradona brokker sig over arbejdsforhold (11:23)
» Insert headline (11:14)
» Tomasson: Vi har et sundt hierarki (11:10)
» Tre gode råd til Flexlånere (10:51)

Additional Comments on Parameters

Items to Notice

- A default value can be provided for each parameter
- Stored procedures cannot be overloaded on the parameter signature
- Stored procedures can be called by position or by name
- Works like in most programming languages, however different syntax!

Outline

1 Introduction

2 Stored Procedures
- An Overview
- Data Types
- Parameter Parsing
- Cursors

3 Packages
- Case Study: Table API

4 Summary

Overview

Definition

A cursor is a mechanism that ensure a result set can be identified by a declarative language such as SQL and processed by a procedural language such as PL/SQL or C#

Overview

Definition

A cursor is a mechanism that ensure a result set can be identified by a declarative language such as SQL and processed by a procedural language such as PL/SQL or C#

Solution

Solves the well-known impedance mismatch problem!

Overview

Definition

A cursor is a mechanism that ensure a result set can be identified by a declarative language such as SQL and processed by a procedural language such as PL/SQL or C#

Solution

Solves the well-known impedance mismatch problem!

Generality

Knowledge about cursors in PL/SQL is directly transferable to many other programming languages.

The Unix `ls` command

Example (List Tables)

```
create or replace procedure ls is
    cursor c_tables is
        select * from cat;
    v_table_name cat.table_name%type;
    v_type       cat.table_type%type;
begin
    open c_tables;
    loop
        fetch c_tables into v_table_name, v_type;
        exit when c_tables%notfound;
        p(v_table_name);
    end loop;
    close c_tables;
end;
```

Note

- The view tab is a table that contains all table names
- The cursor is declared, opened, fetched, and closed

Cursor Attributes

Attributes

Attribute	Type	Description
notfound	Boolean	True if a record is fetched unsuccessfully
found	Boolean	True if a record is fetched successfully
rowcount	Integer	The number of records fetched from the cursor
isopen	Boolean	True if cursor is open

Note

- There are additional attributes for bulk operations.

Quiz: Using rowcount

Example

```
create or replace procedure ls_cnt is
    cursor c_tables is
        select table_name from cat;
    v_table_name cat.table_name%type;
begin
    open c_tables;
    loop
        fetch c_tables into v_table_name;
        exit when c_tables%notfound;
        p(c_tables%rowcount || ' ' || v_table_name);
    end loop;
    close c_tables;
end;
```

Question

- What is printed?

Quiz: Using isopen?

Example

```
create or replace procedure ls_isopen is
    cursor c_tables is
        select table_name from cat;
    v_table_name cat.table_name%type;
    v_status boolean := false;
begin
    v_status := c_tables%isopen; pb(v_status);
    open c_tables;
    v_status := c_tables%isopen; pb(v_status);
    loop
        fetch c_tables into v_table_name;
        exit when c_tables%notfound;
    end loop;
    v_status := c_tables%isopen; pb(v_status);
    close c_tables;
    v_status := c_tables%isopen; pb(v_status);
end;
```

Question

- What is printed?

Outline

1 Introduction

2 Stored Procedures
- An Overview
- Data Types
- Parameter Parsing
- Cursors

3 Packages
- Case Study: Table API

4 Summary

Introduction

Idea

- A class like concept
- Very good for building libraries
  - A way to cluster related stored procedures
- Has a header and a body (think C-style languages)

Outline

1 Introduction

2 Stored Procedures
- An Overview
- Data Types
- Parameter Parsing
- Cursors

3 Packages
- Case Study: Table API

4 Summary

Introduction

Goal

To build a uniform way to address the data stored in table!

Methods

Name	Description
`exist(<primary key>)`	Return true if primary key exists
`to_string(<primary key>)`	Return string representation of row
`print(<primary key>)`	Convenient way to display a row

Introduction

Goal

To build a uniform way to address the data stored in table!

Methods

Name	**Description**
`exist(<primary key>)`	Return true if primary key exists
`to_string(<primary key>)`	Return string representation of row
`print(<primary key>)`	Convenient way to display a row

Note

- Many more methods can be envisioned
- Think object-relational mapping (ORM) tools

Header File: Course Table

Example (Header)

```
create or replace package ccourse is
  function exist(cid course.cid%type) return boolean;
  function to_string(cid course.cid%type) return string;
  procedure print(cid course.cid%type);
end;
```

Note

- The header lists all the public stored procedures
- The naming convention table name course package name `ccourse`
- The design is influenced by the `Object` class from Java and C#

Header File: Course Table

Example (Header)

```
create or replace package ccourse is
  function exist(cid course.cid%type) return boolean;
  function to_string(cid course.cid%type) return string;
  procedure print(cid course.cid%type);
end;
```

Note

- The header lists all the public stored procedures
- The naming convention table name course package name `ccourse`
- The design is influenced by the `Object` class from Java and C#

Quiz

- Why is the method called `exist` and not `exists`?

