A Guide for Migrating From Oracle to MySQL

A MySQL® Technical White Paper

March 2006

Table of Contents

1 Introduction

With the rapid growth of MySQL in the database market, many corporations, government agencies, educational institutions, and others have begun to migrate away from their expensive and proprietary databases. Of course, a migration from any database is not something to be taken lightly, and so countless organizations are considering their options for migrating to MySQL.

In particular, many MySQL customers are migrating from Oracle because they have reached the conclusion that the combination of cost-savings and feature set of MySQL make for a compelling business case to offload some or all their database-driven applications to the MySQL database server. This paper provides insight into what is needed for considering a move from Oracle to MySQL, and presents a number of options that help make the transition easy. Both the business and technical sides of migrating to MySQL will be dealt with, so whether you are a manager or a seasoned DBA, you will find the needed answers to questions that revolve around migrating to the world's most popular open source database - MySQL.

2 Why Migrating from Oracle makes Good Business Sense

Before making any large-scale commitment to a technology, modern enterprises must look beyond the technical promises software vendors make and first consider the business side of acquiring a technology. While each business follows its own methodology for approving the deployment of a given piece of software, the core set of factors that normally govern acceptance are calculating the total cost of ownership and validating the viability of the software vendor.

Let's take a more detailed look at each of the two factors above and see how MySQL measures up when it comes to calculating the total cost of ownership of the MySQL database server, as well as the current health of the company that stands behind that database.

2.1 Measuring the Costs of using Oracle vs. MySQL

One of the driving factors that can be attributed to the huge popularity and adoption of open source software in business today is the dramatic cost savings that accompanies owning such software. Given that many large Global 2000 enterprises spend between $500,000 and $10,000,000 in annual costs for new licenses and existing maintenance for proprietary database software, MySQL becomes an incredibly attractive alternative as most companies find they can slash their costs some 80-90% by using MySQL for new application projects as well as upgrading existing systems to use MySQL in place of its more expensive and proprietary alternatives.

Such license and maintenance cost savings are amplified when businesses find they must scale their applications to meet higher user demand. For example, a recently published study by database industry guru Charlie Garry compared the cost of deploying Oracle RAC and MySQL and offered the following insight:

> "Oracle has encouraged its customers to scale out through its RAC approach.
> Although there are technical pros and cons to using RAC, one important consideration is the additional cost. Since RAC is an additionally charged item, and since the majority of RAC users also purchase the data partitioning feature to help with scaling, the cost per additional processor is $85,400 including license, support and mandatory upgrade fees. To put that in context, for every

processor added you could purchase approximately 11 commodity servers, install an open source database such as MySQL and purchase gold level database support for each server."[1]

2.2 Scaling Out vs. Scaling Up

Countless MySQL customers have become successful in using the modern scale out approach to database infrastructure design vs. the older scale up methodology. As the previous quote from Charlie Garry makes clear, from a cost standpoint, scaling out with Oracle doesn't make much financial sense, even when the RAC option is not used. Consider the following comparison and cost-savings that can occur when using scale-out with MySQL vs. scale up with Oracle:

	Scale Up	**Scale Out**
Hardware Cost	SMP Server: 1 CPUs Per server: 24 Total CPUs: 24 **$1,250,000**	Dell Server: 12 CPUs Per Server: 2 Total CPUs: 24 **$120,000**
Software Cost	Oracle Enterprise Edition Price: $40,000/CPU Up-front License Cost: **$960,000** Annual Support Cost: **$211,200**	MySQL Network Gold Price: $2,995/Server/Year Up-front License Cost: **$0** Annual Network Cost: **$36,000**
Total Cost	**$2,421,200**	**$156,000**
Total Savings	**$2,265,200**	
Total Software Savings	**$1,135,200**	

Such black-and-white cost comparisons are why so many are turning to scale-out architectures to grow their business. Hardware costs aside, scaling out with MySQL makes complete business sense as the above example shows that software costs can be reduced nearly 97% by going with MySQL instead of Oracle. Companies like Weather Channel find MySQL the perfect weapon to use in their quest for high database performance and lowest total cost of ownership:

"Today, Weather.com is the 10th-largest website in the world and runs almost 100 percent on the open-source database MySQL. 'With the switch from a Sun Solaris environment to Linux on Intel, elimination of hardware maintenance, replacement of commercial software with open source, and the better price and performance of an Intel platform, we reduced costs by one-third and increased website processing capacity by 30 percent,' reports Dan Agronow, CTO of The Weather Channel Interactive."[2]

2.3 A Quick Look at MySQL AB

Migrating to a new database platform generally means making a long term commitment, so modern enterprises considering the move to MySQL want to know the history behind the database as well as things like who's using it and how.

[1] Database Scale-Out Using Open Source Software:Achieving Infrastructure Agility, September 2005.
[2] http://www.cio.com/archive/090105/forum.html, September 2005

According to industry surveys, the number one concern of enterprises considering the adoption of open source software is lack of support[3]. Understandably, no major company is going to bet its critical business systems on software that isn't backed by a reputable vendor – one that has a long-standing and excellent reputation in the industry, and who offers first-class support.

Many who first happen upon MySQL are surprised to learn that MySQL AB has been in business for over ten years. With operations in over 22 countries and eight million installations, MySQL AB continues to expand and grow as the world's most popular open source database. Every day, over 50,000 people across the world download MySQL software to power their business applications, which helps explain why MySQL has been able to double its revenues every year for the past three years.

In 2006, the popularity of MySQL continues to grow with Forrester Research finding in a recent survey that 74% planned to use open source in 2006 and 53% planned to use MySQL as one of their database platforms.

Due to an ever-increasing enterprise customer base, MySQL now offers MySQL Network for customers who desire top-notch enterprise grade support and services. For enterprise customers, MySQL Network provides enterprise certified software, around-the-clock support, software advisors, indemnification and much more. World-class IT system and solution vendors have recognized the popularity of MySQL Network and have now partnered with MySQL to resell MySQL Network to their large customer base. Such MySQL partner resellers now include HP, Novell, Dell, and others.

For those wishing references and validation that MySQL can be depended upon for critical application needs, one need only look at MySQL's rapidly growing customer base, which reads like a Who's-Who in the business community. Modern enterprises such as Yahoo, Google, Bank of America, Weather Channel, Sabre Holdings, Amazon, NASA, and many others across all business industries can be found in MySQL's customer rolodex. For more information, see http://www.mysql.com/customers/.

3 The Technical Case for Migrating from Oracle to MySQL

While MySQL passes the business case litmus test with flying colors, the next hurdle to overcome with respect to the question of whether to migrate to MySQL from Oracle is the technical review of the database feature set and capabilities. Just exactly how good can a open source database be?

3.1 All the Right Features

No one disputes the fact that Oracle has more bells and whistles than the MySQL database server. But then, MySQL is not designed to be like Oracle, but instead was created to be a high-performance database server that contains the majority of features needed by modern enterprises to run their business. Many MySQL customers use both Oracle and MySQL in the same data center and use each where it makes the most sense.

In terms of core features, DBAs and CIO's alike will be pleased to find that MySQL contains all the necessary capabilities to run the vast majority of their application needs:

[3] Forrester Research, February 2004.

Database Feature	Available in MySQL
Open Source	√
Pluggable Storage Engine Architecture (MyISAM, InnoDB, Merge, Memory, Archive, Cluster)	√
High-Availability Clustered Database	√
ANSI SQL, SubQueries, Joins, Cursors, Prepared Statements	√
Stored Procedures, Triggers, SQL and User-Defined Functions	√
Updateable Views	√
ACID Transactions, Commit, Rollback	√
Distributed Transactions	√
Row-level Locking	√
Snapshot/Consistent Repeatable Reads (readers don't block writers and vice-versa)	√
Server-enforced Referential Integrity	√
Strong Data type support (Numeric, VARCHAR, BLOB, etc)	√
High-Precision Numeric Data types	√
Robust Indexing (clustered, b-tree, hash, full-text)	√
Dynamic Memory Caches	√
Unique Query Cache	√
Cost-Based Optimizer	√
Unicode, UTF-8	√
XML, XPath	√
Geospatial support	√
Replication (Row-based and Statement-based)	√
Partitioning (Range, List, Hash, Key, Composite)	√
VLDB (terabytes) capable	√
High-speed, parallel data load utility	√
Online Backup with Point-in-Time Recovery	√
Automatic Restart/Crash Recovery	√
Automatic Storage Management (auto-expansion, undo management)	√
Compressed and Archive Tables	√
Information Schema	√
Robust Security (SSL, fine grained object privileges)	√
Built-in data encryption and decryption	√
Built-in Event Scheduler	√
Drivers (ODBC, JDBC, .NET, PHP, etc)	√
Tools (Workbench, Administrator, Query Browser, Migration Toolkit)	√
Multiple Platforms (RedHat, SuSE, Fedora, Solaris, HPUS, AIX, SCO, FreeBSD, Mac OS, Windows)	√

Table 1 – Core feature set of the MySQL database server

Unlike Oracle, DBAs and developers will find that they can download, completely install, and be up and running with the MySQL software in under 15 minutes. Once installed, a database administrator or developer will find that the adherence to ANSI SQL standards means that they will be able to be immediately productive with MySQL. Users of MySQL will also enjoy a graphical set of management and development tools, available on Windows, Linux, and Mac, that both increases productivity and simplifies database interaction.

3.2 Examining MySQL Performance and Scalability

One of MySQL's hallmarks has been outstanding performance in all areas of database activity, whether that be transaction processing, data warehousing, or high-traffic Web sites. As has already been mentioned, many modern enterprises are realizing incredible performance gains by utilizing a scale-out architecture design coupled with MySQL, where data is replicated across many MySQL commodity servers and load balanced so even the largest end user load can be served with tremendously fast response times.

Customers such as Friendster serve up over 1.5 billion queries a day across 24TB of data with MySQL. Other customers like Craigslist and Sabre Holdings serve countless customer requests each day with MySQL. On the data warehouse front, customers such as Los Alamos Labs who manages 7TB of data with MySQL, and Cox Communications, who experiences 4 million inserts of new customer data every 2 hours, say that MySQL is more than able when it comes to performance and scale. As Alan Walker, a VP of Sabre said, "MySQL ran faster or as fast as any commercial database we tested."

3.3 MySQL – Always Up, Always On

Reliability and contribution to system uptime are two key technical requirements for any piece of software, but especially for database software. MySQL doesn't disappoint in this area and offers many features that help ensure continuous database operation.

A standalone MySQL installation is highly reliable with many MySQL installations experiencing little or no downtime over several years. As was previously discussed, many large-scale enterprises use MySQL replication to implement scale-out architectures that provide very high levels of redundancy, with the end result being completely uninterrupted operations over dozens and even hundreds of servers, which oftentimes eliminates the need for something like Oracle RAC.

For those needing the highest possible levels of uptime, MySQL Cluster can be utilized to provide "5 9's" high availability solutions. Using a shared-nothing architecture, MySQL Cluster is used by customers who simply cannot afford for their systems to be offline at any time. MySQL Cluster is particularly well suited to applications such as telecommunications and eCommerce transactional and catalog systems.

