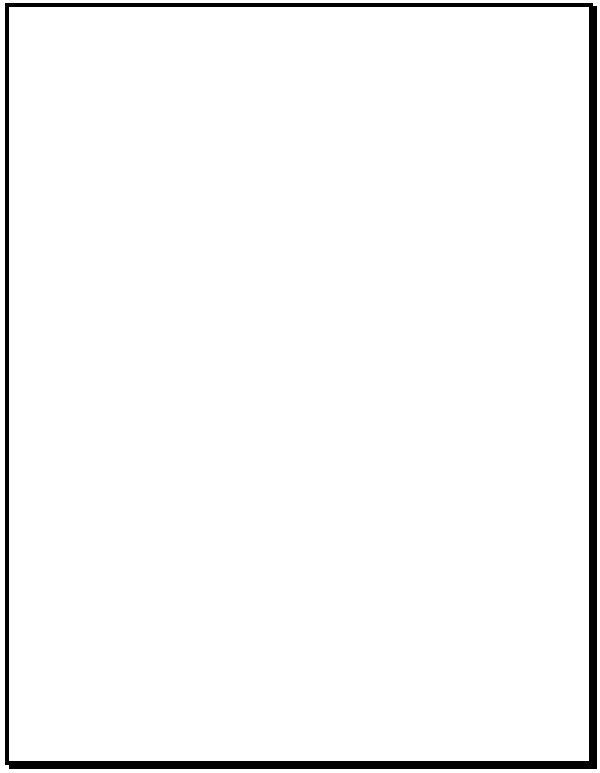
Interbase 4.0 to Oracle7

Stored Procedures Conversion Guide
Developer's Release

Oracle Worldwide Alliances
Design and Migration Services



ORACLE®

DRAFT — 29/May/1997

STORED PROCEDURES CONVERSION GUIDE

INTERBASE 4.0 TO ORACLE7

VERSION 1.0
JUNE 1997

Part Number CXXXXXX

Enabling the Information Age™

Stored Procedures Conversion Guide: Interbase 4.0 to Oracle7
Version 1.0
June 1997

Part Number CXXXXX

Major Contributors: Barry McGillin

Contributors:

PREFACE

This guide explains how to convert stored procedures in a Interbase 4.0 application to PL/SQL procedures or packages in an Oracle7 application. It includes information about the *sp_converter* program, about Interbase 4.0 stored procedures and PL/SQL stored subprograms (procedures and packages), and reference information to aid you in completing the conversion of different types of stored procedures using the *sp_converter*.

Audience

The information in this manual is intended primarily for Oracle Value Added Resellers (VARs) and Oracle consultants. Readers should have a working knowledge of SQL, PL/SQL, SQL*Plus and Oracle concepts.

How This Guide Is Organized

Chapter 1: Overview
This chapter provides an overview of *sp_converter* that converts Interbase 4.0 stored procedures to Oracle7 stored subprograms.

Chapter 2: Performing the Conversion
This chapter describes the *sp_converter* files.

Chapter 3: T-SQL and PL/SQL Language Elements
This chapter compares the language elements used by Interbase 4.0 systems and the PL/SQL elements used by Oracle7.

Chapter 4: Using the PL/SQL Generated in PRO*C

This chapter is full of Interbase code and the Oracle7 equivelant. It also show how a pro*c program can acess a PL/SQL table of records.

Chapter 5: Datatypes
This chapter provides detailed descriptions of the differences in datatypes used by the Interbase 4.0 stored procedures and Oracle7 stored subprograms.

Chapter 6: Schema Objects
This chapter provides detailed descriptions of the schema objects: stored procedures in Interbase 4.0 and stored subprograms in Oracle7.

Chapter 7: D-SQL vs. PL/SQL Constructs
This chapter describes how Interbase 4.0 constructs are converted to Oracle7 constructs by *sp_converter*.

Appendix A: List of Conversion Steps
This appendix lists the steps for migrating stored procedures from Interbase 4.0 to Oracle7. It also helps you to estimate the time required to complete your conversion project.

Related Publications

Along with this guide, you may want to refer to the following documents published by Oracle Corporation:

- *Database Conversion Guide: Interbase 4.0 to Oracle7*, Part No. AXXXXX
- *ORACLE7 Server Concepts Manual*, Part No. 6693-70
- *ORACLE7 Server Administrator's Guide*, Part No. 6694-70
- *ORACLE7 Server Application Developer's Guide*, Part No. 6695-70
- *ORACLE7 Server Messages and Codes Manual*, Part No. 3605-70
- *ORACLE7 Server Parallel Server Administrator's Guide*, Part No. 5990-70
- *ORACLE7 Server SQL Language Reference Manual*, Version 7, Part No. 778-70
- *PL/SQL User's Guide and Reference*, Part No. 800-V1.0
- *Programmer's Guide to ORACLE Precompilers*, Part No. 5315-15
- *Pro*C Precompiler Supplement*, Part No. 5452-15
- *SQL*Net Administrator's Guide*, Part No. A11325-1

References

You may want to refer to Interbase reference materials for specific details on Interbase 4.0 functionality.

Conventions Used in This Guide

The following conventions are used in this guide:

UPPERCASE	Calls attention to command keywords, command names, table names, object names, and filenames. Enter text exactly as spelled; it need not be in uppercase.
lowercase, italics	A clause value; substitute an appropriate value.
`font change`	A different font distinguishes examples of commands and statements from the rest of the text.
bold words	Calls attention to important information.
|	Separates alternative syntax items that may be optional or mandatory. Do not enter the |.
[ ]	One or more optional items. Do not enter the brackets.
{ }	A choice of mandatory items; enter one of the items. Do not enter the braces.
…	Preceding item(s) may be repeated any number of times.

Your Comments Are Valued

We welcome and appreciate your comments. As we develop our software and documentation, your opinions are the most important input we receive. To contact us, send your e-mail to:

infockit@us.oracle.com

If you prefer, call us at (415) 506-0329 or write to:

Oracle Corporation
World Headquarters
Worldwide Alliances
Design & Migration Services
Box 659604
500 Oracle Parkway
Redwood Shores, California 94065
U.S.A.

CONTENTS

CHAPTER

1 OVERVIEW

This chapter provides an overview of the stored procedure converter (intspconv) that converts Interbase V4 stored procedures into Oracle7 stored subprograms (procedures or packages). Specifically, this chapter contains the following information:

- an introduction to the converter
- a graphic overview of the conversion process
- prerequisites for running the converter

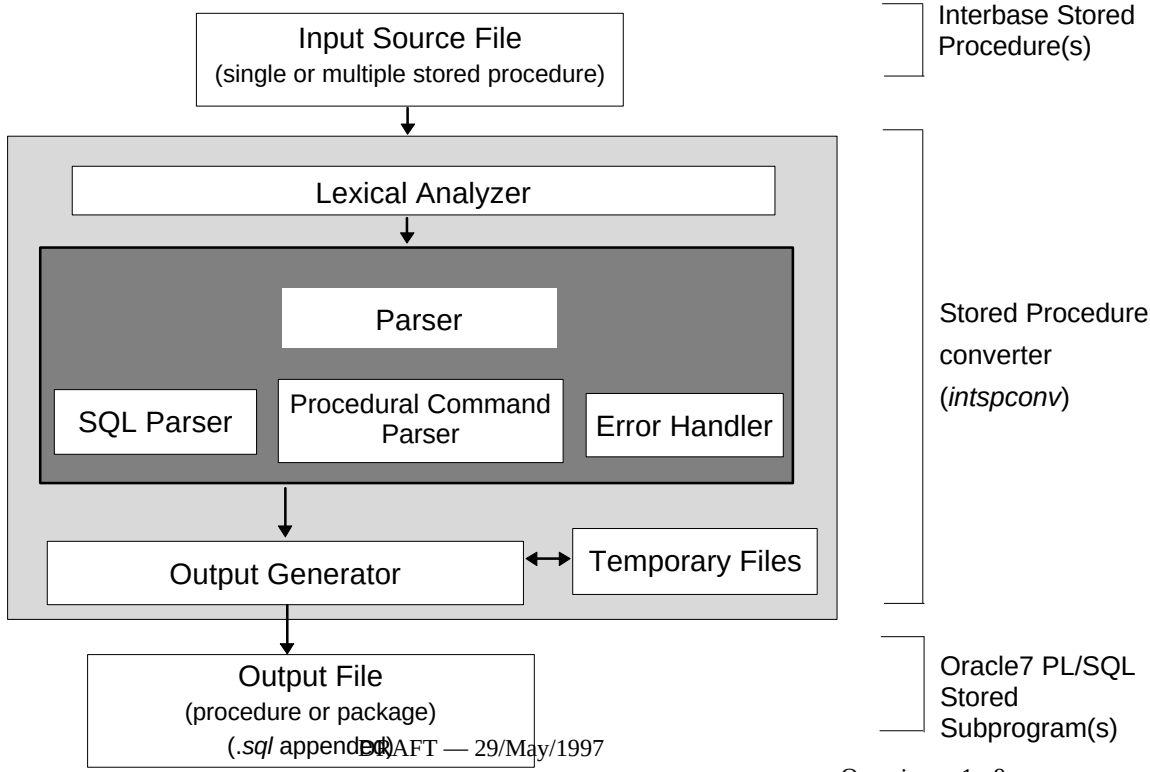
Introduction to the Converter

The Oracle7 stored procedure converter (*intspconv*) is a standalone utility that converts Interbase stored procedures to Oracle7 PL/SQL stored subprograms (procedures or packages).

