Oracle to MySQL Migration

Stored Procedures, Packages, Triggers, Scripts and Applications

White Paper

March 2009, Ispirer Systems Ltd.

Introduction

The purpose of this white paper is to describe the factors that impact migrating databases and applications from Oracle to MySQL. Cost and risk factors will be detailed, as well as tools and methodologies to help achieve a higher quality conversion.

It is very true that the Sun MySQL database can dramatically reduce the database Total Cost of Ownership (TCO) for a company by lowering license, hardware and administration costs. The largest risk in moving to the MySQL platform is the risk and complexity of migrating business logic from Oracle, particularly when existing applications make significant use of PL/SQL procedures, triggers, packages and Oracle specific SQL statements.

The migration from Oracle to MySQL can be troublesome, time consuming, and expensive. However, proven methodologies and tools can reduce the cost and time required and can significantly mitigate risk. With the help of the migration product SQLWays, a migration can be assessed, planned, and properly automated. With the proper use of automated tools and a strong project management process in place, companies can incur savings of over 70% compared to traditional manual migration techniques. Coupled with the savings from implementing MySQL, automated migration becomes a very attractive alternative.

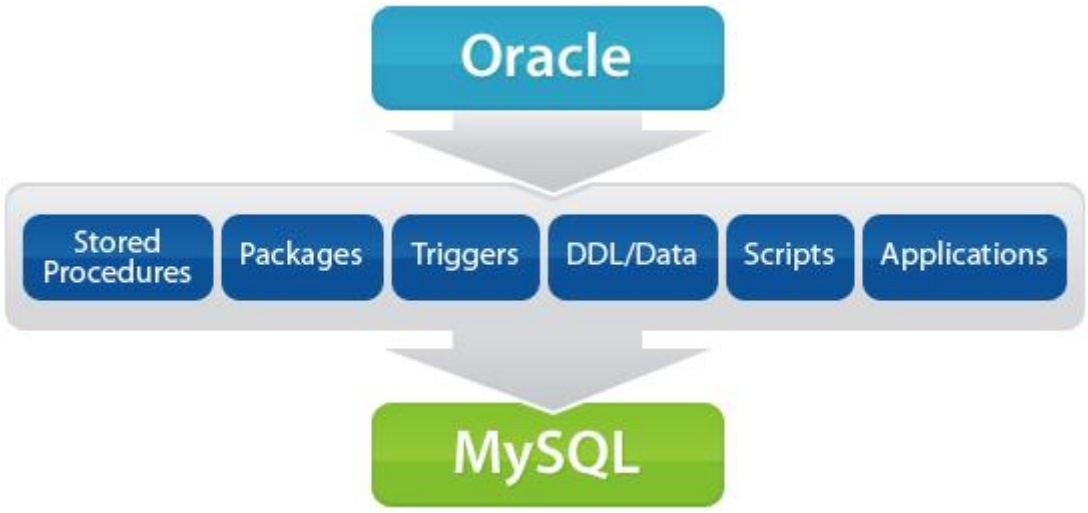
Challenges

The Oracle database provides very advanced capabilities to develop application logic that resides entirely inside the database using PL/SQL stored procedures, functions, packages and triggers.

Oracle PL/SQL is an easy-to-use and powerful procedural extension to SQL that is strongly recommended by Oracle for performance reasons. In most applications, use of PL/SQL naturally leads to a significantly large number of procedures, packages, and triggers. MySQL, although having some similar functionality, does not make use of PL/SQL.

Besides Oracle specific syntax, PL/SQL offers many non-ANSI compliant features, including features which are only found in Oracle. These Oracle specific features include:

- Packages - shared package variables, built-in packages
- %TYPE, %ROWTYPE, exceptions
- Object-oriented features: object types, functions, and collections
- Business intelligence and XML features etc



An Oracle to MySQL migration can be a very challenging process, particularly if Oracle specific features are in use, such as the ones described above.