4 Practical Suggestions for Easy Migration from Oracle to MySQL

The figure below outlines the three basic steps to migrate from Oracle to MySQL:

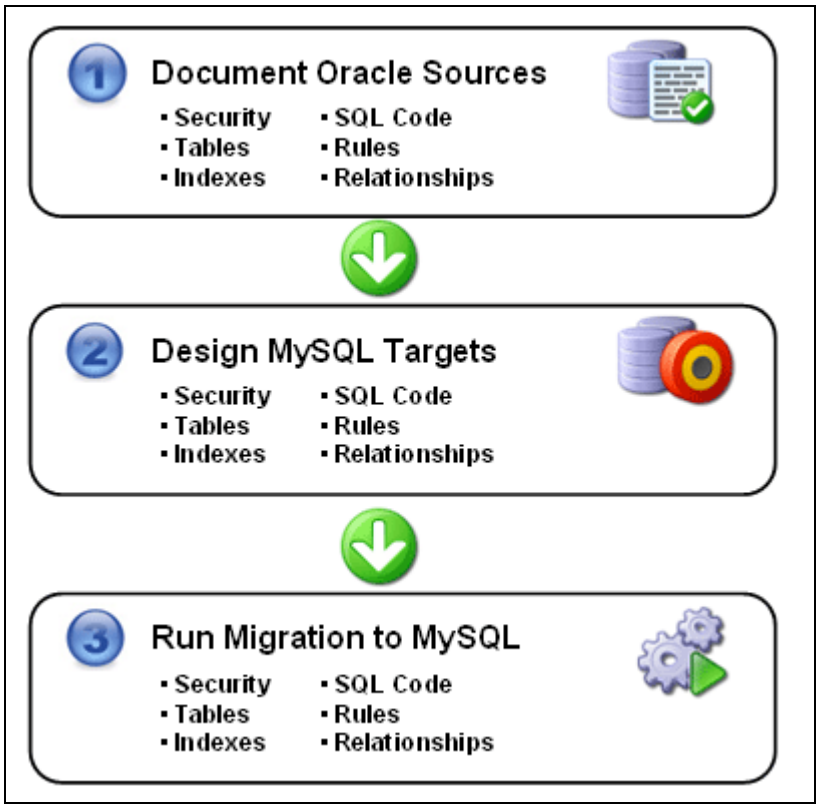


Figure 1 - Three Basic Steps to Migrate from Oracle to MySQL

4.1 All or Nothing?

Before working through each migration step, it is helpful to first point out that one very successful MySQL deployment strategy that customers have used is to move forward with MySQL for new development or migrate existing applications where the need for esoteric Oracle features is not necessary, and use Oracle when very specialized database needs must be met. As Table 1 in this paper has demonstrated, MySQL offers the perfect balance of ease-of-use and a strong enterprise feature set for handling the vast majority of any application's database requirements.

But MySQL can easily coexist in any IT infrastructure alongside Oracle or any other database platform because its tremendous ease of use means little to no additional management overhead. Just one customer example of this is Sabre that utilizes MySQL for its reservation lookup portion of its core application, but utilizes HP Nonstop for its heavy transaction processing needs.

4.2 Step 1 – Document Oracle Sources

Documenting an existing Oracle database structure can be challenging if it is attempted by standard manual processes. While Oracle has a very good metadata dictionary to work with, the process of manually pulling all metadata (table, column, index, etc.) can be a very time consuming event.

The best approach is to use a computerized reverse engineering process that automatically catalogs all applicable metadata that is needed for conversion. A good third party data modeling tool can be used for this, and there are a number of good products on the market, such as Sybase's/Quest's Powerdesigner and Embarcadero's ER/Studio, that support the reverse engineering of multiple datasources such as Oracle. MySQL also provides a free modeling tool called MySQL Workbench that can reverse engineer Oracle table and index structures.

Moving data and index structures over to MySQL isn't typically a challenging task as MySQL supports all the important datatypes, table designs, and index structures. For a complete description of datatype comparisons and other like things of Oracle and MySQL, please see Appendix A-C of this paper, and see Appendix D for a step-by-step example migration of an Oracle schema to MySQL.

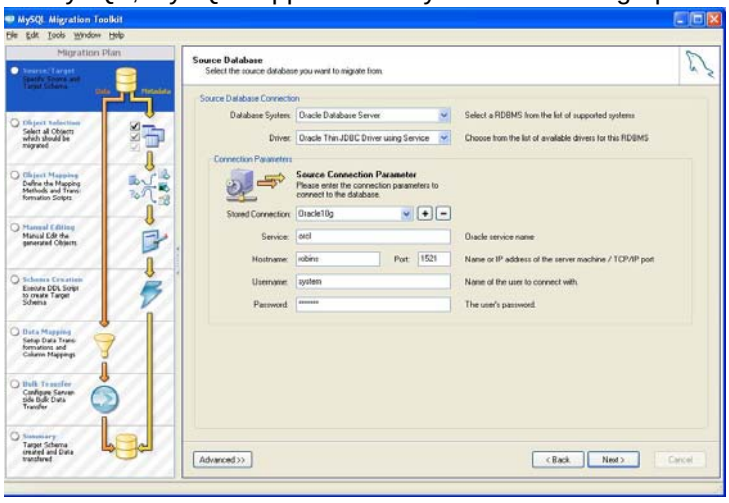
In brief, a few Oracle data-related objects that cannot be migrated in a one-to-one move include:

- TYPEs
- Materialized Views
- Dimensions
- Sequences
- Bitmap indexes
- Global partitioned indexes

The more challenging aspect of migrations is dealing with code objects. Oracle's PL/SQL language sports many features, many of which are not ANSI standard, and so a careful eye must be used when looking at stored procedures, triggers, views, user-defined functions, and the like. Besides general syntax functions and feature differences, the following items require special attention before they can be migrated completely from Oracle to MySQL:

- Packages (although individual procedures may be)
- Java stored procedures
- DDL and statement-based triggers (MySQL has row-based triggers)
- Proprietary Oracle function calls (DBMS_OUTPUT, etc.)
- Certain cases of dynamic SQL
- Overloaded programs/sub-programs
- Oracle PRAGMA's
- Object Reference Functions (DEREF, etc.)

To help migrate data objects from Oracle to MySQL, MySQL supplies a freely downloadable graphical Migration Toolkit. The tool supplies a reverse engineering component exactly like those found in the best data modeling tools. A user first supplies the connection details to the source database system through an easy to use interface and then connects to and reverse engineers the Oracle schemas targeted for migration.



With the MySQL Migration Toolkit, obtaining the necessary source database metadata is a very easy process and helps quickly jump start any migration effort.

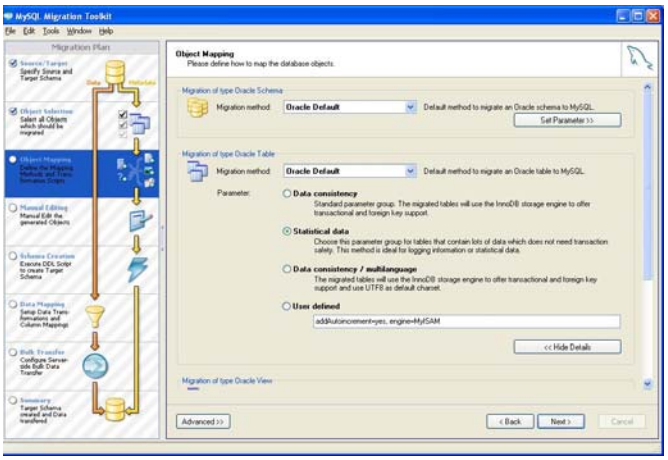
4.3 Step 2 – Design MySQL Targets

Once the Oracle source metadata has been obtained and digested, the next step is to design the MySQL target database. This basically involves translating the source objects and their properties (such as column datatypes) to MySQL complements. As one can imagine, this step can be extremely time consuming and error-prone if attempted by hand as most databases will have thousands of object properties that must be converted. Again, for a detailed listing of Oracle to MySQL transformation datatypes and functions, please see Appendix A of this paper.

Note that many data modeling tools have the ability to convert an Oracle schema to a MySQL schema with only a few mouse clicks. The MySQL Workbench product does the same thing automatically whenever an Oracle database is reverse-engineered by the tool. Naturally, the models can be tweaked if need be. The automatic conversion performed by modeling tools for Oracle to MySQL data objects is a huge time saver and can result in lots of productivity gains for a database migration team.

The conversion of code objects is a different matter, however help is on the way as a number of MySQL partners are working on automatic code conversion tools that will reverse-engineer Oracle PL/SQL code objects and build MySQL complements for them.

Turning the attention back to the conversion of data objects, the MySQL Migration Toolkit automatically converts any source database reverse engineered through the tool to a MySQL counterpart, complete with all datatype translations and other like conversions. A user has complete control over exactly what objects are migrated as well as how MySQL specific designations – such as what underlying storage engine to use – are done.



As already noted, this particular step can be extremely challenging, and if not done properly, a DBA can find themselves spinning their wheels in redesigning objects or troubleshooting data transfer failures that occur because of invalid datatype mapping. However, when one uses the MySQL Migration Toolkit, such issues are a thing of the past as the tool does all the work right the first time.

4.4 Step 3 – Run Migration to MySQL

Once the source Oracle metadata has been understood and the MySQL target database designed, the next step is to run the actual data migration process. The extract, transform, and load (ETL) stage can be quite elaborate depending on what one wants to accomplish, and in addition to the MySQL Migration Toolkit, there are many heavy-duty third party ETL tools on the market that offer extreme flexibility in just how to move, aggregate, map, and transform data from Oracle to MySQL databases. Third-party vendors such as Informatica, Embarcadero DT/Studio, Goldengate and others offer support for migrating complex Oracle databases to MySQL. For those needing to continually perform

database migrations (such as those transferring data from source systems to data warehouses), purchasing such tools is a good investment.

For one-time transfers, however, the pricetag of such ETL tools doesn't make much sense. Fortunately, the MySQL Migration toolkit can come to the rescue as it offers a number of options to those needing to move a source Oracle database to MySQL, and the price can't be beat as the Migration Toolkit is free and published under the open source GPL license.

One data migration strategy is to migrate all the existing Oracle objects and data from the source database to a MySQL staging database. Once safely in the MySQL database server, a DBA or developer can create stored procedures or other migration code that selectively moves and manipulates data from the staging database into another MySQL database that will be used for further development or production.

Another option is to use the scripting capabilities of the MySQL Migration Toolkit to massage the Oracle data as it's being transferred from the Oracle source to the MySQL target database. Although most will use the graphical interface of Migration Toolkit to perform database migrations in point-and-click style, advanced users can exploit the underlying architecture to script out more complex ETL processes and schedule them if need be.

For large Oracle data movement tasks, the Migration Toolkit moves data in bulk fashion and ensures that all database and object properties that contribute to fast load times are automatically set. Users of the Migration Toolkit are kept informed of the Migration progress through easy to read progress bars and other informative messages. Once completed, the user is presented with a summary report of all tasks that have been carried out.

4.5 Oracle to MySQL Migration Validation

Once everything has been extracted from the source Oracle database into MySQL, it is wise to perform follow-up checks to ensure everything is as it should be with respect to matching object and row counts as well as ancillary items such as indexes and supporting code objects (views, etc.). It is also smart to perform a set of benchmark tests so acceptable performance can be validated. This step is perhaps the most neglected of all the migration lifecycle steps as it has traditionally not been easy to do.

Proper performance testing catches the performance problems that inadequate user and quality assurance testing miss. In a nutshell, performance testing simulates what is expected from real world use. It stresses the MySQL database in ways that could otherwise only be accomplished by opening the floodgates of the production user community.

Smart performance testing uses the following components to pull off a realistic simulation of what a database will experience during expected production usage:

- Anticipated user presence – it is critical that the test simulate the number of user connections that are expected during peak times and normal working hours. This is the major area where manual methods that pick a subset of users to test a database and application fail. The database may run just fine with 10 or so user connections, but may fall over when 400 connect to the system.
- Repetitive user activity – once the anticipated user sessions have connected to the database, they obviously have to "do something" for the system to be stressed. And they can't just "do something" once. Either all or a portion of the connected sessions need to repetitively perform tasks as they would occur during a normal workday. For an OLTP system, this may mean entering multiple orders. For a data warehouse, this may mean issuing long running analytical queries. The key is that the work is repetitive so repeated blows are dealt against the database.