The converter automates most of the conversion, but because of differences between Interbase Stored Procedures and Oracle PL/SQL, some manual conversion is required.

The input to the converter program is a flat file containing Interbase stored procedures and the output is a file containing an equivalent Oracle7 PL/SQL subprograms (procedure, or package). The converter does not require the Interbase or Oracle7 servers to be available during the conversion.

Graphic View of the Conversion Process



Prerequisites to Running the Converter

To run successfully, the Interbase Stored procedure Converter *intspconv* requires one of the following operating systems:

- UNIX
- WIN95
- WINNT

The stored procedure converter *intspconv* is a stand-alone tool. It will run in any of the above environments.

For the UNIX tool, the Operating System name and version must be given so that the correct build of the tool can be supplied.

CHAPTER

2 PERFORMING THE CONVERSION

This chapter provides details about converting Interbase stored procedures to Oracle7 Release 7.X stored subprograms (procedures or packages). This chapter contains the following information:

- description of the *sp_converter* files
- how to run the converter
- *sp_converter* command line options
- converter capabilities
- manual conversion requirements

Converter Files

The *sp_converter* program is used to convert Interbase 4.0 stored procedures into Oracle stored subprograms (procedures or packages).

The *sp_converter* is comprised of the following files in the binary distribution:

File Name	Description
intspconv	Interbase 4.0 to Oracle PL/sql converter
README	Readme document with essential instructions for the tool
Changelog	This file lists the changes which have occured since the last release.
TODO	A list of features which will be implemented into the tool in the near future.

Running the Converter

From the *sp_converter* directory, enter:

```
intspconv [-fbc][hdspt] -i <input_filename> -o
<output_filename>
```

where <*input_filename*> is the Interbase 4.0 flat file containing a stored procedure, and <output_name> is the name of the output file.

The converter appends '*.sql*' to the <*input_filename*> to generate the output file when used with no parameters. For example, if the input filename is `TEST_FILE`, the output filename will be `TEST.sql`.

```
E.g., intspconv <input_filename>
```

Command Line Options

The command line options (-f, -b, -c) are important because an Interbase 4.0 stored procedure can be converted into two of three types of Oracle7 7.3 stored subprograms (procedure, function, or package). You must select a command line option that matches the original Interbase 4.0 functionality. The type of output varies depending on the command line option you use.

Since the option you select is applied to all stored procedures in the Interbase 4.0 source file, it is recommended that there be only one stored procedure in the Interbase 4.0 source/input file.

The three conversion options are:

-f `FOR <select_expr> DO <compound_statement>`

The command line option -f converts an Interbase 4.0 stored procedure into an Oracle Stored procedure. This option must be used in a particular set of conditions.

This option is used when the procedure has the following properties:

- It contains one or more FOR statements
- It does not contain any SUSPEND logic.

This option produces the following. Each for statement which is translated is converted into a explicit cursor. A LOOP condition will cycle through the cursor implementing the same functionality as the normal FOR statement.

-b All SUSPEND LOGIC (Complex)

The command line option *-b* converts an Interbase 4.0 stored procedure to a PL/SQL package. This option is used under a general rule

- The procedure must contain SUSPEND logic somewhere in it main block statement.

This plsql package produced by this option has an exposed PL/SQL Table of Records defined. This table of records denotes all return values from the procedure. A row is populated into the table each time the suspend statement is called.

-c SIMPLE SUSPEND LOGIC

The command line option *-c* converts an Interbase 4.0 stored procedure into a PL/SQL package with an exposed cursor. This option should be used under the following conditions.

- `FOR <select_expr> DO suspend;`

A procedure cursor variable is declared as part of the package body to correspond with the statement of the result set in the Interbase 4.0 stored procedure.

NB The only option available when converting Interbase 4.0 triggers is the -f option. SUSPEND logic makes no sense inside a trigger.

Other Command Line Options h,d,s,p

The command line options listed above are provided to help the user make most use of the tool.

-h This produces a banner with a list of command line options. It also gives the ordering of those options.

-d debug mode: This swith will allow you to enter Stored procedure commands and have them converted in an adhoc manner showing the main parts of the conversion process.

-s Scanner Phas debug: This allows the user to identify if the procedure they are converting has all the correct tokens available.

-p Parser phase debug: This allows the user to see what EBNF Interbase rules are being satisfied by the procedure that they are submitting for conversion.

Converter Error Messages

The *sp_converter* will recognise most Interbase 4.0 Stored Procedure lanaguage constructs; This assumes that the procedure being converted is one which is complete and functional on an Interbase 4.0 system. If it does come across an error, it aborts and gives a syntax error. The error message indicates the line number from the source Interbase 4.0 file. Examine the source code and line indicated by the error message. Remove any conflicting constructs from the source code so that you can convert the source file.

Converter Capabilities

Keep in mind the *sp_converter* cannot convert 100% and some manual changes are usually required. The list below tells you what results to expect from the *sp_converter*:

- Stored Procedures
  Converts an Interbase 4.0 stored procedure to one of the following:
  - PL/SQL Procedure
  - PL/SQL Package with a procedure returning a cursor Record.
  - PL/SQL Package with a procedure returning a PL/Sql table.
- Triggers

  Converts Interbase 4.0 triggers into Oracle7 Triggers.
- Datatypes
  Converts most of the base datatypes to Oracle7 equivalents.
- DML Statements
  Converts all DML statements .
- Procedural Language Constructs
  Converts most procedural language constructs including:

  IF ELSE
  FOR
  BLOCK (BEGIN and END)
  EXIT
  SUSPEND
- Interbase 4.0 - specific Functions
  Converts Interbase 4.0 - specific functions which have equivalent support on the Oracle7 Release 7.X side (either direct or indirect).
- Parameters
  Converts all the parameters passed to the stored procedure to corresponding Oracle7 Release 7.X parameters with the appropriate IN and OUT clauses. Converts all default values associated with parameters.
- Calls to Other Procedures
  Converts all calls to another procedure made from within the procedure.

Manual Conversion Considerations

The *sp_converter* cannot convert 100%; some manual conversion is required. Refer to the list below for manual conversion requirements and to Chapters 3, Interbase 4.0 Dynamic Sql *and PL/SQL Language Elements*, and Chapter 4, *Datatypes*, for detailed information.

- Exceptions

  Exceptions are recognised and produce a comment in the generated code. These must be converted manually. These will make it into a later release of the tool.

- Post Events

  Post events in triggers are recognised but commented out. These have to be converted manually.

- Datatypes
  The converter has a problem with time datatype in a lot of the functional part of the converter. Because of the unusual way that interbase deals with time, there is no concrete way as yet to convert date and time fields to dates in Oracle. This has been handled to a degree. For example,

Replace...	**...With...**
'0000'	00:00
'today'	SYSDATE
'now'	SYSDATE

Review the list above and make sure that all necessary manual conversion tasks are performed on the converted procedure. See Chapter 7, *INTERBASE 4.0 vs. PL/SQL Constructs*, for a detailed description of the conversion performed by the *sp_converter*.

You are now ready to convert Interbase 4.0 stored procedures & triggers to Oracle7 Release 7.X stored subprograms and triggers.

CHAPTER

3 USING PL/SQL PRODUCED IN PRO*C

This chapter discusses the possible ways of accessing the PL/SQL produced by the sp_converter in the realms of PRO*C. This chapter will look at two examples:

1. How to extract information from a PL/SQL table of records as a result of converting a complex SUSPEND statement
2. How to extract information from a Cursor which has been return from the PL/SQL Package.

- Sample Interbase Stored Procedure with complex suspend functionality
- Oracle7 package with PL/SQL table constructs.
- Sample packages PRO*C to call and extract information from PL/SQL tables

Sample Interbase Stored Procedure With Complex SUSPEND Functionality

```
ALTER PROCEDURE BAG_MANIFEST1 (FNO CHAR(8),
FDT CHAR(5))
RETURNS (S CHAR(5),BAG CHAR(10),C CHAR(5),
DEST CHAR(5),TIME_BSM CHAR(5),
TIME_LOADED CHAR(5))
AS
declare variable t_s char(5);
declare variable t_bag char(10);
declare variable t_c char(5);
declare variable t_dest char(3);
declare variable t_time_bsm char(5);
declare variable t_time_loaded char(5);
declare variable fid integer;
declare variable i smallint;
declare variable fn char(8);
declare variable fd char(5);
declare variable uno char(10);
begin
 uno = '';
 s = '';
 bag = '';
 c = '';
 dest = '';
 time_bsm = '';
 time_loaded = '';


 select flight_id, flight_no, flight_date
 from flight
```

```
 where flight_no = upper(:fno)
 and flight_date = upper(:fdt)
 into :fid, :fn, :fd;
 s = 'FLT:';
 bag = fn;
 suspend;
 s = 'DATE:';
 bag = fd;
 suspend;
 s = '';
 bag = '';

 for select flight_destination
  from flight_dests
  where flight_id = :fid
  into :bag
 do
 begin
  s = 'DEST:';
  suspend;
  s = '';
 end
 exit;
end ^
```