However, such a migration could be relatively easy and risk-free. Such would be the case if the target database contained a relatively small amount of tables and simple business logic.

Since cost and risk may vary from project to project, it is important to fulfill a preliminary assessment.

Assessment

The purpose of the assessment is to define the scope, feasibility, cost and risk associated in migrating from an Oracle database to a MySQL based database implementation.

Database Assessment

Firstly, you need to define the types of database objects and how many of them you will need to migrate. Objects are items such as the following:

- Tables
- Views
- Procedures
- Functions
- Packages
- Triggers
- Sequences, synonyms etc.

If you need to convert PL/SQL code (procedures, packages, functions and triggers) or view/queries containing Oracle specific SQL syntax, you have to investigate what features are used and define the number of their occurrences. Examples of items that need to be accounted for are:

- Non-ANSI compatible SQL functions, operators and statements
- Results sets
- Cursor loops
- Exceptions
- Temp tables
- Object types and functions
- Collections

- Dynamic SQL
- Built-in packages
- OLAP functions
- XML functions etc.

After you have finished the examination, it is best to select MySQL equivalents or solutions to replace Oracle specific functionality. You can find typical solutions in subsequent chapters.

Application Assessment

Besides schema and server-side business logic conversion, you may also need to modify SQL statements within the application. It is vital to assess how much of this work will need to be done to complete the migration.

To start, you have to check what database API is used in your applications to access the Oracle database. It is important to note how many application source files contain Oracle specific code and therefore need to be modified to work with MySQL.

Most applications use a standard API like ODBC, JDBC, or ADO.NET to access Oracle, but some applications may use a native API like Oracle OCI or Pro*C/C++. Collecting all this information is imperative.

Even if you use a standard API, such as using ODBC/JDBC drivers, significant changes may need to be made to existing SQL statements. For example DECODE functions or legacy left outer join syntax (*) will need to be modified. It is recommended to estimate the number of native SQL statements.

If the application happens to use a native API like Oracle OCI, you will need to completely redesign the database access code to use the MySQL API or ODBC.

Assessment Tools

It is important to understand how much use is made of specific database features. How best is a 'feature use' assessment performed?

Start first by calculating the number of tables, procedures, views etc. like in the table below. For a more detailed analysis, you can use Ispirer's SQLWays product to collect comprehensive statistics.

The following is a sample assessment:

Database	**Number**
Tables	350
Views	280
Procedures	420
Functions	135
Triggers	50
Packages	10
Database Details	
BLOBs	37
Outer joins	155
Ref cursors	89
Exceptions	450
Temp tables	34

Application	
Java/JDBC	590 files
Outer joins	190
SQL functions	356
Result sets	47
Migration Approach	Automated conversion

Based on the assessment results, you can then develop the migration plan. If you have dozens of procedures, you might consider a manual conversion, but if hundreds or thousands of procedures need to be migrated, it is best to examine automated migration tools in the market. SQLWays provides such functionality.

Cost and Risk

The cost and risk associated with the conversion project depends on the scope of migration. It is important to note that the cost and risk are also impacted by the diversity and frequency of Oracle features in use in the database and application. The more Oracle features in use, the more complex and costly the conversion. Also, the more Oracle features in use, the more automated tools could help achieve success.

Data and DDL Migration Cost

Migration of data and DDL (Schema) objects are typically done very easily, as there are many tools on the market that can assist you with this kind of migration.

Typical Data and DDL migration involves conversion of

- Data types
- Constraints (primary and foreign keys, unique constraints, NULL, defaults etc)
- Data transfer
- Indexes

Although there are syntax differences in Oracle and MySQL DDL statements, both have similar data types (character, number, date, time, LOBs) and allow you specifying similar integrity constraints.