- Extended duration - once you have a set number of sessions performing repetitive work, you next need to ensure that the work continues for a period of time that makes the test meaningful. What you are looking for is to unearth problems that take time to develop. For example, a MySQL table may not become fragmented after 30 minutes of OLTP work, but may surprisingly fragment in a dramatic fashion after 2 or more hours of repeated action.
- Expected production data volume – to have a truly valid test, you need to load your database with test data that is approximately the size you expect your database to be in the real world. This component is easily met if an existing production database has been migrated to MySQL.

Most DBAs are in a quandary over how to practically put in play all the points above. There exists open source benchmarking tools, but most have some shortcomings that cause them to miss the mark in producing an accurate performance test.

Third party vendor tools can come to the rescue, and some do the job better than others. A very nice performance testing tool, Benchmark Factory, is marketed by Quest Software. Benchmark Factory is extremely easy to use and offers very flexible point-and-click wizards that take the user completely through the process of setting up and running many of the standard TPC benchmarks for MySQL as well as other scalable performance tests. After the tests are finished, Benchmark Factory supplies analysis facilities that provide easy comparisons of benchmark runs, as well as the ability to export test results to Microsoft Excel.

In MySQL 5.1, a new utility has been introduced – mysqlslap – that offers a command line driven way of benchmarking a MySQL server. The utility offers a quick way of creating a test schema, populating that schema with test data, creating any number of virtual users that execute a series of predefined or ad-hoc SQL statements against a test database, and reporting on the test results.

4.6 Advice on Managing Oracle and MySQL Together

If MySQL will coexist in a data center alongside Oracle, what are some best practice tips that help a DBA who is charged with managing both platforms? Although not exhaustive, the following advice will help DBAs ensure they are being as productive as they possibly can be:

- Database tools can provide huge gains in productivity if the right products are chosen. Many third party database tools vendors today support MySQL and Oracle together, so the DBA should investigate whether these tools make sense for them. For example, Quest TOAD is heavily used in many Oracle shops by DBAs because of the time-savings benefits it offers. A DBA who is used to the TOAD interface can easily transfer to MySQL by using the TOAD for MySQL product offered by Quest. The same is true for users of Embarcadero's DBArtisan product that supports Oracle. DBArtisan now supports MySQL, which means a DBA can use one tool to manage both the Oracle and MySQL platforms. Of course, a DBA can also utilize tools supplied from the database vendor such as Oracle's OEM product and MySQL's Administrator and Query Browser tools (all free from MySQL).
- Oracle DBAs who are used to enabling Oracle to automatically take care of itself should utilize the same complementary MySQL features as well so any additional management is avoided. This includes things like enabling InnoDB tablespace files to automatically grow (like AUTOGROW for Oracle datafiles), setting up MySQL binary logging for point-in-time recovery (like Archive logging in Oracle), and other like items.
- Remember that in MySQL, unlike Oracle, you have a number of different underlying database engines at your disposal that are designed for particular application scenarios. Make sure you choose the right engine for the right purpose.
- Generous memory allocations help MySQL in the areas of performance just like in Oracle, so depending on the underlying engine being used, ensure enough RAM is provided for enabling memory access to data (as opposed to data being read from disk) as the defaults for MySQL are too low for most every enterprise application. Realize too, just as in Oracle, that just

because data is always read in RAM, it doesn't mean the actual SQL code is efficient. Unnecessary logical I/O can hurt performance just like physical I/O can.

- If multiple instances of MySQL will be managed on one box (like many DBAs do with Oracle), then the MySQL instance manager should be utilized. Introduced in version 5.0, it boosts a MySQL DBA's productivity by providing an easy way to remotely start, stop, edit, and control MySQL instances. Further, it allows easy remote access to error and other MySQL logs.
- Because inefficient SQL can have such a drastic negative impact on an otherwise well-running MySQL database, MySQL DBAs should enable the slow log to catch poorly-tuned SQL code and review the results of the log on a periodic basis. With MySQL, the slow log or the general log can be set up to mirror the results found in Oracle's V$SQLAREA and other performance views that are used to ferret out bad SQL code. Version 5.1 provides SQL performance tables that can be queried via SQL to get inefficient code metrics. In terms of other performance monitoring advice, the MySQL SHOW GLOBAL STATUS and SHOW INNODB STATUS commands can be used to examine raw MySQL performance metrics. MySQL Network offers more advanced monitoring and advice functionality, and should be used in most production environments where MySQL is deployed.
- MySQL DBAs should use whatever schedule they use to check Oracle Alert Logs to also check the MySQL Error Log for any occurrences of abnormal behavior.

5 Conclusion

From both a business and technical perspective, the question of whether a migration to MySQL from Oracle makes sense is easily answered. Countless modern enterprises are enjoying huge cost savings with MySQL while at the same time powering their most demanding database-driven systems with ease using MySQL's scale-out architecture.

Although Oracle database migrations are not something normally tackled with ease, following the simple data migration steps outlined in this paper helps ensure a smooth transition and ultimate success in the end. And to assist with migrations from Oracle, MySQL offers the freely downloadable Migration Toolkit, which greatly lessens the amount of time it takes to perform database migrations and drastically reduces the amount of errors that normally result when such complex operations are attempted by hand.

Making the switch from Oracle to MySQL – whether done in full or in a partial format where both MySQL and Oracle are used for the proper application situation – can make great sense, both from a financial and a technology perspective. By following the guidelines and steps in this paper, you can be assured that you will succeed in your implementation of MySQL no matter whether you are just testing the waters with open source or have made the full commitment to make it your preferred deployment platform.

6 Resources

6.1 MySQL Migration Aids

MySQL Migration Toolkit
http://www.mysql.com/products/tools/migration-toolkit/
Graphically migrate any existing Oracle, SQL Server, Microsoft Access or JDBC accessible database to MySQL.

MySQL Migration Central
http://www.mysql.com/why-mysql/migration/
Learn how to quickly and easily migrate your existing database to MySQL.

Migration Forums
http://forums.mysql.com/
Interact with other MySQL professionals who are involved with current database migrations.

6.2 Case Studies

Cox Communications Powers Massive Data Warehouse with MySQL, MySQL AB
http://www.mysql.com/it-resources/case-studies/cox.php
Cox Communications is the fourth largest cable-television provider in the United States, serving approximately 6.3 million customers. To maintain optimum performance and customer-service levels, Cox has developed a huge data warehousing application. At the heart of this business-critical system is a 2-billion row MySQL database.

Friendster Scales Out with MySQL Network, MySQL AB
http://www.mysql.com/it-resources/case-studies/cox.php
As the largest social networking site on the web — with over 60 million page-views each day, Friendster selected MySQL Network to provide the right combination of affordable database reliability and scalability. Its Scaled-Out LAMP system handles over one billion database queries a day, and has saved its IT staff several millions of dollars in hardware and software costs.

Citysearch Saves Over $1 Million with MySQL, MySQL AB
http://www.mysql.com/it-resources/case-studies/mysql-citysearch-casestudy.pdf
Citysearch is a leading online local search service, providing the most up-to-date information on businesses, from restaurants and retail, to travel and professional services. The Citysearch web site receives over 10 million unique visitors each month. As a result of an exponentially growing user base, the front end Oracle system could no longer keep up with the traffic coming from the web site. By implementing replicated MySQL servers to handle the web traffic and a terabyte of data, Citysearch was able to deliver an extremely high performance solution and save more than $1 million in database licenses and hardware costs.

6.3 White Papers

A Guide to Lower Database TCO, MySQL AB
http://www.mysql.com/tco
A Computerworld article, "MySQL Breaks Into the Data Center" revealed how MySQL has become the world's most popular open source database and why corporations intent on lowering their cost of operations are using it to further commoditize their IT infrastructure. In this white paper we'll show you

how. You'll also learn how organizations such as Cox Communications, NASA, Sabre Holdings and Yahoo! have improved database reliability, performance and TCO using MySQL.

A Guide Cost Effective Scale-Out using MySQL, MySQL AB
http://www.mysql.com/why-mysql/white-papers/mysql_wp_scaleout.php
Countless modern enterprises are using MySQL in conjunction with commodity hardware to realize incredible cost savings and nearly unlimited application scalability. In this white paper, we'll cover the techniques that MySQL customers are using today to power and protect their most important business applications.

7 Appendix A – Oracle to MySQL - Datatypes

7.1 Overview

This section provides a list of the built-in data type's native to Oracle Database 10g and compares them to equivalent data types supported by MySQL version 5.0. The data types are split into four categories:

- Data types for binary data
- Data types for character data
- Data types for numeric data
- Data types for temporal data

7.2 Synopsis

Oracle 10g supports only 22 native data types. Numerous other data type keywords are, however, accepted by the Oracle server, which maps them to the appropriate native data type.

MySQL provides an exact data type equivalent for many of Oracle's native data types, as well as for many of the data type synonyms supported by Oracle. The exceptions are:

- Oracle native INTERVAL DAY TO SECOND
- INTERVAL YEAR TO MONTH
- ROWID
- TIMESTAMP WITH LOCAL TIME ZONE
- TIMESTAMP WITH TIME ZONE
- UROWID

The Oracle native DATE data type has a MySQL equivalent (DATE or DATETIME) only for data with values that fall into the range January 1, 1000 AD to December 31, 9999 AD. Oracle DATE data that falls outside this range cannot be mapped to a MySQL equivalent.

The Oracle native TIMESTAMP data type has a MySQL equivalent (DATETIME) only for data with values that fall into the range '1000-01-01 00:00:01' to '9999-12-31 23:59:59'. Oracle TIMESTAMP data that falls outside this range cannot be mapped to a MySQL equivalent.

Oracle native BFILE, BINARY_DOUBLE, BINARY_FLOAT, BLOB, CLOB, LONG RAW, NCLOB, NUMBER, NVARCHAR2, RAW, and VARCHAR2 data types must be mapped to equivalent MySQL data types with different names.

Oracle data type synonyms CHAR VARYING, NATIONAL CHAR VARYING, NATIONAL CHARACTER VARYING, and NCHAR VARYING must be mapped to equivalent MySQL data types with different names.

Note: In the case of character string data, Oracle character set information must also be mapped to the equivalent MySQL character set information.

7.3 Data types for binary data

Summary

Map Oracle BFILE to MySQL LONGBLOB.
Map Oracle BLOB to MySQL LONGBLOB.
Map Oracle RAW(size) to MySQL BLOB.
Map Oracle LONG RAW to MySQL MEDIUMBLOB or LONGBLOB.

No other binary data types are supported by Oracle, but VARCHAR can be used to store binary data, and is often used for this purpose.

Oracle native data types

BFILE

Stores a locator to a large binary file stored outside the database.
Maximum size: 4GB.
Synonyms: None.
MySQL equivalent data type: None.

BLOB

Binary Large Object, stores variable-length binary data.
Maximum size: (4GB - 1) * (database block size).
Synonyms: None.
MySQL equivalent data type: LONGBLOB, maximum size 4GB.
Note: MySQL also supports a BLOB data type, but with maximum size of 65,536 bytes.

RAW(size)

Stores fixed-length, raw binary data up to size bytes long.
Maximum size: 2000 bytes.
Synonyms: None.
MySQL equivalent data type: BLOB.

LONG RAW

Stores variable-length, raw binary data, up to 2GB long.
Deprecated: Oracle recommends use of BLOB.
Synonyms: None.
MySQL equivalent data type: MEDIUMBLOB.

Non-native data types accepted as synonyms by Oracle

None.

7.4 Data types for character data

Summary

Map Oracle CLOB to MySQL LONGTEXT.
Map Oracle NCLOB to MySQL LONGTEXT CHARACTER SET utf8.
Map Oracle NVARCHAR2(length), and its synonyms NATIONAL CHARACTER VARYING(length), NATIONAL CHAR VARYING(length), NCHAR VARYING(length), to MySQL VARCHAR(length) CHARACTER SET utf8.
Map Oracle VARCHAR2(length), and its synonyms CHAR VARYING(length) and VARCHAR(length), to MySQL VARCHAR(length).
Oracle ROWID and UROWID have no MySQL equivalent.