Generated PL/SQL From Complex SUSPEND Statements

```
CREATE OR REPLACE PACKAGE BAG_MANIFEST1_PACK
AS
      TYPE BAG_MANIFEST1_REC IS RECORD
       (
          S  CHAR(5), BAG  CHAR(10),
          C  CHAR(5), DEST  CHAR(5),
          TIME_BSM  CHAR(5), TIME_LOADED  CHAR(5)
       );
      TYPE BAG_MANIFEST1_TABLE_TYPE IS TABLE OF
       BAG_MANIFEST1_REC INDEX BY BINARY_INTEGER;

 -- Procedure Declaration
 PROCEDURE BAG_MANIFEST1
     (FNO IN OUT  CHAR(8),    FDT IN OUT  CHAR(5),
                        BAG_MANIFEST1_TABLE IN OUT
                     BAG_MANIFEST1_TABLE_TYPE);

END BAG_MANIFEST1_PACK;
/

CREATE OR REPLACE PACKAGE BODY BAG_MANIFEST1_PACK
AS
PROCEDURE BAG_MANIFEST1

     (FNO IN OUT  CHAR(8),    FDT IN OUT  CHAR(5),
                        BAG_MANIFEST1_TABLE IN OUT
                     BAG_MANIFEST1_TABLE_TYPE)
IS

t_s CHAR(5);
t_bag CHAR(10);
t_c CHAR(5);
t_dest CHAR(3);
t_time_bsm CHAR(5);
t_time_loaded CHAR(5);
fid NUMBER(10);
i NUMBER(6);
fn CHAR(8);
fd CHAR(5);
uno CHAR(10);

BEGIN

uno := ''; s := '';
bag := ''; c := '';
```

```
dest := '';time_bsm := '';
time_loaded := '';
.
.
.
.
DECLARE
 CURSOR CSR_0 IS SELECT flight_id fid,flight_no fn,flight_date
fd,flight_destination bag
FROM flight_dests WHERE flight_id = fid;
i BINARY_INTEGER:=0;
BEGIN
 FOR CSR_REC_0 IN CSR_0 LOOP
  s := 'DEST:';
 -- This is an insert in to PL/Sql Table;
  BAG_MANIFEST1_TABLE(i).S := CSR_REC_0.S;
  BAG_MANIFEST1_TABLE(i).BAG := CSR_REC_0.BAG;
  BAG_MANIFEST1_TABLE(i).C := CSR_REC_0.C;
  BAG_MANIFEST1_TABLE(i).DEST := CSR_REC_0.DEST;
  BAG_MANIFEST1_TABLE(i).TIME_BSM := CSR_REC_0.TIME_BSM;
  BAG_MANIFEST1_TABLE(i).TIME_LOADED := CSR_REC_0.TIME_LOADED;
   s:= '';
  i:=i+1;
 END LOOP;
END;

GOTO end_procedure;


<<end_procedure>>
NULL;
END BAG_MANIFEST1;
END BAG_MANIFEST1_PACK;

/
```

Calling PL/SQL Tables from PRO*C

This section will describe a small test package which has a PL/SQL table in it and will show the way to call this from PRO*C. Normally PL/SQL table have only one dimension. In this case the produced code from the converter mimicks a two-dimensional table through the use of records.

PL/SQL Test Package for Proc.

```
create or replace package test_rec as
type rec_type is record (f1 varchar2(10), f2 number);
type tab_type is table of rec_type index by
binary_integer;
procedure test_it( p1 in out tab_type );
end;
/
create or replace package body test_rec as
procedure test_it( p1 in out tab_type )
is
begin
    p1(1).f1 := p1(1).f1 || '*';
    p1(1).f2 := p1(1).f2 * 10;
    p1(2).f1 := p1(2).f1 || '*';
    p1(2).f2 := p1(2).f2 * 10;
end;
end;
/
```

PRO*C Code for PL/SQL Table of Records

```
#include <stdio.h>
#include <string.h>
/*
 * Macros for use with VARCHARs.
 */
#define TERM(X) ( X.arr[X.len] = '\0' )
#define SLEN(X) ( X.len = strlen(X.arr) )
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR oracleid[20];
    VARCHAR f1[3][30];
    int     f2[3];
EXEC SQL END DECLARE SECTION;
main()
{
    char action_str[30];
     EXEC SQL WHENEVER SQLERROR DO
o_error(action_str);
    /*
     * Connect to the database.
     */
     strcpy( oracleid.arr, "scott/tiger" );
    SLEN( oracleid );
     strcpy( action_str, "connecting to d/b" );
    EXEC SQL CONNECT :oracleid;
    EXEC SQL EXECUTE
        DECLARE
         my_tab test_rec.tab_type;
        BEGIN
         my_tab(1).f1 := 'Hello';
         my_tab(1).f2 := 1;
         my_tab(2).f1 := 'Bye';
         my_tab(2).f2 := 2;
         test_rec.test_it(my_tab);
```

```
          :f1(1) := my_tab(1).f1;
          :f1(2) := my_tab(2).f1;
          :f2(1) := my_tab(1).f2;
          :f2(2) := my_tab(2).f2;
         END;
         END-EXEC;
    TERM(f1[0]);
    TERM(f1[1]);
    printf("\n%s %d, %s %d", f1[0].arr, f2[0],
f1[1].arr, f2[1] );
    EXEC SQL WHENEVER SQLERROR CONTINUE;
     EXEC SQL ROLLBACK WORK RELEASE;
}

 int o_error( action_str )
char *action_str;
{
    int i;
    char error_str[150];
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     for ( i = 0; i < sqlca.sqlerrm.sqlerrml; i++ )
    {
   error_str[i] = sqlca.sqlerrm.sqlerrmc[i];
    }
    error_str[i] = '\0';
   printf( "\nFailed with following Oracle error while
   %s:\n\n%s", action_str, error_str );
     /*
     * Log off database.
     */
     EXEC SQL ROLLBACK WORK RELEASE;
     exit(1); }
```

CHAPTER

4

INTERBASE 4.0 AND PL/SQL LANGUAGE ELEMENTS

This chapter discusses the language elements in INTERBASE 4.0 and PL/SQL. INTERBASE 4.0 is the procedural SQL language for Interbase 4.0 and PL/SQL is the procedural SQL language for Oracle7.

This chapter contains information about:

- INTERBASE 4.0 language elements and equivalent PL/SQL constructs
- INTERBASE 4.0 and PL/SQL blocks
- transaction handling semantics
- exception and error handling semantics
- special global variables
- operators
- built-in functions
- sending data to the client: Result Sets
- DDL constructs within Interbase 4.0 stored procedures

INTERBASE 4.0 Language Elements and Equivalent PL/SQL Constructs

INTERBASE 4.0 supports a relatively primitive batch flow control language compared to PL/SQL. The INTERBASE 4.0 stored procedure language is very small and consists of the elements shown in the following table.

INTERBASE 4.0 Language Element	Definition	PL/SQL Constructs
Parameters	May be IN or IN OUT. OUT only is not supported.	Parameters are supported. May be IN, OUT, or IN OUT.
BEGIN	Used for scoping; declares the beginning of a new block. Blocks may be nested.	BEGIN
END	Used for scoping; declares the end of a block. Must be paired with a BEGIN.	END

IF <condition> THEN
  <compound_statement>
ELSE
  <compound_statement>
	IF ELSE IF type of blocks. Examples:
	
	IF (x>5) THEN
	  /* statement here */
	ELSE IF (x = 5)
	BEGIN
	  /* 1-n statements */
	END
	ELSE
	BEGIN
	  /* code here */
	END
		If <condition>
		  <block>
		[ELSIF <condition>
		  <block>
		  ...]
		[ELSE
		  <block>]
		ENDIF;

FOR <select_expr> DO
BEGIN
  <compound_statement>
END
 	
  where compound_statement is a block or a singular statement
	WHILE loop. Example:
	
	FOR SELECT avg(qty) from test
	  where t<500 into :x
	DO
	BEGIN
	  /* code here */
	END
		CURSOR C0 IS <select_expr>;
		OPEN C0;
		LOOP
		  FETCH cur_rec into XX
		  <statements>
		END LOOP;

INTERBASE 4.0 Language Element	Definition	PL/SQL Constructs
`DECLARE statement STATEMENT`	`DECLARE` declares a local variable.	`DECLARE varname datatype [DEFAULT <expression|value>] | [=<expresson|value>];`
`varname = value`	`Assignment`	`varname :=<value>;`

`CREATE PROCEDURE procedure`
`[ (param <datatype> [, param <datatype> …])]`
`[RETURNS datatype [, param <datatype> …]) ]`
`AS <procedure_body> [terminator]`
 	
`<procedure_body> = [ variable_declaration_list ]`
`<block>`
	Creates and compiles a stored procedure in the database.
	
	'`procedure`' is the name of the procedure.
		Procedure, function or a package.
		
		See Chapter 6, *Schema Objects*, for detailed syntax.

`OPEN cursor_name`	Opens the cursor.	PL/SQL offers same functionality.
`CLOSE cursor_name`	Closes the cursor.	PL/SQL offers same functionality.
`DECLARE cursor_name CURSOR FOR select_statement FOR {read only | update [of column_list]}`	Declares a cursor either as a read-only cursor or for update.	PL/SQL declare cursor. The read-only option should be ignored because it is not needed in Oracle.
`FETCH cursor_name [INTO fetch_list]`	Returns a row from the cursor result set.	PL/SQL offers the same functionality.

INTERBASE 4.0 Language Element	Definition	PL/SQL Constructs
`EXIT`	Unconditional 'GOTO' to the statement following the end of the current block of execution.	`EXIT`
`SUSPEND`	suspends execution of a select procedure until the next FETCH is issued and returns values to the calling application	None: It can be worked around by using a PL/SQL table in a Package.