Sample DDL/Data migration estimation:

Database	
Tables	<100 tables
LOBs	10 columns
Max table rows	<10M
Max table size	<300 Mb
Migration Process	
Assessment	2-8 hrs
MySQL configuration	4-16 hrs
Automated transfer	2-4 hrs
Testing, configuration change, next iteration	4-12 hrs
Total time	12-40 hrs

Tools	Free, or less than $500

Due to automation, the cost of migrating DDL and data is not directly proportional to the number of tables and data size. For example, the cost of migration for 100 and 300 tables can be similar in the cost, if the tables have similar structure and data size.

When the number of tables and their size increase, you may need to spend more time to properly configure the MySQL database, tune data transfer, and focus on things like index creation performance.

Risk Mitigation for Typical DDL and Data Migration

Typical DDL/Data migration imposes relatively low risk. Using SQLWays, it is possible to run the full database transfer in evaluation mode, review data, and run applications connected to the new MySQL database:

This is general workflow:

1. Run full database transfer in evaluation mode
2. Check transfer errors, compare tables structures, the number of rows in Oracle and MySQL
3. Review and test data in all or representative tables using SQL tools such as Oracle SQL*Plus, MySQL Query Browser, or the mysql command line utility.
4. Run and test the target application connected to MySQL

Challenging Data Migrations

Although, in general, data/DDL migration is relatively easy compared with business logic conversion, there are some conditions that commonly increase the complexity of a data/DDL migration:

- **Large volumes of data**

If you need to migrate large volumes of data, you may need more effort to configure MySQL servers.

A large amount of data may impact the migration process, particularly in terms of the time it takes to complete the migration. In order to mitigate the time required to complete the migration, you might execute the migration in a concurrent fashion. This increases the complexity of the migration.

Also, the transfer of large volume of data can complicate error handling, as you now cannot afford to re-run a full migration if just a few tables fail.

The project might benefit from tools that allow for bulk insert options, as tools that issue a commit after each row may be no longer be a viable option.

- **Minimal downtime**

In some mission-critical environments, you have to ensure downtime is kept to a minimum. To meet these requirements, you must properly design the migration process to do things such as run data transfer concurrently, or transfer static tables out of the downtime window. Sometimes replication tools will need to be used to reduce downtime.

- **Strict performance requirements**

Some environments have very strict performance requirements for applications. When migrating to MySQL, it is imperative that existing performance is maintained or improved. This requires you to spend more time on database design and tuning, as well as performing test migrations in order to test post migration performance.

Risk Mitigation for Challenging DDL and Data Migration

Challenging data migrations cannot be completed with a simple double-click. A Proof-of-Concept migration needs to be executed in order to ensure requirements can be met.

Most commonly, the following migration process is recommended for complex data migration projects:

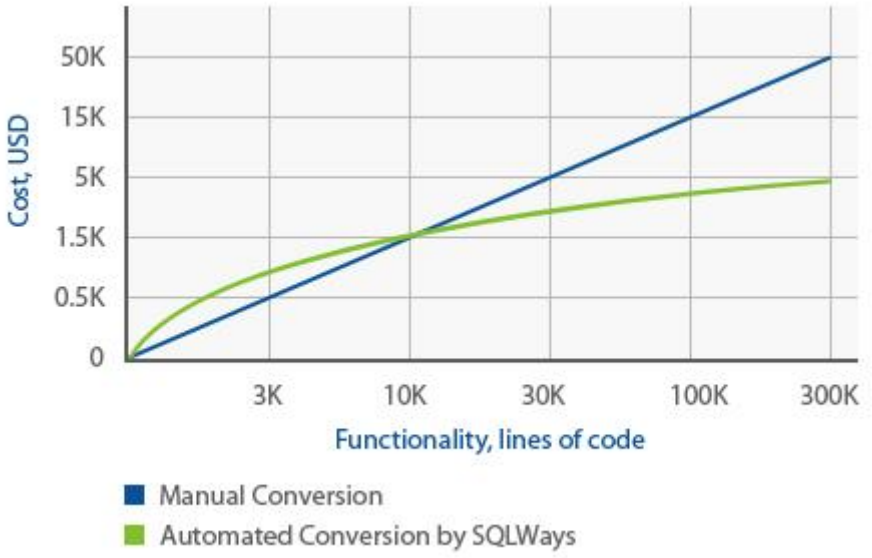
- Proof-of-Concept migration to check feasibility of requirements
- Test migration to fully emulate production migration and run comprehensive testing
- Production migration