All other Oracle character data types have exact MySQL equivalents, and therefore no action is necessary when migrating from Oracle to MySQL.

7.5 Oracle native data types

CHAR(length [BYTE | CHAR])

Stores fixed-length character strings, exactly length bytes or characters long.
Maximum length: 2000 bytes or characters.
Default and minimum length: 1 byte.
The data is stored in the database character set.
Synonyms: CHARACTER(length)
MySQL equivalent data type: CHAR(length).

CLOB

Character Large Object, stores variable-length character data.
The data is stored in the database character set.
Maximum size: (4GB - 1) * (database block size).
Synonyms: None.
MySQL equivalent data type: LONGTEXT, maximum size 4GB.

LONG

Stores variable-length character data, up to 2GB long.
Deprecated; Oracle recommends use of CLOB, NCLOB, BLOB.
Synonyms: LONG VARCHAR
MySQL equivalent data type: MEDIUMTEXT.

NCHAR(length)

Stores fixed-length character strings, exactly length characters long.
Maximum length: the number of characters that will fit in 2000 bytes.
Default and minimum length: 1 character.
The data is stored in the national character set (Unicode).
Synonyms: NATIONAL CHAR, NATIONAL CHARACTER
MySQL equivalent data type: NCHAR(length).

NCLOB

National Character Large Object, stores variable-length character data.
The data is stored in the national character set (Unicode).
Maximum size: (4 gigabytes - 1) * (database block size).
Synonyms: None.
MySQL equivalent data type: LONGTEXT CHARACTER SET utf8, maximum size 4GB.

NVARCHAR2(length)

Stores variable-length character strings, up to length characters long.
The data is stored in the national character set (Unicode).
Maximum length: the number of characters that will fit in 4000 bytes.
Synonyms: NATIONAL CHARACTER VARYING(length), NATIONAL CHAR VARYING(length), NCHAR VARYING(length)
MySQL equivalent data type: VARCHAR(length) CHARACTER SET utf8.

ROWID

Stores Base 64 strings that represent the unique address of a row in its table.
A ROWID is a unique identifier of a row. ROWID is a pseudo-column, i.e. you can always do SELECT ROWID FROM <table>. The data type of the ROWID pseudo-column is ROWID. A ROWID column created explicitly may be used to reference some other row, but this is uncommon. Rather, the ROWID data type is mostly used by stored procedures, taking a ROWID as an argument or, more commonly, returning a ROWID.
Synonyms: None.
MySQL equivalent data type: None.

UROWID[(length)]

Stores Base 64 strings that represent the logical address of a row of an index-organized table.
Maximum and default length: 4000 bytes.
See the discussion about ROWID, above. A UROWID is used for the same purpose as a ROWID, but for index-organized tables.
Synonyms: None.
MySQL equivalent data type: None.

VARCHAR2(length [BYTE | CHAR])

Stores variable-length character strings, up to length bytes or characters long.
Maximum length: 4000 bytes or characters.
Minimum length: 1 byte or character.
Synonyms: CHAR VARYING(length), CHARACTER VARYING(length), VARCHAR(length)
MySQL equivalent data type: VARCHAR(length).

Non-native data types accepted as synonyms by Oracle

CHARACTER(length)

Standard SQL data type, treated as a synonym for Oracle CHAR(length).
MySQL equivalent data type: CHARACTER(length).

CHAR VARYING(length)

Standard SQL data type, treated as a synonym for Oracle VARCHAR2(length).
MySQL equivalent data type: VARCHAR(length).

CHARACTER VARYING(length)

Standard SQL data type, treated as a synonym for Oracle VARCHAR2(length).
MySQL equivalent data type: CHARACTER VARYING(length).

LONG VARCHAR

IBM DB2 data type, treated as a synonym for Oracle LONG.
MySQL equivalent data type: MEDIUMTEXT.

NATIONAL CHAR(length), NATIONAL CHARACTER(length)

Standard SQL data types, treated as synonyms for Oracle's NCHAR(length).
MySQL equivalent data types: NATIONAL CHAR(length), NATIONAL CHARACTER(length).

NATIONAL CHARACTER VARYING(length), NATIONAL CHAR VARYING(length), NCHAR VARYING(length)

Standard SQL data types, treated as synonyms for Oracle's NVARCHAR2(length).
MySQL equivalent data type: VARCHAR(length) CHARACTER SET utf8.

VARCHAR(length)

Standard SQL data type, treated as a synonym for Oracle VARCHAR2(length).
MySQL equivalent data type: VARCHAR(length).

7.6 Data types for numeric data

Summary

Map Oracle BINARY_DOUBLE to MySQL DOUBLE PRECISION.
Map Oracle BINARY_FLOAT to MySQL FLOAT.
Map Oracle NUMBER to MySQL DOUBLE PRECISION.

All other Oracle numeric data types have exact MySQL equivalents, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle native data types

BINARY_DOUBLE

Stores 64-bit floating point numbers.
Synonyms: None.
MySQL equivalent data type: DOUBLE PRECISION.

BINARY_FLOAT

Stores 32-bit floating point numbers.
Synonyms: None.
MySQL equivalent data type: FLOAT.

NUMBER(p,s)

Stores numbers, with precision p and scale s.
Precision can range from 1 to 38 digits of decimal precision.
Scale can range from -84 to 127.
The NUMBER data type can be either fixed point or floating point. If p and/or s are specified, NUMBER is treated as fixed point. This means that incorrect data type mappings may lead to rounding errors.
Synonyms: DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, NUMERIC, REAL, SMALLINT.
MySQL equivalent data types: DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, NUMERIC, REAL, SMALLINT, depending on precision, scale, and range requirements.

Non-native data types accepted as synonyms by Oracle

DECIMAL(p,s)

Standard SQL data type, treated as a synonym for Oracle NUMBER(p,s).
MySQL equivalent data type: DECIMAL(p,s).
See discussion above, on NUMBER.

DOUBLE PRECISION

Standard SQL data type, treated as a synonym for Oracle NUMBER with 64 bits precision.
MySQL equivalent data type: DOUBLE PRECISION.

FLOAT[(precision)]

Standard SQL data type, treated as a synonym for Oracle NUMBER with 64 bits precision.
MySQL equivalent data type: FLOAT(precision).

INT, INTEGER

Standard SQL data type, treated as a synonym for Oracle NUMBER(38).
MySQL equivalent data types: INT, INTEGER.

NUMERIC(p,s)

Standard SQL data type, treated as a synonym for Oracle NUMBER(p,s).
MySQL equivalent data type: NUMERIC(p,s).

REAL

Standard SQL data type, treated as a synonym for Oracle NUMBER with 64 bits precision.

MySQL equivalent data type: REAL.

SMALLINT

Standard SQL data type, treated as a synonym for Oracle NUMBER(38).
MySQL equivalent data types: SMALLINT.

7.7 Data types for temporal data

Summary

The Oracle DATE data type is really a TIMESTAMP, although the default format for a DATE column leaves out the TIME part.

Map Oracle DATE to MySQL DATE if (a) only the DATE part of the data is of interest and (b) the date falls into the range supported by MySQL (January 1, 1000 AD to December 31, 9999 AD). Otherwise, map to MySQL DATETIME, with the same limited range.

Map Oracle TIMESTAMP[(precision)] to MySQL DATETIME only for data with no fractional seconds precision that falls into the range '1000-01-01 00:00:01' to '9999-12-31 23:59:59'. Oracle TIMESTAMP data that falls outside this range cannot be mapped to a MySQL equivalent.

Oracle INTERVAL DAY[(precision)] TO SECOND[(fractional_precision)] and INTERVAL YEAR[(precision)] TO MONTH have no MySQL equivalent.

Oracle TIMESTAMP[(precision)] WITH TIME ZONE and TIMESTAMP[(precision)] WITH LOCAL TIME ZONE have no MySQL equivalent.

No other temporal data types are supported by Oracle.

Oracle native data types

DATE

Stores date and time.
Valid range: January 1, 4712 BC to December 31, 9999 AD.
Synonyms: None.
MySQL equivalent data type: DATETIME, with a limited range from January 1, 1000 AD to December 31, 9999 AD, or, if the TIME part of the value is irrelevant, DATE, with the same range.

INTERVAL DAY[(precision)] TO SECOND[(fractional_precision)]

Stores periods of time in days, hours, minutes, and seconds.
precision is the maximum number of digits in the DAY field.
fractional_precision is the number of digits in the fractional portion of the SECOND field.
precision range: 0 to 9, defaults to 2.
fractional_precision range: 0 to 9, defaults to 6.
Synonyms: None.
MySQL equivalent data type: None.

INTERVAL YEAR[(precision)] TO MONTH

Stores periods of time in years and months.
precision is the number of digits in the YEAR field.
precision range: 0 to 9, defaults to 2.
Synonyms: None.
MySQL equivalent data type: None.

TIMESTAMP[(precision)]

Stores the YEAR, MONTH, DAY values of a date, plus the HOUR, MINUTE, SECOND values of the related time.
precision is the number of digits in the fractional portion of the SECOND field.
precision range: 0 to 9, defaults to 6.
Synonyms: None.
MySQL equivalent data type: DATETIME, with no fractional seconds precision and a limited range from the beginning of January 1, 1000 AD to the end of December 31, 9999 AD (e.g. '1000-01-01 00:00:01' to '9999-12-31 23:59:59').

TIMESTAMP[(precision)] WITH TIME ZONE

Stores TIMESTAMP values that include a time zone displacement value.
Synonyms: None.
MySQL equivalent data type: None.

TIMESTAMP[(precision)] WITH LOCAL TIME ZONE

Stores TIMESTAMP WITH TIME ZONE values, normalized to the database time zone when stored. On retrieval, the data is shown in the session time zone.
Synonyms: None.
MySQL equivalent data type: None.

Non-native data types accepted as synonyms by Oracle

None.

8 Appendix B – Oracle to MySQL - Predicates

8.1 Overview

This section provides a list of the SQL predicates supported by Oracle Database 10g and compares them to equivalent predicates supported by MySQL version 5.0. The predicates are split into nine categories:

- Predicates for comparison conditions
- Predicates for exists conditions
- Predicates for floating-point conditions
- Predicates for in conditions
- Predicates for is/member conditions
- Predicates for null conditions
- Predicates for pattern matching conditions
- Predicates for range conditions
- Predicates for XML conditions

8.2 Synopsis

Oracle 10g and MySQL 5.0 both support the use of conditions in the WHERE clause of the SELECT, DELETE, and UPDATE statements, as well as in the HAVING clause of the SELECT statement. Oracle also supports conditions in the START WITH and CONNECT BY clauses of the SELECT statement; MySQL does not support this feature.

Oracle supports 43 predicates and 3 Boolean operators that can be used to construct conditions in SQL statements. MySQL provides an exact equivalent for most of these. The exceptions are:

Oracle REGEXP_LIKE(string_expression, pattern) must be mapped to MySQL string_expression REGEXP pattern. Oracle REGEXP_LIKE(string_expression, pattern, match_parameter) can not be mapped to a MySQL equivalent.

Oracle EQUALS_PATH, IS A SET, IS ANY, IS EMPTY, IS INFINITE, IS NAN, IS OF type, IS NOT A SET, IS NOT EMPTY, IS NOT INFINITE, IS NOT NAN, IS NOT OF type, IS PRESENT, LIKEC, LIKE2, LIKE4, MEMBER, NOT LIKEC, NOT LIKE2, NOT LIKE4, NOT MEMBER, SUBMULTISET, NOT SUBMULTISET, and UNDER_PATH have no MySQL equivalent.

8.3 Boolean conditions

A Boolean condition combines two other conditions, to return a single result. Oracle supports three Boolean conditions.

Summary

Each Oracle Boolean condition has an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle boolean conditions

expression1 AND expression2
Returns TRUE if both expressions return TRUE, UNKNOWN if either expression is NULL, FALSE otherwise.