INTERBASE 4.0 and PL/SQL Blocks

Interbase 4.0

A block in INTERBASE 4.0 is defined as the INTERBASE 4.0 statements enclosed by the keywords BEGIN and END. Blocks may be nested. All local variables for the block must be declared just before 'BEGIN'. The variables cease to exist after the completion of the block. Local variables for the entire stored procedure must be declared immediately after the AS keyword. Refer to Chapter 6, *Schema Objects*, for the syntax.

The general structure of a typical INTERBASE 4.0 procedure is as follows:

```
CREATE PROCEDURE <procedure_name>
   <parameters>
AS
BEGIN
   <procedure body>
END
```

Oracle7

PL/SQL is a block-structured language. The basic program unit in PL/SQL is the block. A PL/SQL block is defined by the following keywords:

DECLARE
BEGIN
EXCEPTION
END

These keywords partition the PL/SQL block as follows:

declarative
executable (mandatory)
exception-handling

You can nest a block within another block wherever you can place an executable statement. The nested blocks are the sub-blocks. A block or sub-block lets the developer group logically related declarations close to where they are used. Declarations are local to the block and cease to exist when the block completes. The identifiers declared in a PL/SQL block are considered local to that block and global to all its sub-blocks. You can define local subprograms in the declarative part of any block. However, you can call a local subprogram only from the block in which it is defined.

Recommendations:

The block structure of both languages is very similar except for the exception handling semantics.

In Oracle7 all the error conditions for the block can be dealt with from within the EXCEPTION section. If the block does not have an EXCEPTION section, the errors propagate to the outer block until an exception handler for the error condition is encountered.

Exception Handling Semantics

Interbase 4.0

In Interbase 4.0, the error handling is extremly similar to that of Oracle7. Below some recommendations for these are given below. These are not converted in this release. See below for some recommendations for converting these.

Oracle7

In Oracle7, each SQL statement is automatically checked for errors before proceeding with the next statement. If an error occurs, control immediately jumps to an exception handler if one exists. This frees the developer from needing to check the status of every SQL statement. For example, if a SELECT statement does not find any row in the database, an exception is raised. The corresponding exception handler part of the block should include the code to deal with this error. The built-in procedure RAISE_APPLICATION_ERROR notifies the client of the server error condition and returns immediately to the calling routine.

Oracle7 places an implicit SAVEPOINT at the beginning of a procedure. The built-in procedure RAISE_APPLICATION_ERROR rolls back to this SAVEPOINT or the last committed transaction within the procedure. The control is returned to the calling routine.

Oracle7's RAISE_APPLICATION_ERROR statement allows the user to customize the error message. If an exception is raised, SQLCODE is returned automatically by PL/SQL to the caller. It keeps propagating until it is handled.

Recommendations:

In Oracle7, to simulate Interbase 4.0 behavior, you must enclose each SQL statement in an equivalent PL/SQL block. This block must deal with the exceptions that need to be trapped for the SQL statement.

If the RAISERROR statement in a Interbase 4.0 stored procedure is immediately followed by the RETURN statement, these two statements can be converted to Oracle7's RAISE_APPLICATION_ERROR.

Operators

Comparison Operators

This section compares the operators used in the Interbase 4.0 and Oracle7 databases.
Comparison operators are used in WHERE clauses and COLUMN check constraints/rules to compare values.

Operator	Same in Both Databases	Interbase 4.0 Only	Oracle7 Only
Equal to?	=		
Not equal to?	!= <>		^=
Less than?	<		
Greater than?	>		
Less than or equal to?	<=		
Greater than or equal to?	>=		
Greater than or equal to x and less than or equal to y ?			BETWEEN x AND y
Less than x or greater than y			NOT BETWEEN x AND y
No value exists?	IS NULL	= NULL	
A value exists?	IS NOT NULL	!= NULL	
At least one row returned by query?	EXISTS (query)		
No rows returned by query?	NOT EXISTS (query)		

Recommendations:

1. **Convert all !< and !> to >= and <=**

   If you have this in Interbase 4.0:

   ```
   WHERE col1 !< 100
   ```

Convert to this for Oracle7:

```
WHERE col1 >= 100
```

2. Change NULL constructs:

The table below shows that in Oracle7, NULL is never equal to NULL. Change the all = NULL constructs to IS NULL to retain the same functionality.

NULL Construct	Interbase 4.0	Oracle7
`where col1 = NULL`	depends on the data	FALSE
`where col1 != NULL`	depends on the data	TRUE
`where NULL = NULL`	TRUE	FALSE

If you have this in Interbase 4.0:

```
WHERE col1 = NULL
```

Convert to this for Oracle7:

```
WHERE col1 IS NULL
```

Arithmetic Operators

Operator	Same in Both Databases	Interbase 4.0 Only	Oracle7 Only
Add	+		
Subtract	-		
Multiply	*		
Divide	/		
Modulo			`mod(x, y)`

String Operators

Operator	Same in Both Databases	Interbase 4.0 Only	Oracle7 Only
Concatenate	`||`		
Identify Literal	`'this is a string'`	`"this is also a string"`	

Recommendations:

1. Replace all double-quote string identifiers with single-quote identifiers.

Set Operators

Operator	Same in Both Databases	Interbase 4.0 Only	Oracle7 Only
Distinct row from either query	`UNION`		
All rows from both queries	`UNION ALL`		
All distinct rows in both queries			`INTERSECT`
All distinct rows in the first query but not in the second query			`MINUS`

Built-in Functions

Character Functions

Interbase 4.0	Oracle7	Description
`AVG(list)`	`AVG(list)`	Calculates the average of a set of values
`Cast( val as datatype)`	`to_char, to_number, to_date, to_label, chartorowid, rowidtochar, hextochar, chartohex`	Converts one datatype to another using the optional format. The majority of the functionality can be matched. Refer to your Oracle manual for more information.
`COUNT()`	`COUNT()`	Returns the number of rows that satisfy a query's search condition
`GEN_ID`	`variable.NEXTVAL`	generates the next value in a sequence. In Oracle the sequence is built separatly from the procedure and so must only be incremented.
`MAX()`	`MAX()`	Retrieves the maximum value froma set of values
`MIN()`	`MIN()`	Retrieves the minimum value froma set of values
`SUM()`	`SUM()`	Totals the values in a set of numeric values
`UPPER()`	`UPPER()`	Coverts a string to all uppercase.

CHAPTER

5 DATATYPES

This chapter provides detailed descriptions of the datatypes used by the Interbase 4.0 INTERBASE 4.0 stored procedures and Oracle7 PL/SQL stored subprograms (procedures, functions, or packages). Specifically, this chapter contains the following information:

- a table showing the available Interbase 4.0 server datatypes and how they are mapped to Oracle7 datatypes
- recommendations based on the information listed in the table

INTERBASE 4.0 local variables can be any server datatype except TEXT and IMAGE. PL/SQL local variables can be any server datatype including:

BINARY_INTEGER

BOOLEAN

PL/SQL local variables can also be these composite datatypes allowed by PL/SQL:

RECORD

TABLE

Server Datatypes

The following table describes Interbase 4.0 server datatypes and lists the Oracle7 equivalent datatypes chosen by the *sp_converter*.

Interbase 4.0	**Description**	**Oracle7**
FLOAT	32 bits Single Precision: 7 digits of precision	NUMBER
INTEGER	32 bits - Signed Long	INTEGER
SMALLINT	16 bits - signed short (word)	INTEGER
DECIMAL(p,[q]), NUMERIC(p,[q])	Packed decimal number, *p* digits and sign, with assumed decimal point *q* digits from the right. *p* is in range 1..15 and the scale *q* in the range 0..15.	NUMBER
DOUBLE PRECISION	Scientific: 15 digits of precision	NUMBER
CHAR(*n*)	Fixed-length string of exactly *n* 8-bit characters, blank padded. Synonym for CHARACTER. $0 < n < 256$	CHAR(*n*)
VARCHAR(*n*)	Varying-length character string. $0 < n < 256$	VARCHAR2(*n*)
BLOB	Binary Large Object. Stores large data, such as graphics, text and digitized voice. Basic Structural unit: segment.	RAW
DATE	64 bits - Also included time information	DATE

Interbase 4.0	Description	Oracle7
[Not Applicable]	The BINARY_INTEGER datatype represents values stored as signed binary integers. Unlike NUMBER values, these can be used in calculations without conversion and thus improve performance. There are two sub-types of BINARY_INTEGER: NATURAL (0..2147483647) and POSITIVE (1..2147483647).	BINARY_INTEGER
[Not Applicable]	TRUE, FALSE, or NULL values can be stored as BOOLEAN values.	BOOLEAN

Composite Datatypes

Interbase 4.0 does not have composite datatypes.

Oracle7	Comments
RECORD	You can declare a variable to be of type RECORD. Records have uniquely named fields. Logically related data that is dissimilar in type can be held together in a record as a logical unit.
TABLE	PL/SQL tables can have one column and a primary key, neither of which can be named. However, the table column may be a record therefore conceptually increasing the number of columns availableThe column can belong to any scalar datatype. The primary key must belong to type BINARY_INTEGER.