Cost of Business Logic Conversion

If your database contains a dozen procedures and triggers, it is easy to manually rewrite them to MySQL SP syntax. But if you have thousands of procedures and triggers, manual conversion is quite expensive. You have to consider how an automated tool can assist you.

The cost of manual conversion is directly proportional to the number of lines of code you need to convert. On the other hand, automated tools can limit the cost, and make the migration of even a million lines of code very reasonable in cost and effort.

Depending on the lines of code to be converted, automated conversion of business logic using a tool like SQLWays can cost 7-10 times less than manual conversion.



The diversity and frequency of Oracle specific features defines the complexity of business logic migration and level of automation that tools can provide.

For effective automation, our experts feel that a migration tool such as SQLWays must be able to convert over 95% of the business logic.

A sample server side business logic migration project estimation looks as follows:

Database	
Stored procedures	1000
Triggers	300
Functions	250
Packages	10 (50 procedure per package)
Manual conversion	5,000 hrs (~30 man-months)
Labor cost	$50,000-$250,000 (depending on country)
Automated conversion	
Assessment, solutions discussion	16-40 hrs
Iterative conversion, analysis	40-80 hrs
Testing	40-80 hrs
Total time	96-200 hrs
Tools cost	less than $5,000-$10,000

If you compare DDL/Data and business logic migration, you can see that the latter can make up to 95% of the total project cost. This is especially true in large Oracle to MySQL migration projects.

Mitigating Risk for Business Logic Conversion

If there are many lines of code to convert, and a wide diversity of database features from Oracle are used, the conversion can impose significant risk, so you need to take several important steps to mitigate it.

- **Experience**

The staff responsible for the migration project should have developer and administrative skills and experience both for both Oracle and MySQL databases. They need to clearly understand the scope, challenges, tasks and stages to implement migration successfully.

- **Comprehensive Assessment**

At the initial stage, you have to perform a comprehensive assessment of the databases you want to migrate. As a result, you will know what specific functionality you need to convert, what solutions you will use to replace non-ANSI compliant Oracle features.

You need to determine if there is a solution for every feature in use. Some Oracle features are not easy to map to a similar equivalent in MySQL, so you may need to redesign some functionality.

- **Proof-of-Concept on Full Code Base**

Automated tools like SQLWays make it possible to easily run conversions on the full code base at the beginning of migration evaluation. We suggest doing this as part of any complex migration, as this will help expose potential bottlenecks and better clarify "the percent of automation " or success factor the migration tool can provide.

More important, it will make you confident that conversion of heavy PL/SQL code is feasible at low cost.

- **Use Automated Migration as much as possible**

In addition to its high cost, manual migration reduces the visibility of bottlenecks at early stages, which may result in the need to redesign the migration. This then increases the effort and the cost of the migration even more.

By comparison, automated tools allow the conversion to be executed repeatedly for low cost, but with high levels of feedback. This allows for highly refined and tuned migrations without significant cost penalties.

In general, manual conversion is a tedious task that leads to a high probability of human error. Very often developers can often produce different conversion results for similar code. As a result, this leads to large cost and time problems for testing.

- **Early Testing**

Testing at early stages should be used to minimize the project risk. You can run unit tests, or perform code reviews even when application level functional testing is not possible yet.

You can use features of automated tools that can generate test cases to invoke procedures and functions with specific values and compare the results. Please note that this cannot replace functional testing at the application level, but it can help discover many potential issues.

Application Conversion

Besides server-side business logic conversion, in most cases you need to modify your applications to work with MySQL.