MySQL equivalent: AND

NOT expression
Negates the result of expression; returns TRUE if expression is FALSE, UNKNOWN if expression is NULL, and FALSE if expression is TRUE.

MySQL equivalent: NOT

expression1 OR expression2
Returns TRUE if either expression returns TRUE, UNKNOWN if either expression is NULL, FALSE otherwise.

MySQL equivalent: OR

8.4 Comparison conditions

Comparison conditions compare two expressions. Oracle supports eight comparison conditions.

Summary

Each Oracle comparison condition has an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle comparaison conditions

expression1 = expression2
Returns TRUE if the expressions are equal, UNKNOWN if either expression is NULL, and FALSE otherwise.

MySQL equivalent: =

expression1 <> expression2
Additional formats accepted (not all forms available on every platform): != ^= ¬=
Returns TRUE if the expressions are not equal, UNKNOWN if either expression is NULL, and FALSE otherwise.

MySQL equivalent: <> !=

expression1 < expression2
Returns TRUE if expression1 is less than expression2, UNKNOWN if either expression is NULL, and FALSE otherwise.

MySQL equivalent: <

expression1 <= expression2
Returns TRUE if expression1 is less than or equal to expression2, UNKNOWN if either expression is NULL, and FALSE otherwise.

MySQL equivalent: <=

expression1 > expression2

Returns TRUE if expression1 is greater than expression2, UNKNOWN if either expression is NULL, and FALSE otherwise.
MySQL equivalent: >

expression1 >= expression2

Returns TRUE if expression1 is greater than or equal to expression2, UNKNOWN if either expression is NULL, and FALSE otherwise.

MySQL equivalent: >=

expression1 comparison ANY expression2

expression1 comparison SOME expression2
Synonyms; return TRUE if the comparison condition returns TRUE for at least one value in expression2 and FALSE if the comparison condition returns either FALSE for every value in expression2 or returns an empty set (zero rows).
comparison must be one of: = <> < <= > >=.
expression2 is generally a subquery, but may resolve to any expression.

MySQL equivalent: ANY, SOME, provided expression2 is a subquery.

expression1 comparison ALL expression2

Returns FALSE if the comparison condition returns FALSE for at least one value in expression2 and TRUE if the comparison condition returns either TRUE for every value in expression2 or returns an empty set (zero rows).
comparison must be one of: = <> < <= > >=.
expression2 is generally a subquery, but may resolve to any expression.

MySQL equivalent: ALL, provided expression2 is a subquery.

8.5 Exists conditions

Exists conditions test for rows in a sub query. Oracle supports two exists conditions.

Summary

Each Oracle exists condition has an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle exists conditions

*EXISTS (*subquery*)*

Returns TRUE if *subquery* returns at least one row, FALSE otherwise.

MySQL equivalent: EXISTS

*NOT EXISTS (*subquery*)*

Returns TRUE if *subquery* returns zero rows, FALSE otherwise.

MySQL equivalent: NOT EXISTS

8.6 Floating-point conditions

Floating-point conditions determine whether the result of an expression is infinite or NaN (not a number). Oracle supports four floating-point conditions.

Summary

Oracle IS INFINITE, IS NOT INFINITE, IS NAN, and IS NOT NAN have no MySQL equivalent.

No other floating-point conditions are supported by Oracle.

Oracle floating-point conditions

numeric_expression IS INFINITE
Returns TRUE if *numeric_expression* resolves to the special value +INF or -INF, UNKNOWN if the expression is NULL, FALSE otherwise.
MySQL equivalent: None.

numeric_expression IS NOT INFINITE
Returns TRUE if numeric_expression does not resolve to the special value +INF or -INF, UNKNOWN if the expression is NULL, FALSE otherwise.
MySQL equivalent: None.

numeric_expression IS NAN
Returns TRUE if numeric_expression resolves to the special value Nan, UNKNOWN if the expression is NULL, FALSE otherwise.
MySQL equivalent: None.

numeric_expression IS NOT NAN
Returns TRUE if numeric_expression does not resolve to the special value Nan, UNKNOWN if the expression is NULL, FALSE otherwise.
MySQL equivalent: None.

8.7 In conditions

In conditions test whether a value is found in a set. Oracle supports two in conditions.

Summary

Each Oracle in condition has an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle in conditions

expression IN {value_list | (subquery)}
Returns TRUE if expression equals any value in value_list (or returned by subquery), UNKNOWN if either argument is NULL, FALSE otherwise.

MySQL equivalent: IN

expression NOT IN {value_list | (subquery)}
Returns TRUE if expression does not equal any value in value_list (or returned by subquery), UNKNOWN if either argument is NULL, FALSE otherwise.

MySQL equivalent: NOT IN

8.8 Is/member conditions

Is/member conditions test whether a value is present in a specific way. Oracle supports 12 is/member conditions.

Summary

Oracle IS [NOT] A SET, IS ANY, IS [NOT] EMPTY, IS [NOT] OF type, IS PRESENT, [NOT] MEMBER, [NOT] SUBMULTISET have no MySQL equivalent.

No other is/member conditions are supported by Oracle.

Oracle is/member conditions

nested_table IS A SET

Returns TRUE if nested_table contains only unique elements, UNKNOWN if nested_table is empty, FALSE otherwise.

MySQL equivalent: None.

nested_table IS NOT A SET

Returns TRUE if nested_table contains duplicate elements, UNKNOWN if nested_table is empty, FALSE otherwise.

MySQL equivalent: None.

[dimension_column IS] ANY

Always returns TRUE; relevant only for interrow calculations.
IS ANY qualifies all values of a dimension column, including NULL.
IS ANY can be used only in the MODEL clause of a SELECT statement.

MySQL equivalent: None.

nested_table IS EMPTY

Returns TRUE if nested_table is empty, UNKNOWN if the argument is NULL, FALSE otherwise.
MySQL equivalent: None.

nested_table IS NOT EMPTY

Returns TRUE if nested_table is not empty, UNKNOWN if the argument is NULL, FALSE otherwise.

MySQL equivalent: None.

expression IS OF [type1] ({[ONLY schema.] type2} [, ...])

Returns TRUE if either (a) ONLY is not specified and the most specific type of expression is a subtype of type2 or (b) the most specific type of expression is in type2, UNKNOWN if expression is NULL, FALSE otherwise.

MySQL equivalent: None.

expression IS NOT OF [type1] ({[ONLY schema.] type2} [, ...])

Returns FALSE if either (a) ONLY is not specified and the most specific type of expression is a subtype of type2 or (b) the most specific type of expression is in type2, UNKNOWN if expression is NULL, TRUE otherwise.

MySQL equivalent: None.

cell IS PRESENT

Returns TRUE if cell exists before the MODEL clause is executed, FALSE otherwise; relevant only for interrow calculations.
IS PRESENT can be used only in the MODEL clause of a SELECT statement.

MySQL equivalent: None.

expression MEMBER [OF] nested_table

Returns TRUE if expression is a member of nested_table, UNKNOWN if expression is NULL or nested_table is empty, FALSE otherwise.

MySQL equivalent: None.

expression NOT MEMBER [OF] nested_table

Returns TRUE if expression is not a member of nested_table, UNKNOWN if expression is NULL or nested_table is empty, FALSE otherwise.
MySQL equivalent: None.

nested_table1 SUBMULTISET [OF] nested_table2

Returns TRUE if nested_table1 is a submultiset of *nested_table2*, UNKNOWN if either argument is NULL, FALSE otherwise.

MySQL equivalent: None.

nested_table1 NOT SUBMULTISET [OF] nested_table2

Returns TRUE if nested_table1 is not a submultiset of *nested_table2*, UNKNOWN if either argument is NULL, FALSE otherwise.

MySQL equivalent: None.

8.9 Null conditions

Null conditions test for null values. Oracle supports two null conditions.

Summary

Each Oracle null condition has an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle null conditions

expression IS NULL

Returns TRUE if the result of expression is NULL, FALSE otherwise.

MySQL equivalent: IS NULL

expression IS NOT NULL

Returns TRUE if the result of expression is not NULL, FALSE otherwise.

MySQL equivalent: IS NOT NULL

8.10 Pattern matching conditions

Pattern matching conditions test whether a value matches a specific pattern. Oracle supports nine pattern matching conditions.

Summary

Map Oracle REGEXP_LIKE(*string_expression, pattern*) to MySQL *string_expression* REGEXP *pattern*. Oracle REGEXP_LIKE(*string_expression, pattern*, *match_parameter*) can not be mapped to a MySQL equivalent**.**
Oracle [NOT] LIKEC, [NOT] LIKE2, and [NOT] LIKE4 have no MySQL equivalent.

Oracle pattern matching conditions

string_expression LIKE pattern [ESCAPE escape_string]

Returns TRUE if string_expression matches pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
Uses the current (input) character set to interpret all arguments.

MySQL equivalent: LIKE

string_expression NOT LIKE pattern [ESCAPE escape_string]

Returns TRUE if string_expression does not match pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
Uses the current (input) character set to interpret all arguments.

MySQL equivalent: NOT LIKE

string_expression LIKEC pattern [ESCAPE escape_string]

Returns TRUE if string_expression matches pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
Uses Unicode complete characters to interpret all arguments.

MySQL equivalent: None.

string_expression NOT LIKEC pattern [ESCAPE escape_string]

Returns TRUE if string_expression does not match pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
Uses Unicode complete characters to interpret all arguments.

MySQL equivalent: None.

string_expression LIKE2 pattern [ESCAPE escape_string]

Returns TRUE if string_expression matches pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
Uses UCS2 code points to interpret all arguments.

MySQL equivalent: None.

string_expression NOT LIKE2 pattern [ESCAPE escape_string]

Returns TRUE if string_expression does not match pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
Uses UCS2 code points to interpret all arguments.

MySQL equivalent: None.

string_expression LIKE4 pattern [ESCAPE escape_string]

Returns TRUE if string_expression matches pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
Uses UCS4 code points to interpret all arguments.

MySQL equivalent: None.

string_expression NOT LIKE4 pattern [ESCAPE escape_string]

Returns TRUE if string_expression does not match pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
Uses UCS4 code points to interpret all arguments.

MySQL equivalent: None.

REGEXP_LIKE(string_expression, pattern [, match_parameter])

Returns TRUE if string_expression matches pattern, UNKNOWN if any argument is NULL, FALSE otherwise.
REGEXP_LIKE tests string_expression against pattern using regular expression matching instead of simple pattern matching (done by LIKE).
pattern must be a regular expression.
match_parameter is a constant; it indicates the predicate's default matching behavior. The possible values are (a) 'i', to specify case-insensitive matching, (b) 'c', to specify case-sensitive matching, (c) 'n', to specify that a period (.) should match the newline character, and (d) 'm', to specify that string_expression should be treated as multiple lines.

If *match_parameter* is omitted, the default case sensitivity is determined by the value of the NLS_SORT parameter, a period does not match the newline character, and string_expression is treated as a single line.

MySQL equivalent: string_expression REGEXP pattern, only if match_parameter is omitted from the Oracle expression.

8.11 Range conditions

Range conditions test for inclusion in a range of values. Oracle supports two range conditions.

Summary

Each Oracle range condition has an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle range conditions

expression1 BETWEEN expression2 AND expression3

Returns TRUE if expression1 falls into the range of values specified by the other expressions (i.e. if expression1 is greater than or equal to expression2 and less than or equal to expression3), UNKNOWN if any expression is NULL, FALSE otherwise.

MySQL equivalent: BETWEEN

expression1 NOT BETWEEN expression2 AND expression3

Returns TRUE if expression1 does not fall into the range of values specified by expression2 and expression3, UNKNOWN if any expression is NULL, FALSE otherwise.

MySQL equivalent: NOT BETWEEN

8.12 XML conditions

XML conditions test whether an XML instance is present in a specific place. Oracle supports two XML conditions.