CHAPTER

6 SCHEMA OBJECTS

This chapter provides comparisons of these Interbase 4.0 and Oracle7 schema objects:

- Procedure
- Function
- Package

Each schema object is compared in separate tables based on:

- Create
- Drop
- Execute
- Alter

Each table is divided in these sections:

- Syntax
- Description
- Permissions
- Examples

Procedure

Create:

Interbase 4.0	Oracle7

Syntax:
`CREATE PROCEDURE procedure`
`  [(formal_parameter`
`  formal_parameter_datatype`
`  [, formal_parameter`
`  formal_parameter_datatype])]`
`RETURNS`
`  [(formal_parameter`
`  formal_parameter_datatype`
`  [, formal_parameter`
`  formal_parameter_datatype])]`
`AS`
`  <procedure_body>`
 	
`<procedure_body>=`
`  [variable_decalration_list]`
`  <block>`
 	
`<variable_declaration_list> =`
`  DECLARE VARIABLE var <datatpe>;`
`  [ DECLARE VARIABLE var <datatype>; …]`
	Syntax:
	`CREATE [OR REPLACE] PROCEDURE`
	`  [schema.]procedure`
	`[()`
	`  [formal_parameter`
	`  [IN | OUT | IN OUT]`
	`  formal_parameter_datatype]`
	`  [DEFAULT default_value]`
	`  [,formal_parameter`
	`  [IN | OUT | IN OUT]`
	`  formal_parameter_datatype]`
	`  [DEFAULT default_value]] ...`
	`[)]`
	`IS | AS`
	`[local_variable datatype;]...`
	`BEGIN`
	`  PL/SQL statements | PL/SQL blocks`
	`END;`

Description: The CREATE PROCEDURE statement creates the named stored procedure in the database. The keyword AS indicates the start of the body of the procedure.	**Description:** The OR REPLACE keywords replace the procedure by the new definition if it already exists. The parameters passed to the PL/SQL procedure can be specified as 'IN' (input) , 'OUT' (output only) or 'IN OUT‘ (input and output). In the absence of these keywords the parameter is assumed to be the ‘IN’ parameter. The keyword IS or AS indicates the start of the procedure. After the keyword IS or AS and before the keyword BEGIN, the local variables are declared.

Interbase 4.0	Oracle7
	The keywords BEGIN and END enclose the body of the procedure. Refer to the discussion on the PL/SQL statements and block structure in Chapter 3 for more information on the contents of the PL/SQL procedure body.
Permissions: In interbase there is only one user and therefore no permissions are required.	**Permissions:** To create a procedure in the user's own schema, the user must have CREATE PROCEDURE system privilege. To create a procedure in another schema, the user must have CREATE ANY PROCEDURE system privilege.

Example:
`CREATE PROCEDURE myproc`
`(cust char(30))`
`RETURNS`
`(cust_id integer,`
`param3 char)`
`AS`
`BEGIN`
`DECLARE VARIABLE local_var1 integer;`
`DECLARE VARIABLE local_var2 char(4);`
 	
`local_var2 = time_now(today)`
`param3 = local_var2;`
 	
`SELECT customer_id`
`FROM customer`
`WHERE customer = @cust`
`INTO :local_var1`
`cust_id = local_var1`
`END`
	Example:
	`CREATE OR REPLACE PROCEDURE sam.credit`
	`(`
	`acc_no IN NUMBER DEFAULT 0,`
	`acc IN VARCHAR2,`
	`amount IN NUMBER,`
	`return_status OUT NUMBER`
	`)`
	`AS`
	`BEGIN`
	`UPDATE accounts`
	`SET balance = balance + amount`
	`WHERE account_id = acc_no;`
	`EXCEPTION`
	`WHEN SQL%NOTFOUND THEN`
	`RAISE_APPLICATION_ERROR(-20101,`
	`'Error updating accounts table');`
	`END;`

Recommendations:

Functionally identical parts can be identified in the Interbase 4.0 procedure and PL/SQL procedure structure. It is therefore easy to automate the conversion of most of the constructs from Interbase 4.0 to Oracle7.

OR REPLACE keywords in an Oracle7 CREATE PROCEDURE statement provide an elegant way of recreating the procedure. In Interbase 4.0, the procedure must be dropped explicitly before replacing it.

Procedure

Drop:

Interbase 4.0	Oracle7
Syntax: `DROP PROCEDURE procedure`	**Syntax:** `DROP PROCEDURE [schema.]procedure`
Description: The procedure definition is deleted from the data dictionary. All the objects that reference this procedure must have references to this procedure removed.	**Description:** When a procedure is dropped, Oracle7 invalidates all the local objects that reference the dropped procedure.
Permissions: N/A	**Permissions:** The procedure must be in the schema of the user or the user must have DROP ANY PROCEDURE system privilege to execute this command.
Example: `DROP PROCEDURE myproc`	**Example:** `DROP PROCEDURE sam.credit;`

Recommendations:

The above statement does not have any effect on the conversion process. This information is provided for reference.

Procedure

Execute:

Interbase 4.0	Oracle7

Syntax:
`EXECUTE PROCEDURE procedure`
`  [(formal_parameter`
`  [,formal_parameter])]`
`[RETURNING_VALUES`
`  [(formal_parameter`
`  [,formal_parameter])]`
	Syntax:
	`procedure`
	`([([{actual_parameter |`
	`  constant_literal |`
	`  formal_parameter =>`
	`  {actual_parameter |`
	`  constant_literal}`
	`  }]`
	`[,{actual_parameter |`
	`  constant_literal |`
	`  formal_parameter =>`
	`  {actual_parameter |`
	`  constant_literal}`
	`  }] ....`
	`)]`

Description: Interbase 4.0 *Positional notation:* The actual parameters are supplied to the procedure in the same order as the formal parameters in the procedure definition.	**Description:** Oracle7 PL/SQL procedures send data back to the calling routine by means of OUT parameters. Oracle7 offers FUNCTIONS that are a different type of schema objects. Functions can return an atomic value to the calling routine using the RETURN statement. RETURN statement can return value of any datatype. The ‘formal_parameter’ is the parameter in the procedure definition. The ‘actual_parameter’ is defined in the local block which calls the procedure supplying the value of the actual parameter for the respective formal parameter. The association between an actual and formal parameter can be indicated using either positional or named notation. *Positional notation:* The actual parameters are supplied to the procedure in the same order as the formal parameters in the

Interbase 4.0	Oracle7

	procedure definition.
	
	Named notation:
	
	The actual parameters are supplied to the procedure in an order different than that of the formal parameters in the procedure definition by using the name of the formal parameter as:
	
	`formal_parameter => actual_parameter`
	
	A constant literal can be specified in the place of
	
	`'actual_parameter' as:`
	`  formal_parameter => 10`
	
	If the 'formal_parameter' is specified as OUT or IN OUT in the procedure definition, the value will be made available to the calling routine after the execution of the procedure.

Permissions:	**Permissions:** The user should have EXECUTE privilege on the named procedure. The user need not have explicit privileges to access the underlying objects referred to within the PL/SQL procedure.

Example:	**Example:**
	
	Positional notation:
	
	`credit (accno, accname, amt, retstat);`
	
	Named notation:
	
	`credit (acc_no => accno, acc => accname,`
	`  amount => amt,`
	`  return_status => retstat)`
	
	Mixed notation (where positional notation must precede named notation):
	
	`credit (accno, accname, amount => amt,`
	`  return_status => retstat)`

Procedure

Alter:

Interbase 4.0	Oracle7

Syntax:
`ALTER PROCEDURE procedure`
`  [(formal_parameter`
`  formal_parameter_datatype`
`  [, formal_parameter`
`  formal_parameter_datatype])]`
`RETURNS`
`  [(formal_parameter`
`  formal_parameter_datatype`
`  [, formal_parameter`
`  formal_parameter_datatype])]`
`AS`
`  <procedure_body>`
 	
`<procedure_body>=`
`  [variable_decalration_list]`
`  <block>`
 	
`<variable_declaration_list> =`
`  DECLARE VARIABLE var <datatpe>;`
`  [ DECLARE VARIABLE var <datatype>; …]`
	Syntax:
	`ALTER PROCEDURE [schema.]procedure COMPILE`

Description: This command causes the recompilation of the procedure.	**Description:** This command causes the recompilation of the procedure. Procedures that become invalid for some reason should be recompiled explicitly using this command. Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead.
Permissions:	**Permissions:** The procedure must be in the user's schema or the user must have ALTER ANY PROCEDURE privilege to use this command.

Interbase 4.0	Oracle7

Example:
`CREATE PROCEDURE myproc`
`(cust char(30))`
`RETURNS`
`(cust_id integer,`
`param3 char)`
`AS`
`BEGIN`
`DECLARE VARIABLE local_var1 integer;`
`DECLARE VARIABLE local_var2 char(4);`
 	
`local_var2 = time_now(today)`
`param3 = local_var2;`
 	
`SELECT customer_id`
`FROM customer`
`WHERE customer = @cust`
`INTO :local_var1`
`cust_id = local_var1`
`END`
	Example:
	`ALTER PROCEDURE sam.credit COMPILE;`

Package

Create:

Interbase 4.0	Oracle7

Syntax:
 	
Interbase 4.0 does not support this concept.
	Syntax:
	
	`CREATE [OR REPLACE] PACKAGE [user.]package {IS | AS}`
	`{variable_declaration |`
	`cursor_specification |`
	`exception_declaration |`
	`record_declaration |`
	`plsql_table_declaration |`
	`procedure_specification |`
	`function_specification |`
	`[{variable_declaration |`
	`cursor_specification |`
	`exception_declaration |`
	`record_declaration |`
	`plsql_table_declaration |`
	`procedure_specification |`
	`function_specification}; ]...}`
	`END [package]`

Description: N/A	**Description:** This is the external or public part of the package. CREATE PACKAGE sets up the specification for a PL/SQL package which can be a group of procedures, functions, exception, variables, constants, and cursors. Functions and procedures of the package can share data through variables, constants. and cursors. The OR REPLACE keywords replace the package by the new definition if it already exists. This requires recompilation of the package and any objects that depend on its specification. Refer to the discussion on the PL/SQL statements and block structure for more information on the contents of the PL/SQL package body.