Body File: Private Method and Cursor

Example (Body)

```
create or replace package body ccourse is
    -- private constant
    c_error_cid_null constant int := -20001;
    -- a cursor used in the implementation
    cursor cur_exist(cv_cid course.cid%type) is
        select c.cid, c.cname, c.semester, c.dsc
        from   course c
        where  c.cid = cv_cid;
    -- a private method
    procedure check_valid_cid(cid course.cid%type) is
    begin
        if cid is null then
            raise_application_error(c_error_cid_null,
                                    'Course ID is null');
        end if;
    end;
```

Body File: Private Method and Cursor

Example (Body)

```
create or replace package body ccourse is
    -- private constant
    c_error_cid_null constant int := -20001;
    -- a cursor used in the implementation
    cursor cur_exist(cv_cid course.cid%type) is
        select c.cid, c.cname, c.semester, c.dsc
        from   course c
        where  c.cid = cv_cid;
    -- a private method
    procedure check_valid_cid(cid course.cid%type) is
    begin
        if cid is null then
            raise_application_error(c_error_cid_null,
                                    'Course ID is null');
        end if;
    end;
```

Note

- The method `check_valid_cid` is private

Body File: The Exist Method

Example (Method)

```
function exist(cid course.cid%type) return boolean is
    rec_exist cur_exist%rowtype;
begin
    check_valid_cid(cid); -- precondition
    open cur_exist(cid);
    fetch cur_exist into rec_exist;
    close cur_exist;
    return (rec_exist.cid is not null);
end;
```

Note

- Uses the private method `check_valid_cid` to check preconditions
- Uses the private cursor `cur_exist`
- Returns true if a valid primary key is found

Body File: The to_string Method

Example (Method)

```
function to_string(cid course.cid%type) return string is
    v_rv string(512);
    rec_exist cur_exist%rowtype;
begin
    check_valid_cid(cid); -- precondition
    open cur_exist(cid);
    fetch cur_exist into rec_exist;
    close cur_exist;
    v_rv := 'course name: ' || rec_exist.cname || ' ' ||
            'course desc: ' || rec_exist.dsc;
    return v_rv;
end;
```

Note

- Uses the private method `check_valid_cid` to check preconditions
- Uses the private cursor `cur_exist`

Body File: The print Methods

Example (Method)

```
procedure print(cid course.cid%type) is
begin
    check_valid_cid(cid); -- precondition
    dbms_output.put_line(to_string(cid));
end;
```

Note

- Uses the private method `check_valid_cid` to check preconditions
- `print` calls `to_string`

Exercising the Package

Example

```
SQL>set serveroutput on
-- execute the procedure
SQL>execute ccourse.print(4);
```

Exercising the Package

Example

```
SQL>set serveroutput on
-- execute the procedure
SQL>execute ccourse.print(4);
```

Note

- Similar to executing a stored procedure
- Access member by the dot notation

Exercising the Package

Example

```
SQL>set serveroutput on
-- execute the procedure
SQL>execute ccourse.print(4);
```

Note

- Similar to executing a stored procedure
- Access member by the dot notation

Example

```
SQL>execute ccourse.print(null);
```

Note

- Results in an error “ORA-20001: Course ID is null”

Summary: Packages

Main Points

- Can have a public and a private part
  - Has no protected access modifiers as in Java or C#
- Is used to cluster related stored procedures
- Cursors, constants, and variables can be shared between methods in a package
- The foundation for building larger libraries in PL/SQL
- There is a huge library of built-it packages on Oracle
- Has very good exception handling facilities

Comparison to Object-Oriented Languages

- No inheritance
- Only static methods
- No concept of an object

Outline

1 Introduction

2 Stored Procedures
- An Overview
- Data Types
- Parameter Parsing
- Cursors

3 Packages
- Case Study: Table API

4 Summary

Advantages

Advantages

- A complete programming language
  - You are not missing stuff as you sometimes are in SQL
- In wide-spread usage in the industry
  - Adds to your market value
- Very good integration of programming logic and SQL
- Impedance mismatch is basically removed
  - PL/SQL data types are super set of SQL data types
  - Cursors enable the processing of sets (or bags)

Disadvantages

Disadvantages

- Proprietary programming language
- There is a very large number (>1000) of reserved words
  - Can be hard to come up with a variable name that is not a reserved word!
- Pascal-family language (C-family more well-known)
  - Which lead to a number of surprises
- Object-oriented features are “clumsy”
  - This has not been covered in this lecture

Additional Information

Web Sites

- `www.oracle.com/technology/tech/pl_sql/index.html` PL/SQL's home
- `www.psoug.org/library.html` A very good and complete wiki with PL/SQL information
- `plsql-tutorial.com/` A crash course covering many PL/SQL features
- `en.wikibooks.org/wiki/Oracle_Programming/SQL_Cheatsheet` A short overview of PL/SQL
- `www.java2s.com/Tutorial/Oracle/CatalogOracle.htm` Many good examples, too many commercials