Summary

Oracle EQUALS_PATH and UNDER_PATH have no MySQL equivalent.

No other XML conditions are supported by Oracle.

Oracle XML conditions

EQUALS_PATH(column, path [, integer])

Returns TRUE if a resource in the XML database is found in the SQL database at path, UNKNOWN if any argument is NULL, FALSE otherwise.

MySQL equivalent: None.

UNDER_PATH(column [, levels] , path [, integer])

Returns TRUE if the resources in column are found under path in the XML database repository, UNKNOWN if any argument is NULL, FALSE otherwise.

MySQL equivalent: None.

8.13 Predicate precedence

Precedence refers to the order in which a DBMS evaluates multiple conditions in the same SQL expression. When assessing an expression, Oracle evaluates multiple conditions based on their assigned precedence, from highest to lowest. Conditions with equal precedence are evaluated from left to right. Parentheses can be added to an expression to override the assigned precedence; Oracle evaluates conditions inside parentheses before evaluating those outside. MySQL follows the same procedure.

Table 1 compares the levels of predicate precedence for Oracle and MySQL, from highest to lowest. Predicates shown on the same line have equal precedence and are therefore evaluated from left to right when found in the same expression. Oracle's documentation does not specify the precedence level for predicates that are not shown in Table 1.

ORACLE PREDICATE	**MYSQL PREDICATE**
SQL operators	SQL operators, NOT
Comparison (= <> < <= > >=)	Comparison (= <> < <= > >=), IS [NOT] NULL, LIKE, [NOT] IN, REGEXP
IS [NOT] NULL, LIKE, [NOT] BETWEEN, [NOT] IN, EXISTS, IS OF type	[NOT] BETWEEN
NOT	AND
AND	OR
OR	

Table 2 - Oracle and MySQL Predicate Precedence's

With the following exceptions, Oracle and MySQL conditions have the same level of precedence:

MySQL considers NOT to have the same precedence as an SQL operator, while Oracle places the precedence of NOT after that of the majority of predicates. MySQL considers IS [NOT] NULL, LIKE, and [NOT] IN to have a higher precedence than does Oracle.

Oracle expressions involving any of these conditions must thus be properly parenthesized when migrating to MySQL.

9 Appendix C – Oracle to MySQL - Operators

9.1 Overview

This paper provides a list of the built-in SQL operators supported by Oracle Database 10g and compares them to equivalent operators supported by MySQL version 5.0. The operators are split into five categories:

- Arithmetic operators
- Concatenation operators
- Hierarchical Query operators
- Multiset operators
- Set operators

9.2 Synopsis

Oracle 10g supports 17 operators that can be used to construct SQL expressions. MySQL provides an exact equivalent for some of these operators. The exceptions are:
Oracle || (concatenation) has an exact MySQL equivalent only if the MySQL server is started with the –ansi option. In all other cases, || must be mapped to an equivalent MySQL function (CONCAT).

Oracle CONNECT_BY_ROOT, INTERSECT, MINUS, MULTISET EXCEPT ALL, MULTISET EXCEPT DISTINCT, MULTISET INTERSECT ALL, MULTISET INTERSECT DISTINCT, MULTISET UNION ALL, MULTISET UNION DISTINCT, and PRIOR have no MySQL equivalent.

MySQL considers concatenation (||) to be a function rather than an operator. To ensure proper expression evaluation, parentheses should be added to concatenation in Oracle expressions when migrating to MySQL. No other action is needed to ensure proper expression evaluation when migrating from Oracle to MySQL.

9.3 Arithmetic operators

Summary

Each Oracle arithmetic operator has an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle arithmetic operators

+
When used as a unary operator, indicates a positive numeric or temporal expression.
When used as a binary operator, adds two numeric or temporal values.

MySQL equivalent: +

-
When used as a unary operator, indicates a negative numeric or temporal expression.
When used as a binary operator, subtracts one numeric or temporal value from another.
MySQL equivalent: -

*
Binary operator; multiplies numeric expressions.

MySQL equivalent: *

/
Binary operator; divides numeric expressions.

MySQL equivalent: /

9.4 Concatenation operators

Summary

When the MySQL server is started with the –ansi option, the Oracle || (concatenation) operator has an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

When the MySQL server is started in the default mode, map Oracle || to MySQL CONCAT('string1', 'string2').

No other concatenation operators are supported by Oracle, although Oracle's CONCAT string function serves the same purpose.

Oracle concatenation operators

||

Concatenates character strings and CLOB values.
MySQL equivalent (--ansi mode): ||
MySQL equivalent (default mode): CONCAT('string1', 'string2')

9.5 Hierarchical query operators

Summary

Oracle CONNECT_BY_ROOT and PRIOR have no MySQL equivalent.
No other hierarchical query operators are supported by Oracle.

Oracle hierarchical query operators

CONNECT_BY_ROOT
A unary operator, valid only in hierarchical queries. When used to qualify a column, causes Oracle to return the column value using data from the root row.

MySQL equivalent: None.

PRIOR
Valid only in hierarchical queries.
When used to compare column values, causes Oracle to use the value of the parent row.
MySQL equivalent: None.

9.6 Multiset operators

Summary

Oracle MULTISET EXCEPT, MULTISET INTERSECT, and MULTISET UNION have no MySQL equivalent.

No other multiset operators are supported by Oracle.

Oracle multiset operators

MULTISET EXCEPT ALL

Combines two nested tables.
Returns a nested table consisting of all elements (including duplicates) found in the first nested table, but not in the second nested table.

MySQL equivalent: None.

MULTISET EXCEPT DISTINCT

Combines two nested tables.
Returns a nested table consisting of all distinct (non-duplicate) elements found in the first nested table, but not in the second nested table.

MySQL equivalent: None.

MULTISET INTERSECT ALL

Combines two nested tables.
Returns a nested table consisting of all elements (including duplicates) found in both nested tables.

MySQL equivalent: None.

MULTISET INTERSECT DISTINCT

Combines two nested tables.
Returns a nested table consisting of all distinct (non-duplicate) elements found in both nested tables.

MySQL equivalent: None.

MULTISET UNION ALL

Combines two nested tables.
Returns a nested table consisting of all elements (including duplicates) found in either nested table.

MySQL equivalent: None.

MULTISET UNION DISTINCT

Combines two nested tables.
Returns a nested table consisting of all distinct (non-duplicate) elements found in either nested table.

MySQL equivalent: None.

9.7 Set operators

Summary

Oracle INTERSECT and MINUS have no MySQL equivalent.

All other Oracle set operators have an exact MySQL equivalent, and therefore no action is necessary when migrating from Oracle to MySQL.

Oracle set operators

INTERSECT

Combines two SELECT queries.
Returns all distinct (non-duplicate) rows found by both SELECTs.

MySQL equivalent: None.

MINUS

Combines two SELECT queries.
Returns all distinct (non-duplicate) rows found by the first SELECT but not the second.

MySQL equivalent: None.
Note: Some DBMSs, and the SQL Standard, call this the EXCEPT operator.

UNION

Combines two SELECT queries.
Returns all distinct (non-duplicate) rows found by either SELECT.

MySQL equivalent: UNION.

UNION ALL

Combines two SELECT queries.
Returns all rows found by either SELECT, including duplicates.

MySQL equivalent: UNION ALL.

9.8 Operator precedence

Precedence refers to the order in which a DBMS evaluates multiple operators in the same SQL expression. When assessing an expression, Oracle evaluates multiple operators based on their assigned precedence, from highest to lowest. Operators with equal precedence are evaluated from left to right. Parentheses can be added to an expression to override the Oracle's assigned operator precedence; the server evaluates expressions inside parentheses before evaluating those outside. MySQL follows the same procedure.

Table 1 compares the levels of operator precedence for Oracle and MySQL, from highest to lowest. Operators shown on the same line have equal precedence and are therefore evaluated from left to right when found in the same expression. The set operators (INTERSECT, MINUS, UNION) are not included in the table because they combine sets of rows, rather than atomic data values. In Oracle, all set operators have equal precedence.

ORACLE OPERATOR	MYSQL OPERATOR
Unary +, Unary - , PRIOR, CONNECT_BY_ROOT	Unary +, Unary -

ORACLE OPERATOR	MYSQL OPERATOR
*, /	*, /
Binary +, Binary -, ||	Binary +, Binary -
SQL predicates	SQL predicates

Table 3 - Oracle and MySQL Operator Precedence's

With the following exceptions, Oracle and MySQL operators have the same level of precedence:

MySQL has no operators equivalent to Oracle's PRIOR and CONNECT_BY_ROOT. Operator precedence is thus irrelevant. MySQL considers concatenation (||) to be a function rather than an operator. To ensure proper expression evaluation, parentheses should be added to concatenation in Oracle expressions when migrating to MySQL.

10 Appendix D – Sample Migration

This section provides a simple example of moving an Oracle database schema over to MySQL using the MySQL Migration Toolkit.

10.1 Sample Oracle Schema

The Oracle schema is a small database used for tracking financial investments in a small brokerage company. A graphical representation of the schema model is:

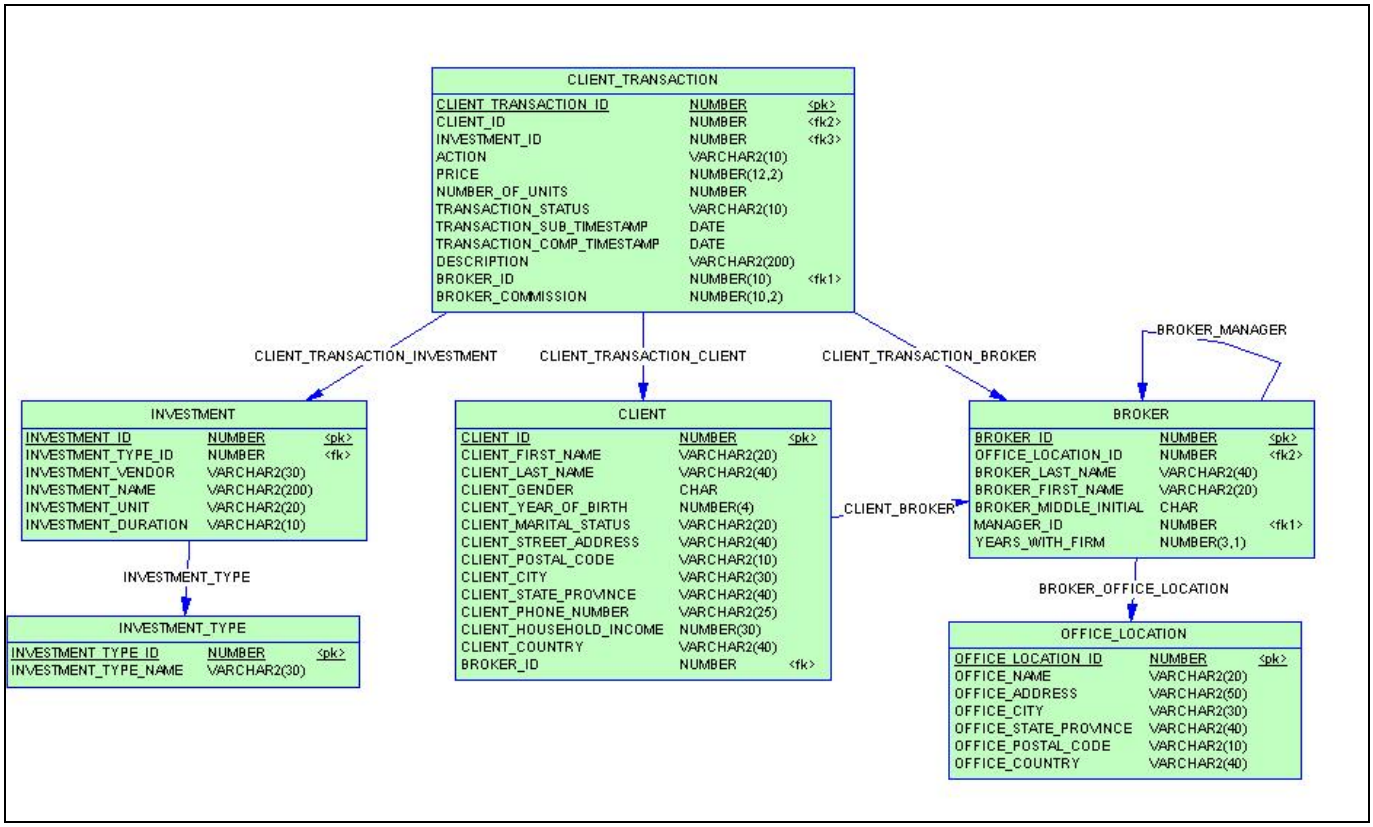


Figure 2 - Data model of the Oracle investment financial schema

The Oracle schema DDL is as follows:

```
/*==============================================================*/
/* DBMS name:      ORACLE Version 10g                           */
/* Created on:     2/27/2006 5:28:28 PM                         */
/*==============================================================*/


/*==============================================================*/
/* Table: BROKER                                                */
/*==============================================================*/
create table BROKER  (
   BROKER_ID            NUMBER                         not null,
   OFFICE_LOCATION_ID   NUMBER,
   BROKER_LAST_NAME     VARCHAR2(40)                   not null,
   BROKER_FIRST_NAME    VARCHAR2(20)                   not null,
```