There can be non-ANSI SQL statements in Java or PowerBuilder applications, i.e. syntax that differs from MySQL SQL syntax and needs to be modified.

Specifically, the most typical syntax features that need attention for Oracle to MySQL conversions are Oracle left outer join (+) syntax scripts. Functions like DECODE, NVL, and SYSDATE all will need attention.

You cannot just replace function names using Find/Replace in these situations. In many cases, functions can have different parameter syntax, or require SQL statement changes such as the left outer join.

Moreover, simple string replacements can change the text in unintended places, such as character strings, or Java language statements.

The better approach is to use a tool like SQLWays which is capable of automatically modifying application code and converting SQL statements to proper MySQL syntax.

Such tools can correctly identify SQL statements in code, perform the conversion, and generate reports about every change, greatly simplifying the application conversion task.

Planning – Migration Stages

Proper planning is very important for a successful migration, and typical migration stages are as follows:

- **Assessment**

Assessment (which is described previously in this document) is intended to analyze the databases and application you need to migrate, define the scope of migration, and document any Oracle specific functionality that will need to be mapped to MySQL.

Based on the information gathered by the assessment, you can define what approaches should be used (manual or automated conversion), and the cost and risks that are associated with the migration.

- **Full Conversion at the Early Stage – Proof-of-Concept**

Assume you have a database with 2,000 procedures. You can run SQLWays to convert the entire code base during proof-of-concept stages. This is suggested even if you decide to test and deploy module by module.

Very early in the process (when automation tools are used), feedback and visibility become available regarding the migration. This is in direct contrast to a manual migration where many man hours may often be spent on tasks before realizing the migration process has encountered a pitfall and must backtrack.

You can apply a more integrated and uniform migration approach by using automated solutions like SQLWays. Very often, as manual migration tasks are spread among different people within an organization, different procedures, approaches are often used for the same syntax. As a result, the more uniform and integrated a migration results are, the easier it will be to test and modify.

Ideally, you need to achieve nearly 100% error-free object creation in MySQL at an early stage. This means that all tables, functions, procedures, trigger are successfully created in MySQL.

Since 100% conversion is quite hard to achieve for any databases in the current release of any conversion tool, the Ispirer team offers free customization (1-2 days per fix) to achieve nearly 100% automation during initial evaluation.

- **Run-Time, Logical and Performance Testing**

Migration is often deployed module-by-module. After you have converted server-side business logic, even before applications are converted and application level testing is possible, you should test the database conversion.

You can select several representative or most critical procedures and perform a code review. Of course, you cannot find all issues reviewing the code, but at the early stages it is very useful. With the code, you can review how solutions are applied, and estimate the conversion quality in general.

It is best to create a list of feature conversion of which you want to study in depth.

Even if you can create a procedure or function in the database, it does not mean that it does not contain any critical errors. Many errors can be quickly discovered by executing the procedures. An easy and effective way to test procedures is to generate test cases.

SQLWays can generate a set of procedure calls with different input parameters. By examining code, SQLWays can find out what values, string and date constants, control-flow conditions are used, and generate reasonable test cases in many situations.

To perform more comprehensive logic and performance testing, you can develop test scripts with real data, implementing various scenarios.

If you use automated quality assurance software for your Oracle database and applications, you can consider updating them to run against MySQL and to ensure comprehensive migration testing.

Typical Conversion Solutions - Samples

Although conversion tasks and solutions vary from project to project, many of them are typical for any migration.

Note. All the conversions described below are done by SQLWays automatically.

DDL

Both Oracle and MySQL support the CREATE TABLE statement, but there are many syntax differences.