Interbase 4.0	Oracle7
Permissions: N/A	**Permissions:** To create a package in the user's own schema, the user must have CREATE PROCEDURE system privilege. To create a package in another schema, the user must have CREATE ANY PROCEDURE system privilege.

Example:
 	
N/A
	Example:
	
	`CREATE PACKAGE emp_actions AS`
	`  -- specification`
	`  TYPE EmpRecTyp IS RECORD (emp_id INTEGER, salary REAL);`
	`  CURSOR desc_salary (emp_id NUMBER) RETURN EmpRecTyp;`
	
	`  PROCEDURE hire_employee`
	`    (ename   CHAR,`
	`     jobCHAR,`
	`     mgr     NUMBER,`
	`     sal     NUMBER,`
	`     comm    NUMBER,`
	`     deptno  NUMBER);`
	`  PROCEDURE fire-employee (emp_id NUMBER);`
	`END emp_actions;`

Package

Drop:

Interbase 4.0	Oracle7
Syntax: Interbase 4.0 does not support this concept.	**Syntax:** `DROP PACKAGE [BODY] [schema.]package`
Description: N/A	**Description:** The BODY option drops only the body of the package. If you omit BODY, Oracle7 drops both the body and specification of the package. If you drop the body and specification of the package, Oracle7 invalidates any local objects that depend on the package specification. 'schema' is the schema containing the package. If you omit 'schema.', Oracle7 assumes the package is in your own schema. When a package is dropped, Oracle7 invalidates all the local objects that reference the dropped package.
Permissions: N/A	**Permissions:** The package must be in the schema of the user or the user must have DROP ANY PROCEDURE system privilege to execute this command.
Example: N/A	**Example:** `DROP PACKAGE emp_actions;`

Package

Alter:

Interbase 4.0	Oracle7
Syntax: Interbase 4.0does not support this concept.	**Syntax:** `ALTER PACKAGE [user.]package COMPILE [PACKAGE | BODY]`
Description: N/A	**Description:** Packages that become invalid for some reason should be recompiled explicitly using this command. This command causes the recompilation of all package objects together. You cannot use the ALTER PROCEDURE or ALTER FUNCTION commands to individually recompile a procedure or function that is part of a package. PACKAGE, the default option, recompiles the package body and specification. BODY recompiles only the package body. Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead.
Permissions: N/A	**Permissions:** The package must be in the user's schema or the user must have ALTER ANY PROCEDURE privilege to use this command.
Example: N/A	**Example:** `ALTER PACKAGE emp_actions COMPILE PACKAGE`

Package Body

Create:

Interbase 4.0	Oracle7
Syntax: Interbase 4.0 does not support this concept.	**Syntax:** `CREATE [OR REPLACE] PACKAGE BODY [schema.]package {IS | AS} pl/sql_package_body`
Description: N/A	**Description:** This is the internal or private part of the package. CREATE PACKAGE creates the body of a stored package. OR REPLACE recreates the package body if it already exists. If you change a package body, Oracle7 recompiles it. 'schema' is the schema to contain the package. If omitted, the package is created in your current schema. 'package' is the of the package to be created. 'pl/sql_package_body' is the package body which can declare and define program objects. (See the *PL/SQL User's Guide and Reference* for information on writing package bodies.)
Permissions: N/A	**Permissions:** To create a package in your own schema, you must have CREATE PROCEDURE privilege. To create a package in another user's schema, you must have CREATE ANY PROCEDURE privilege.

Interbase 4.0	Oracle7

Example:
 	
N/A
	Example:
	
	`CREATE PACKAGE BODY emp_actions AS`
	`-- body`
	`  CURSOR desc_salary (emp_id NUMBER)`
	`  RETURN EmpRecTyp IS`
	`  SELECT empno, sal FROM emp`
	`  ORDER BY sal DESC;`
	`  PROCEDURE hire_employee`
	`  (ename  CHAR,`
	`  job  CHAR,`
	`  mgr  NUMBER,`
	`  sal  NUMBER,`
	`  comm  NUMBER,`
	`  deptno  NUMBER) IS`
	`  BEGIN`
	`  INSERT INTO emp VALUES`
	`  (empno_seq.NEXTVAL, ename,`
	`  job, mgr, SYSDATE, sal,`
	`  comm, deptno);`
	`  END hire_employee;`
	
	`  PROCEDURE fire_employee`
	`  (emp_id  NUMBER) IS`
	`  BEGIN`
	`  DELETE FROM emp`
	`  WHERE empno = emp_id;`
	`  END fire_employee;`
	
	`END emp_actions;`

Package Body

Drop:

Interbase 4.0	Oracle7
Syntax: Interbase 4.0 does not support this concept.	**Syntax:** `DROP PACKAGE [BODY] [schema.]package`
Description: N/A	**Description:** The BODY option drops only the body of the package. If you omit BODY, Oracle7 drops both the body and specification of the package. If you drop the body and specification of the package, Oracle7 invalidates any local objects that depend on the package specification. 'schema' is the schema containing the package. If you omit 'schema.', Oracle7 assumes the package is in your own schema. When a package is dropped, Oracle7 invalidates all the local objects that reference the dropped package.
Permissions: N/A	**Permissions:** The package must be in the schema of the user or the user must have DROP ANY PROCEDURE system privilege to execute this command.
Example: N/A	**Example:** `DROP PACKAGE BODY emp_actions;`

Package Body

Alter:

Interbase 4.0	Oracle7
Syntax: Interbase 4.0 does not support this concept.	**Syntax:** `ALTER PACKAGE [user.]package COMPILE [PACKAGE | BODY]`
Description: N/A	**Description:** Packages that become invalid for some reason should be recompiled explicitly using this command. This command causes the recompilation of all package objects together. You cannot use the ALTER PROCEDURE or ALTER FUNCTION commands to individually recompile a procedure or function that is part of a package. PACKAGE, the default option, recompiles the package body and specification. BODY recompiles only the package body. Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead.
Permissions: N/A	**Permissions:** The package must be in the user's schema or the user must have ALTER ANY PROCEDURE privilege to use this command.
Example: N/A	**Example:** `ALTER PACKAGE emp_actions COMPILE BODY`

CHAPTER

7

INTERBASE 4.0 VS. PL/SQL CONSTRUCTS

This chapter provides information about the Interbase 4.0 constructs and equivalent Oracle7 constructs generated by the *sp_converter*. The conversions of the following constructs are discussed in detail:

`CREATE PROCEDURE` Statement
`DECLARE` Statement
`IF` Statement
`EXECUTE` Statement
FOR Statement
`ASSIGNMENT` Statement
SELECT Statement

The syntax for the Interbase 4.0 constructs and their Oracle7 equivalents are listed and comments about conversion considerations are provided.

Note that the procedures in the 'Oracle7' column are the direct output of the *sp_converter*. These PL/SQL procedures have more lines of code compared to the source Interbase 4.0 procedures as these PL/SQL procedures are converted to emulate Interbase 4.0 functionality. The PL/SQL procedures written from scratch for the same functionality in Oracle7 would be much more compact.

Also note that the PL/SQL procedures generated by the *sp_converter* indicate the manual conversion needed by adding appropriate commands.

In general, the *sp_converter* deals with the Interbase 4.0 T-SQL constructs in one of the following ways:

1. The ANSI standard SQL statements are converted to PL/SQL because it supports ANSI standard SQL.
2. Interbase 4.0-specific constructs are converted into PL/SQL constructs if the equivalent constructs are available in PL/SQL.
3. Some Interbase 4.0-specific constructs are ignored and appropriate comments are incorporated in the output file.
4. Constructs that need manual conversion are wrapped around with proper comments in the output file.
5. For Interbase 4.0-specific constructs that result in syntax errors, an appropriate error message is displayed including the line number.

CREATE PROCEDURE Statement

Interbase 4.0	Oracle7

CREATE PROC proc1
AS
BEGIN
  exit;
END;
	CREATE OR REPLACE PROCEDURE proc1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	BEGIN
	 RETURN /* 0 */;
	END;
	
	OR...
	
	CREATE OR REPLACE FUNCTION proc1
	RETURN INTEGER
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	BEGIN
	 RETURN 0 ;
	END;

Parameter Passing

Interbase 4.0	Oracle7

CREATE PROC proc1
@x int=-1,
@y money,
@z bit OUT,
@a char(20) = 'TEST'
AS
    RETURN 0
	CREATE OR REPLACE PROCEDURE proc1(
	i_x INTEGER DEFAULT -1,
	i_y NUMBER ,
	i_z IN OUT NUMBER ,
	i_a CHAR DEFAULT 'TEST')
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	BEGIN
	        RETURN /* 0 */;
	END;
	/

Comments

Parameter passing is almost the same in Interbase 4.0 and Oracle7. By default, all the parameters are INPUT parameters, if not specified otherwise.