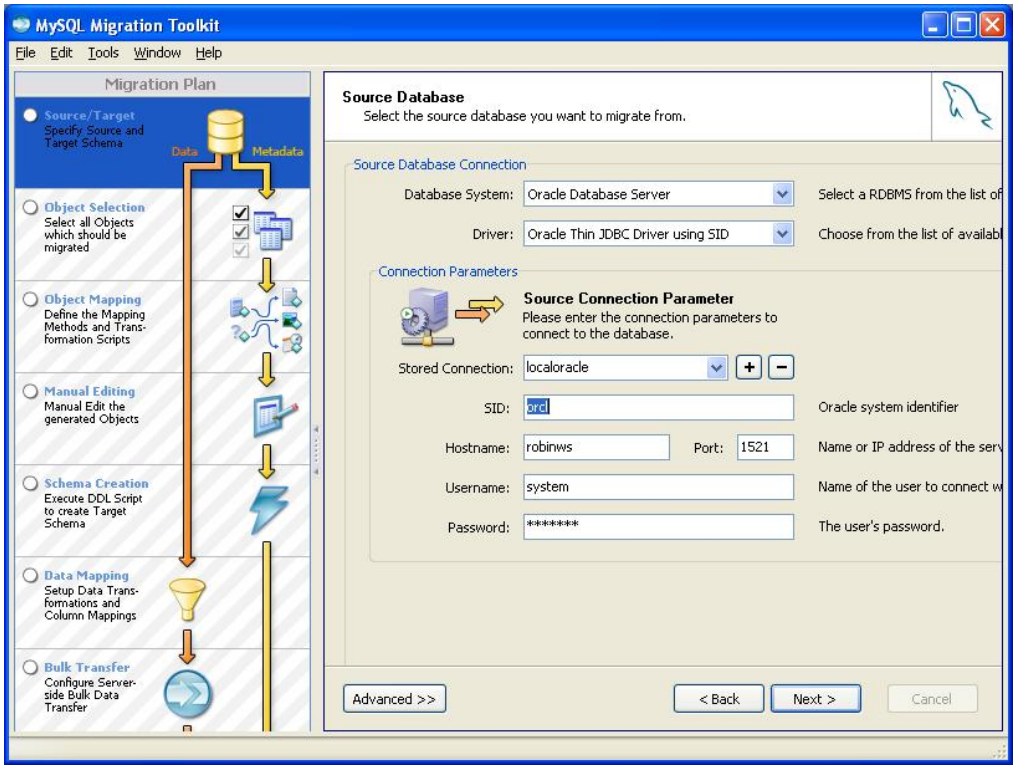
```
   BROKER_MIDDLE_INITIAL CHAR,
   MANAGER_ID           NUMBER,
   YEARS_WITH_FIRM      NUMBER(3,1)                    not null,
   constraint BROKER_PK primary key (BROKER_ID)
);

/*==============================================================*/
/* Index: BROKER_LOCATION                                       */
/*==============================================================*/
create index BROKER_LOCATION on BROKER (
   OFFICE_LOCATION_ID ASC
);

/*==============================================================*/
/* Table: CLIENT                                                */
/*==============================================================*/
create table CLIENT  (
   CLIENT_ID            NUMBER                         not null,
   CLIENT_FIRST_NAME    VARCHAR2(20)                   not null,
   CLIENT_LAST_NAME     VARCHAR2(40)                   not null,
   CLIENT_GENDER        CHAR                           not null,
   CLIENT_YEAR_OF_BIRTH NUMBER(4)                      not null,
   CLIENT_MARITAL_STATUS VARCHAR2(20),
   CLIENT_STREET_ADDRESS VARCHAR2(40)                   not null,
   CLIENT_POSTAL_CODE   VARCHAR2(10)                   not null,
   CLIENT_CITY          VARCHAR2(30)                   not null,
   CLIENT_STATE_PROVINCE VARCHAR2(40)                   not null,
   CLIENT_PHONE_NUMBER  VARCHAR2(25)                   not null,
   CLIENT_HOUSEHOLD_INCOME NUMBER(30),
   CLIENT_COUNTRY       VARCHAR2(40),
   BROKER_ID            NUMBER                         not null,
   constraint CLIENT_PK primary key (CLIENT_ID)
);

/*==============================================================*/
/* Index: CLIENT_BROKER                                         */
/*==============================================================*/
create index CLIENT_BROKER on CLIENT (
   BROKER_ID ASC
);

/*==============================================================*/
/* Table: CLIENT_TRANSACTION                                    */
/*==============================================================*/
create table CLIENT_TRANSACTION  (
   CLIENT_TRANSACTION_ID NUMBER                         not null,
   CLIENT_ID            NUMBER                         not null,
   INVESTMENT_ID        NUMBER                         not null,
   ACTION               VARCHAR2(10)                   not null,
   PRICE                NUMBER(12,2)                   not null,
   NUMBER_OF_UNITS      NUMBER                         not null,
   TRANSACTION_STATUS   VARCHAR2(10)                   not null,
   TRANSACTION_SUB_TIMESTAMP DATE                            not null,
   TRANSACTION_COMP_TIMESTAMP DATE                            not null,
   DESCRIPTION          VARCHAR2(200),
   BROKER_ID            NUMBER(10),
   BROKER_COMMISSION    NUMBER(10,2),
   constraint CLIENT_TRANSACTION_PK primary key (CLIENT_TRANSACTION_ID)
);

/*==============================================================*/
/* Index: CLIENT_TRANSACTION_BROKER                             */
/*==============================================================*/
```

```
create index CLIENT_TRANSACTION_BROKER on CLIENT_TRANSACTION (
   BROKER_ID ASC
);

/*==============================================================*/
/* Index: CLIENT_TRANSACTION_CLIENT                             */
/*==============================================================*/
create index CLIENT_TRANSACTION_CLIENT on CLIENT_TRANSACTION (
   CLIENT_ID ASC
);

/*==============================================================*/
/* Index: CLIENT_TRANSACTION_INVESTMENT                         */
/*==============================================================*/
create index CLIENT_TRANSACTION_INVESTMENT on CLIENT_TRANSACTION (
   INVESTMENT_ID ASC
);

/*==============================================================*/
/* Table: INVESTMENT                                            */
/*==============================================================*/
create table INVESTMENT  (
   INVESTMENT_ID        NUMBER                          not null,
   INVESTMENT_TYPE_ID   NUMBER,
   INVESTMENT_VENDOR    VARCHAR2(30),
   INVESTMENT_NAME      VARCHAR2(200),
   INVESTMENT_UNIT      VARCHAR2(20),
   INVESTMENT_DURATION  VARCHAR2(10),
   constraint INVESTMENT_PK primary key (INVESTMENT_ID)
);


/*==============================================================*/
/* Index: INVESTMENT_INVESTMENT_TYPE                            */
/*==============================================================*/
create index INVESTMENT_INVESTMENT_TYPE on INVESTMENT (
   INVESTMENT_TYPE_ID ASC
);

/*==============================================================*/
/* Table: INVESTMENT_TYPE                                       */
/*==============================================================*/
create table INVESTMENT_TYPE  (
   INVESTMENT_TYPE_ID   NUMBER                          not null,
   INVESTMENT_TYPE_NAME VARCHAR2(30)                    not null,
   constraint INVESTMENT_TYPE_PK primary key (INVESTMENT_TYPE_ID)
);


/*==============================================================*/
/* Table: OFFICE_LOCATION                                       */
/*==============================================================*/
create table OFFICE_LOCATION  (
   OFFICE_LOCATION_ID   NUMBER                          not null,
   OFFICE_NAME          VARCHAR2(20)                    not null,
   OFFICE_ADDRESS       VARCHAR2(50)                    not null,
   OFFICE_CITY          VARCHAR2(30)                    not null,
   OFFICE_STATE_PROVINCE VARCHAR2(40)                    not null,
   OFFICE_POSTAL_CODE   VARCHAR2(10)                    not null,
   OFFICE_COUNTRY       VARCHAR2(40),
   constraint OFFICE_LOCATION_PK primary key (OFFICE_LOCATION_ID)
);
```

10.2 Moving the Oracle Database to MySQL with the Migration Toolkit

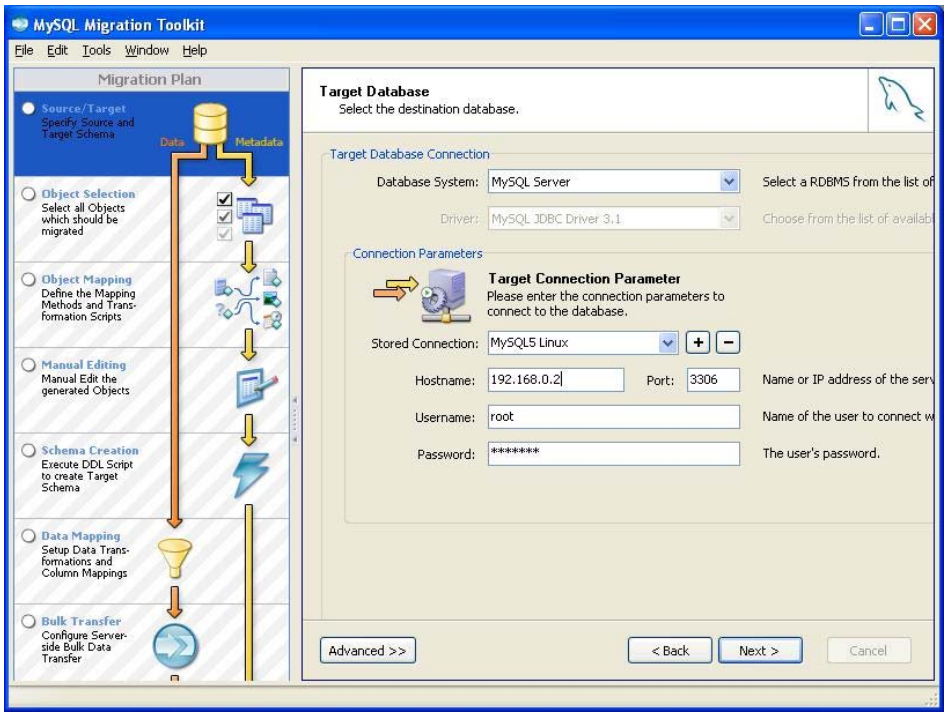
The following steps are used to move the Oracle database schema (with data) over to a MySQL 5.x database.