- **Data Types**

Oracle	MySQL

CREATE TABLE employees
(
  id **NUMBER(5)**,
  name **VARCHAR2(120)**,
  hire_date **DATE**,
  salary **NUMBER(7)**,
  dept_id **NUMBER(2)**
);
	CREATE TABLE employees
	(
	  id **INT**,
	  name **VARCHAR(120)**,
	  hire_date **DATETIME**,
	  salary **INT**,
	  dept_id **TINYINT**
	);

- **Reserved Words**

Oracle and MySQL use different sets of the reserved words, so some column names now have to be quoted in MySQL queries.

Oracle	MySQL
SELECT product_id, **limit** **FROM** product_data;	**SELECT** product_id, **`limit`** **FROM** product_data;

Queries and PL/SQL Code

You need to modify SQL statements mostly to change the syntax of functions and expressions. PL/SQL needs to be completely converted to MySQL SQL procedural syntax.

- **Outer Joins**

Oracle supports specific syntax for outer joins that is widely used in old applications.

Oracle	MySQL
SELECT e.name, d.name **FROM** employees e, departments d **WHERE** e.dept_id = d.id**(+)**;	**SELECT** e.name, d.name **FROM** employees e **LEFT OUTER JOIN** departments d **ON** e.dept_id = d.id;

- **Assigning ID value to column**

Oracle does not support auto-increment (identity) columns, and a sequence object is used to assign new ID values from an application or a trigger.

Although a single sequence object can be used to assign values for multiple tables in Oracle, in many cases it is used for one table only and this functionality can be converted to auto-increment column in MySQL.

For automated conversion, SQLWays inspects SQL queries and INSERT statements in applications, procedures and triggers to identify ID assignment and convert them to auto-increment columns in MySQL.

Oracle	MySQL

CREATE TABLE employees
(
id NUMBER(5) **PRIMARY KEY**,
name VARCHAR2(120),
hire_date DATE,
dept_id NUMBER(2)
);
 	
CREATE TRIGGER emp_id **BEFORE INSERT ON** employees
FOR EACH ROW
BEGIN
SELECT emp_id_seq.nextval
INTO :new.id FROM dual;
END;
	CREATE TABLE employees
	(
	id INT **AUTO_INCREMENT PRIMARY KEY**,
	name VARCHAR(120),
	hire_date DATETIME,
	dept_id TINYINT
	);
	
	-- Trigger is no required anymore

- **Multiple triggers on one event**

In Oracle, for the same table you can define several triggers for the same event (for example, several triggers on INSERT event for employees table).

This is not permissible in MySQL, as you have to put all code for one event in the same trigger.

- **Packages and Shared Variables**

In Oracle, a package is a group of related procedures and functions that can share variables. Package procedure and function have to be converted to standalone objects in MySQL.

Package variables can be modified in one package procedure. Furthermore, another package procedure can see or relay the updated value. To replace this functionality in MySQL, you can use session variables that start with @ sign.

Oracle	MySQL

CREATE PACKAGE BODY emp_pack
AS
 	
processed NUMBER **DEFAULT 0**;
 	
PROCEDURE new_employee **AS**
BEGIN
…
processed := processed + 1;
END;
 	
PROCEDURE raise_salary **AS**
BEGIN
…
processed := processed + 1;
END;
END;
	CREATE PROCEDURE new_employee
	BEGIN
	…
	IF @processed **IS NULL**
	THEN @processed = 0;
	
	@processed = @processed + 1;
	END;
	
	CREATE PROCEDURE raise_salary
	BEGIN
	…
	
	IF @processed **IS NULL**
	THEN @processed = 0;
	
	@processed := @processed + 1;
	END;
	END;

- **Returning results sets**

You have to use cursor variables (REF CURSORs) as OUT parameter to return a result set from Oracle. In many cases, this can be converted to a simple SELECT in MySQL.