The value of the INPUT parameter cannot be changed from within the PL/SQL procedure. Thus an INPUT parameter cannot be assigned any values nor can it be passed to another procedure as an OUT parameter. In Oracle7 only IN parameters can be assigned a default value.

The @ sign in a parameter name declaration is converted to ‘i_’ in Oracle7.

In Oracle7, the datatype definition does not include length/size.

Interbase 4.0 datatypes are converted to Oracle7 base datatypes. For example, all Interbase 4.0 numeric datatypes are converted to NUMBER and all alphanumeric datatypes are converted to VARCHAR2 and CHAR in Oracle7.

DECLARE Statement

Interbase 4.0	Oracle7

CREATE PROC proc1
AS
DECLARE
@x int,
@y money,
@z bit,
@a char(20)
RETURN 0
GO
	CREATE OR REPLACE PROCEDURE proc1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	i_x INTEGER;
	i_y NUMBER;
	i_z NUMBER;
	i_a CHAR(20);
	BEGIN
	RETURN /* 0 */;
	END;
	/

Comments

Interbase 4.0 and Oracle7 follow similar rules for declaring local variables. When automatically converted, a variable name has the @ sign replaced by ‘i_’.

The converter overrides the scope rule for variable declarations. As a result, all the local variables are defined at the top of the procedure body in Oracle7.

User-defined datatypes from Interbase 4.0 are not converted to Oracle7 datatypes.

You must edit the source file and replace the user-defined datatypes to the equivalent base datatypes.

IF Statement

Interbase 4.0	Oracle7

Example 1:
`CREATE PROC proc1 (Flag integer)`
`AS`
`BEGIN`
`DECLARE VARIABLE x integer;`
`IF ( Flag=0 )THEN`
`  x = -1`
`ELSE`
`  x = 10`
`END`
	Example 1:
	`CREATE OR REPLACE PROCEDURE proc1(`
	`i_Flag INTEGER DEFAULT 0)`
	`AS`
	`StoO_selcnt INTEGER;`
	`StoO_error  INTEGER;`
	`StoO_rowcnt INTEGER;`
	`i_x INTEGER;`
	`BEGIN`
	`  IF(i_Flag = 0) THEN`
	`   i_x := -1;`
	`  ELSE`
	`   i_x :=  10 ;`
	`  END IF;`
	`END;`

Example 2:
`CREATE PROC proc1 (Flag char(2))`
`AS`
`BEGIN`
`DECLARE VARIABLE x integer;`
`IF ( Flag='' ) THEN`
`  x = -1;`
`ELSE IF (Flag = 'a') THEN`
`  x = 10;`
`ELSE IF (Flag = 'b') THEN`
`  x = 20;`
`END`
	Example 2:
	`CREATE OR REPLACE PROCEDURE proc1(`
	`i_Flag CHAR DEFAULT '')`
	`AS`
	`StoO_selcnt INTEGER;`
	`StoO_error  INTEGER;`
	`StoO_rowcnt INTEGER;`
	`i_x INTEGER;`
	`BEGIN`
	`  IF(i_Flag IS NULL) THEN`
	`   i_x := -1;`
	`  ELSIF (i_Flag = 'a') THEN`
	`   i_x :=  10 ;`
	`  ELSIF (i_Flag = 'b') THEN`
	`   i_x := 20`
	`  END IF;`
	`END;`

Example 3:
`CREATE PROC proc1`
`AS`
`BEGIN`
`DECLARE @x int`
`IF EXISTS ( SELECT * FROM table2 )`
`  x = -1;`
`END`
	Example 3:
	`CREATE OR REPLACE PROCEDURE proc1`
	`AS`
	`StoO_selcnt INTEGER;`
	`StoO_error  INTEGER;`
	`StoO_rowcnt INTEGER;`
	`i_x INTEGER;`
	`BEGIN`
	`  BEGIN`

Interbase 4.0	Oracle7

	SELECT 1 INTO StoO_selcnt
	FROM DUAL
	WHERE EXISTS (SELECT * FROM table2);
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_selcnt := 0;
	StoO_error := SQLCODE;
	END;
	IF StoO_selcnt != 0 THEN
	i_x := -1 ;
	END IF;
	END;

WHILE Statement

Interbase 4.0	Oracle7

Example 1:
`CREATE PROC proc1`
`@i int`
`AS`
`  WHILE @i > 0`
`  BEGIN`
`  print 'Looping inside WHILE....'`
`  SELECT @i = @i + 1`
`  END`
	Example 1:
	`CREATE OR REPLACE PROCEDURE proc1 (`
	`i_i NUMBER )`
	`AS`
	`StoO_selcnt  INTEGER;`
	`StoO_error  INTEGER;`
	`StoO_rowcnt  INTEGER;`
	`BEGIN`
	`<<i_loop1>>`
	`WHILE i_i > 0 LOOP`
	`BEGIN`
	`DBMS_OUTPUT.PUT_LINE`
	`  ('Looping inside WHILE....') ;`
	`  i_i :=  i_i + 1 ;`
	`END;`
	`END LOOP;`
	`END;`
	`/`

Example 2:
`CREATE PROC proc1`
`@i int,`
`@y int`
`AS`
`  WHILE @i > 0`
`  BEGIN`
`  print 'Looping inside WHILE....'`
`  SELECT @i = @i + 1`
`  END`
	Example 2:
	`CREATE OR REPLACE PROCEDURE proc1 (`
	`i_i NUMBER,`
	`i_y NUMBER )`
	`AS`
	`StoO_selcnt  INTEGER;`
	`StoO_error  INTEGER;`
	`StoO_rowcnt  INTEGER;`
	`BEGIN`
	`  <<i_loop2>>`
	`  WHILE i_i > 1 LOOP`
	`  BEGIN`
	`  IF i_y > -1 THEN`
	`  GOTO i_loop2;`
	`  END IF;`
	`  i_y :=  i_y + 5 ;`
	`  END;`
	`  END LOOP;`
	`END;`
	`/`

Interbase 4.0	Oracle7

Example 3:
`CREATE PROC proc1`
`AS`
`DECLARE @sal money`
`SELECT @sal = 0`
`WHILE EXISTS(SELECT * FROM emp where sal <`
`  @sal )`
`BEGIN`
` SELECT @sal = @sal + 99`
 	
` DELETE emp`
` WHERE sal < @sal`
`END`
`GO`
	Example 3:
	`CREATE OR REPLACE PROCEDURE proc1`
	`AS`
	`StoO_selcnt  INTEGER;`
	`StoO_error  INTEGER;`
	`StoO_rowcnt  INTEGER;`
	`i_sal NUMBER;`
	`BEGIN`
	` i_sal := 0 ;`
	
	` <<i_loop1>>`
	` WHILE 1 = 1 LOOP`
	` BEGIN`
	` SELECT 1 INTO StoO_selcnt FROM DUAL`
	` WHERE EXISTS (SELECT * FROM emp`
	` WHERE sal < i_sal);`
	` EXCEPTION`
	`  WHEN TOO_MANY_ROWS THEN`
	`  StoO_selcnt := 2;`
	`  WHEN OTHERS THEN`
	`  StoO_selcnt := 0;`
	`  StoO_error := SQLCODE;`
	` END;`
	` IF StoO_selcnt != 1 THEN`
	`  EXIT;`
	` END IF;`
	` i_sal := i_sal + 99 ;`
	` BEGIN`
	` DELETE emp`
	` WHERE sal < i_sal;`
	` StoO_rowcnt := SQL%ROWCOUNT;`
	` EXCEPTION`
	`  WHEN OTHERS THEN`
	`  StoO_error := SQLCODE;`
	` END;`
	`END;`
	` END LOOP;`
	`END;`
	`/`

Interbase 4.0	Oracle7

Example 4:
`CREATE PROC proc1`
`AS`
 	
`DECLARE @sal money`
 	
`WHILE (SELECT count (*) FROM emp ) > 0`
`BEGIN`
`SELECT @sal = max(sal) from emp`
`WHERE stat = 1`
 	
`DELETE emp`
`WHERE sal < @sal`
`END`
`GO`
	Example 4:
	`CREATE OR REPLACE PROCEDURE proc1`
	`AS`
	`StoO_selcnt INTEGER;`
	`StoO_error INTEGER;`
	`StoO_rowcnt INTEGER;`
	`i_sal NUMBER;`
	`BEGIN`
	`<<i_loop1>>`
	`WHILE 1 = 1 LOOP`
	`BEGIN`
	`SELECT 1 INTO StoO_selcnt FROM DUAL`
	`WHERE 0 < (SELECT COUNT (*) FROM`
	`emp);`
	`EXCEPTION`
	`WHEN TOO_MANY_ROWS THEN`
	`StoO_selcnt := 2;`
	`WHEN OTHERS THEN`
	`StoO_selcnt := 0;`
	`StoO_error := SQLCODE;`
	`END;`
	`IF StoO_selcnt != 1 THEN`
	`EXIT;`
	`END IF;`
	`BEGIN`
	`SELECT MAX(sal)`
	`INTO i_sal`
	`FROM emp`
	`WHERE stat = 1;`
	`StoO_rowcnt := SQL%ROWCOUNT;`
	`EXCEPTION`
	`WHEN TOO_MANY_ROWS THEN`
	`StoO_rowcnt := 2;`
	`WHEN OTHERS THEN`
	`StoO_rowcnt := 0;`
	`StoO_selcnt := 0;`
	`StoO_error := SQLCODE;`
	`END;`

Interbase 4.0	Oracle7

	BEGIN
	DELETE emp
	WHERE sal < i_sal;
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error := SQLCODE;
	END;
	END;
	END LOOP;
	END;
	/

Comments

The converter can convert most WHILE constructs. However, the `CONTINUE` within a WHILE loop in Interbase 4.0 does not have a direct equivalent in PL/SQL. It is simulated using the GOTO statement with a label. Because the converter is a single-pass converter, it adds a label statement at the very beginning of every WHILE loop (see Example 2).

ASSIGNMENT Statement

Interbase 4.0	Oracle7

CREATE PROC proc1
AS
DECLARE VARIABLE x integer;
BEGIN
  x = -1;
  x= select sum(salary) FROM employee;
END;
	CREATE OR REPLACE PROCEDURE proc1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	i_x INTEGER;
	BEGIN
	  i_x := -1;
	  BEGIN
	  SELECT SUM(salary)
	  INTO i_x
	  FROM employee;
	  StoO_rowcnt := SQL%ROWCOUNT;
	  EXCEPTION
	  WHEN TOO_MANY_ROWS THEN
	  StoO_rowcnt := 2;
	  WHEN OTHERS THEN
	  StoO_rowcnt := 0;
	  StoO_selcnt := 0;
	  StoO_error := SQLCODE;
	  END;
	END;

Comments

PL/SQL has two ways to assign values to a variable:

1. Use the assignment statement to assign the value of a variable or an expression to a local variable.
2. Assign a value from a database using the ‘SELECT..INTO’ clause. This requires that the SQL returns only one row, or a null value is assigned to the variable.

   For example:

```
SELECT empno INTO i_empno
FROM employee
WHERE ename = ‘JOE RICHARDS’
```

SELECT Statement

Interbase 4.0	Oracle7

Example 1:
`CREATE PROC proc1 RETRUNS (test char(10))`
`AS`
`BEGIN`
`FOR`
`SELECT ename FROM employee into test;`
`DO`
`SUSPEND;`
`END;`
	Example 1:
	`CREATE OR REPLACE PROCEDURE proc1(`
	`i_oval1 OUT VARCHAR2)`
	`AS`
	`StoO_selcnt INTEGER;`
	`StoO_error INTEGER;`
	`StoO_rowcnt INTEGER;`
	`BEGIN`
	` BEGIN`
	` SELECT ename`
	` INTO i_oval1`
	` FROM employee;`
	` StoO_rowcnt := SQL%ROWCOUNT;`
	` EXCEPTION`
	` WHEN TOO_MANY_ROWS THEN`
	`  StoO_rowcnt := 2;`
	` WHEN OTHERS THEN`
	`  StoO_rowcnt := 0;`
	`  StoO_selcnt := 0;`
	`  StoO_error := SQLCODE;`
	` END;`
	`END;`

Example 2:
`CREATE PROC proc1`
`AS`
`DECLARE VARIABLE name char(20);`
`BEGIN`
`FOR`
`SELECT ename FROM employee into :name`
`DO`
` NULL;`
`END;`
	Example 2:
	`CREATE OR REPLACE PROCEDURE proc1`
	`AS`
	`StoO_selcnt INTEGER;`
	`StoO_error INTEGER;`
	`StoO_rowcnt INTEGER;`
	`i_name CHAR(20);`
	`BEGIN`
	`  BEGIN`
	`  SELECT  ename`
	`  INTO i_name`
	`  FROM employee;`
	`  StoO_rowcnt := SQL%ROWCOUNT;`
	`  EXCEPTION`
	`  WHEN TOO_MANY_ROWS THEN`
	`  StoO_rowcnt := 2;`
	`  WHEN OTHERS THEN`
	`  StoO_rowcnt := 0;`
	`  StoO_selcnt := 0;`

Interbase 4.0	Oracle7

	StoO_error := SQLCODE;
	END;
	IF StoO_rowcnt = 0 THEN
	RETURN /* 25022 */;
	END IF;
	END;
	
	OR
	
	CREATE OR REPLACE FUNCTION proc1
	RETURN INTEGER
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	i_name CHAR(20);
	BEGIN
	BEGIN
	SELECT ename
	INTO i_name
	FROM employee;
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN TOO_MANY_ROWS THEN
	StoO_rowcnt := 2;
	WHEN OTHERS THEN
	StoO_rowcnt := 0;
	StoO_selcnt := 0;
	StoO_error :=
	SQLCODE;
	END;
	IF StoO_rowcnt = 0 THEN
	RETURN 25022;
	END IF;
	END;

Comments

Because of the differences in the two architectures, Interbase 4.0 and Oracle7 stored procedures return data to the client program in different ways.

Both Interbase 4.0 and Oracle7 can pass data to the client using output parameters in the stored procedures. Interbase 4.0 uses another method known as suspend to transfer the data from the server to client.

APPENDIX

A LIST OF CONVERSION STEPS

This appendix lists the steps for migrating stored procedures from Interbase 4.0 to Oracle7. The tables state whether each step is required or optional and also gives an estimate of the time required to complete the step.

Steps to Migrate the Interbase 4.0 Stored Procedures to Oracle7

The following table lists the preparation steps for converting Interbase 4.0 stored procedures to Oracle7. The table also states whether each step is required or optional and gives an estimate of the time required to complete the step.

Interbase 4.0 Stored Procedure Conversion Preparation Steps	**Required? (Yes/No)**	**Fixed Days**	**Total Days**
Basic Oracle training (Introduction to Oracle)	Yes	5	5
Basic PLSQL training (Develop Applications Using Database Procedures)	Yes	3	3
Install and setup Oracle	Yes	2	2
Convert database schema	Yes	5	5
Install and setup converter	Yes	1	1
Basic converter training	Yes	1	1
Review Conversion Issues	Yes	1	1
• exception/error handling			
• fetching data			
• global variables			
• input/output parameters			
Total Preparation Days		**18**	**18**

The following table lists the remaining steps for converting Interbase stored procedures to Oracle7. The table also states whether each step is required or optional and gives an estimate of the time required to complete the step.

Interbase 4.0 Stored Procedure Conversion Steps	Req'd? (Y/N)	Converter Handling	Changes required before/after conversion?	Conv Rate (p/d)*	No. Procs	Fixed Days	Total Days
Pre-Conversion Steps (Syntactic Check)							
Replace user defined data types with base types		causes syntax error	before (write sed script)			0.5	0.5
Convert Interbase-specific types with Oracle types (binary, timestamp)		Converts 50% correct	before			0.5	0.5
Convert error handling logic		Parses - no translation	verify semantics before/after			0.25	0.25
Remove DDL statements		causes syntax error	before	50		0	
Total Pre-Conversion Days						**TDB**	
Procedure Conversion							
Separate stored procedures into individual files and group by type	Yes		before (use sed script)			1	1
Convert small INSERT, UPDATE and DELETE procedures		automatic				1	1
Convert medium INSERT, UPDATE, DELETE stored procedures		automatic				1	1
Convert large INSERT, UPDATE, DELETE stored		automatic				1	1

Interbase 4.0 Stored Procedure Conversion Steps	Req'd? (Y/N)	Converter Handling	Changes required before/after conversion?	Conv Rate (p/d)*	No. Procs	Fixed Days	Total Days
procedures							
Identify and convert small pocedures with nor FOR or SUSPEND logic		automatic				1	1
Identify and convert small pocedures with complex SUSPEND logic		automatic	convert with -b flag			1	1
Identify and convert small pocedures with FOR statement and single SUSPEND logic		automatic	convert with -c flag			1	1
Identify and convert small pocedures with FOR but no SUSPEND logic		automatic	convert with -f flag			1	1
Total Conversion Days						**TBD**	**TBD**
Post-Conversion Steps (Semantic Check) TO BE ADDED							
Total Post-Conversion Days						**3**	
Test Converted Procedures							
Write stored procedure testing program						5	5
Generate and load test data						3	3
Create small procedures in Oracle				100		0	
Create medium procedures in Oracle				50		0	
Create large procedures in Oracle				25		0	
Create SELECT procedures in Oracle				25		0	

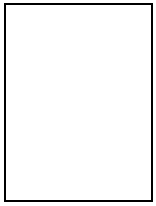
Interbase 4.0 Stored Procedure Conversion Steps	Req'd? (Y/N)	Converter Handling	Changes required before/after conversion?	Conv Rate (p/d)*	No. Procs	Fixed Days	Total Days
Test small procedures				100		0	
Test medium procedures				50		0	
Test large procedures				25		0	
Test select procedures				25		0	
Test client application				50		0	
Total Testing Days						**8**	

***** In the conversion rate, p/d stands for how many procedures can be converted per day. For example, if p/d equals 50 procedures/day and you have 100 procedures to convert, and the fixed days for that step is 0.25, then the estimated time for you to complete the step would be (100/50) + 0.25, or 2.25 days.

If no number is given for p/d, that means that the fixed days estimate for that step holds regardless of the number of procedures that you are converting.

The following table summarizes the total estimated fixed days required for the conversion effort. You can add the number of days estimated for your conversion project, based on the number of procedures that you need to convert.

Conversion Estimates	Fixed Days	Your Estimated Days
total preparation days	18	
total pre-conversion days	7.5	
total conversion days	6	
total post-conversion days	3	
total testing days	8	
Total Days	**42.5**	



Oracle Worldwide Alliances
Design and Migration Services

ORACLE®

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
415.506.7000
Fax: 415.506.7200
http://www.oracle.com/

Part #: CXXXXX