Oracle	MySQL

CREATE PROCEDURE get_salaries
(d_id IN NUMBER, **cur OUT SYS_REFCURSOR**)
AS
BEGIN
 	
OPEN cur FOR
SELECT id, name, salary
FROM employees
WHERE dept_id = d_id
ORDER BY name;
 	
END;
	CREATE PROCEDURE get_salaries (IN d_id INT)
	BEGIN
	
	SELECT id, name, salary
	FROM employees
	WHERE dept_id = d_id
	ORDER BY name;
	
	END;

- **%TYPE and %ROWTYPE data type definitions**

Oracle %TYPE attribute allows you to define data types for PL/SQL variables based on the types of table columns. In MySQL, you have to specify the data type explicitly.

In a similar manner, %ROWTYPE attribute allows you creating record variables based on table rows. In MySQL, you have to create standalone variables and specify their data types explicitly.

Oracle	MySQL
v_emp_name employees.name**%TYPE**; v_emp_rec employees**%ROWTYPE**;	v_emp_name **VARCHAR(120)** v_ emp_id **INT** v_ emp_name **VARCHAR(120)** v_ emp_hire_date **DATETIME** v_ emp_salary **INT** v_ emp_dept_id **TINYINT**

- **SQL conversion in Java applications**

In Java applications, you may need to change syntax of SQL statements.

Oracle	MySQL

…
PreparedStatement ps = null;
ResultSet rs = null;
String sql = "SELECT e.name, d.name" +
"FROM employees e, departments d" +
"WHERE e.dept_id = d.id**(+)**";
 	
ps = conn.prepareStatement(sql);
rs = ps.executeQuery();
…
	…
	PreparedStatement ps = null;
	ResultSet rs = null;
	String sql = "SELECT e.name, d.name" +
	"FROM employees e **LEFT OUTER JOIN**" +
	"departments d **ON** e.dept_id = d.id";
	
	ps = conn.prepareStatement(sql);
	rs = ps.executeQuery();
	…

- **SQL conversion in PowerBuilder applications**

In PowerBuilder applications, you may also need to change syntax of SQL statements.

Oracle	MySQL
datawindow(units=0 processing=0 print.orientation = 0 … print.preview.buttons=no) table(column=(type=char(120) updatewhereclause=yes name=e_name dbname="employees.name") column=(type=char(120) updatewhereclause=yes name=d_name dbname="departments.name") **retrieve**="SELECT e.name, d.name FROM employees e, departments d WHERE e.dept_id = d.id**(+)**")	datawindow(units=0 processing=0 print.orientation = 0 … print.preview.buttons=no) table(column=(type=char(120) updatewhereclause=yes name=e_name dbname="employees.name") column=(type=char(120) updatewhereclause=yes name=d_name dbname="departments.name") **retrieve**=" SELECT e.name, d.name FROM employees e **LEFT OUTER JOIN** departments d **ON** e.dept_id = d.id")

Workarounds for Unsupported Functionality

There are many features in Oracle PL/SQL that are currently not supported by MySQL SQL procedural language. If this functionality is used in the source database, you need to apply various workarounds for get the same behavior in MySQL. Here are some specific examples:

- **PL/SQL Collections**

You can use temporary tables and SQL DML operations (SELECT, INSERT, UPDATE, DELETE) to replace this feature in MySQL.

- **RAISE_APPLICATION_ERROR**

You can use an UDF to raise an error from MySQL stored procedures.

- **UTL_FILE Built-in Package**

You can use an UDF to work with files from MySQL stored procedures.

- **Complex Business Logic**

As a general solution, complex PL/SQL business logic can be converted to Java language.

Conclusion

Automating migration to the MySQL licensing model provides an incredible value. Using SQLWays from Ispirer on complex Oracle to MySQL migration projects increases quality, saving you both time and money. There are many things to keep in mind when planning on migrating business logic and database contents for an existing application. Proper planning, analysis, and attention to detail are needed for each stage of a migration project.

Although complex database migration from Oracle to MySQL that involves business logic conversion is a challenging task, proper approach and use of migration tools allow running migrations at low cost and risk. Ispirer's SQLWays product and Ispirer services can provide a large amount of value when dealing with complex business logic conversion.