Step 1 – Connect to Oracle Source Database:



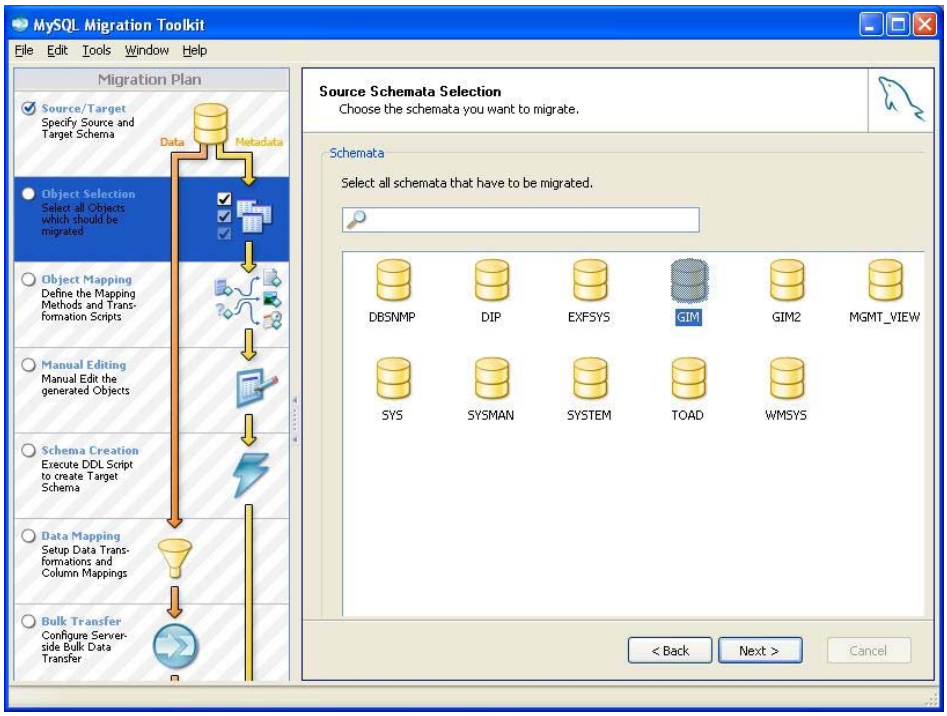
Supply an Oracle connection string along with an ID and password and connect to Oracle.

Step 2 – Connect to MySQL Target Server:



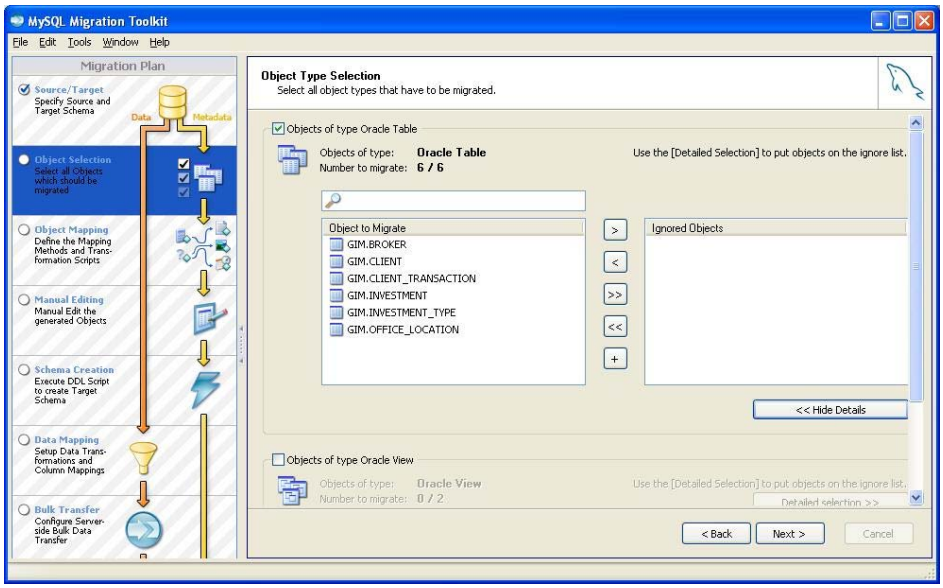
Enter MySQL connection information for the target server. The Migration Toolkit connects to both the source and target servers.

Step 3 – Select Target Schema(s) to Migrate:



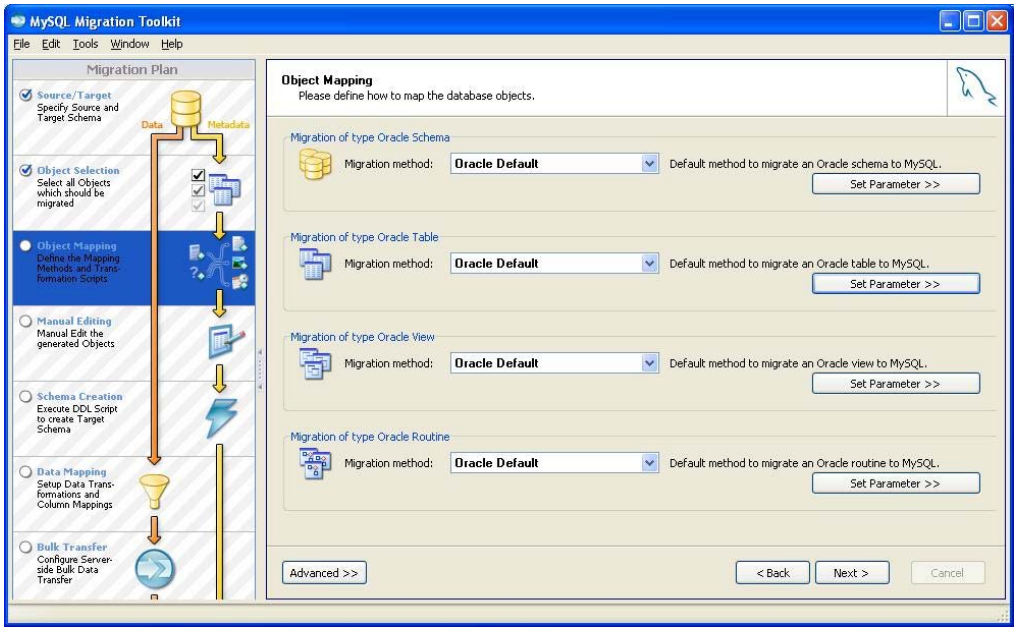
For this example, the Oracle GIM schema will be migrated. Note that more than one schema can be selected. The Migration Toolkit will reverse engineer the source database schema along with all datatypes, etc.

Step 4 (Optional) – Choose Objects to Migrate:



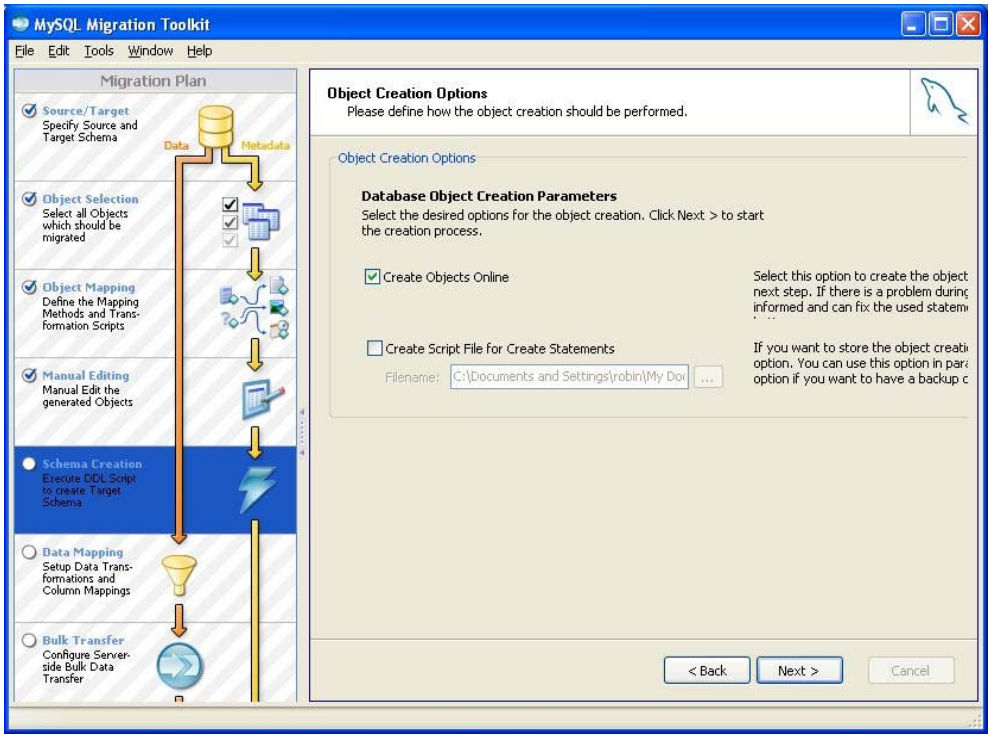
You can optionally select what objects to migrate to MySQL. The default is all data objects in the schema.

Step 5 – Tweak MySQL Defaults:

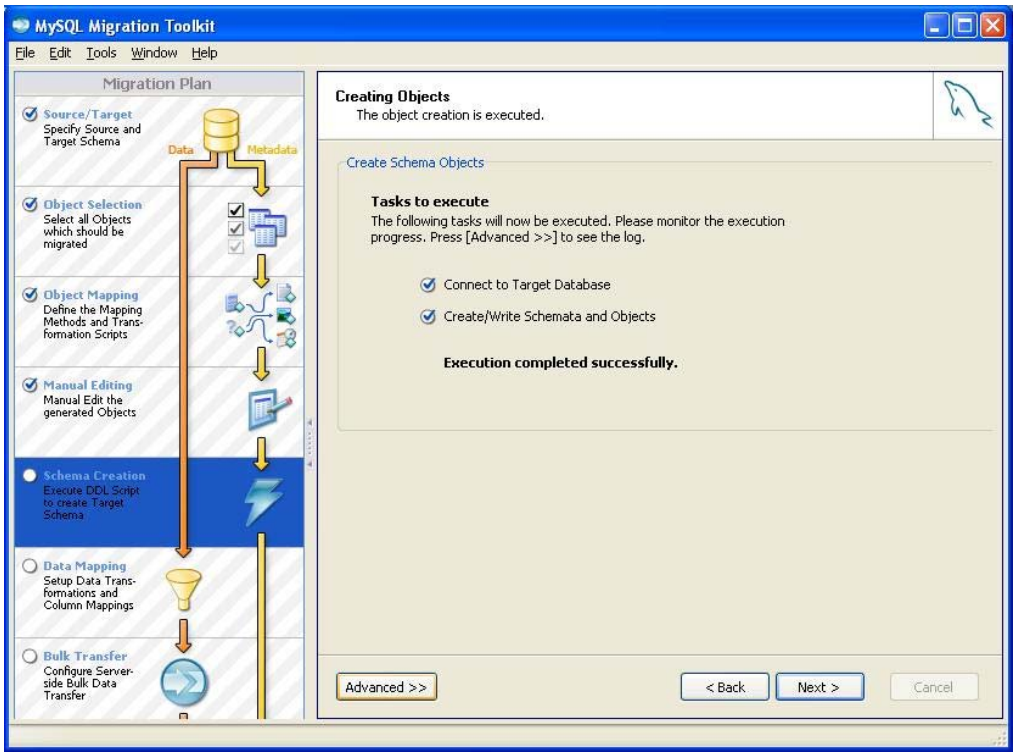


This step allows you to customize your MySQL settings like target engine, etc. Intelligent defaults are provided (such as InnoDB storage engine for Oracle databases, etc.) The Migration Toolkit will then proceed and inform you if any mapping problems are found and generate the SQL necessary to create the MySQL target database.

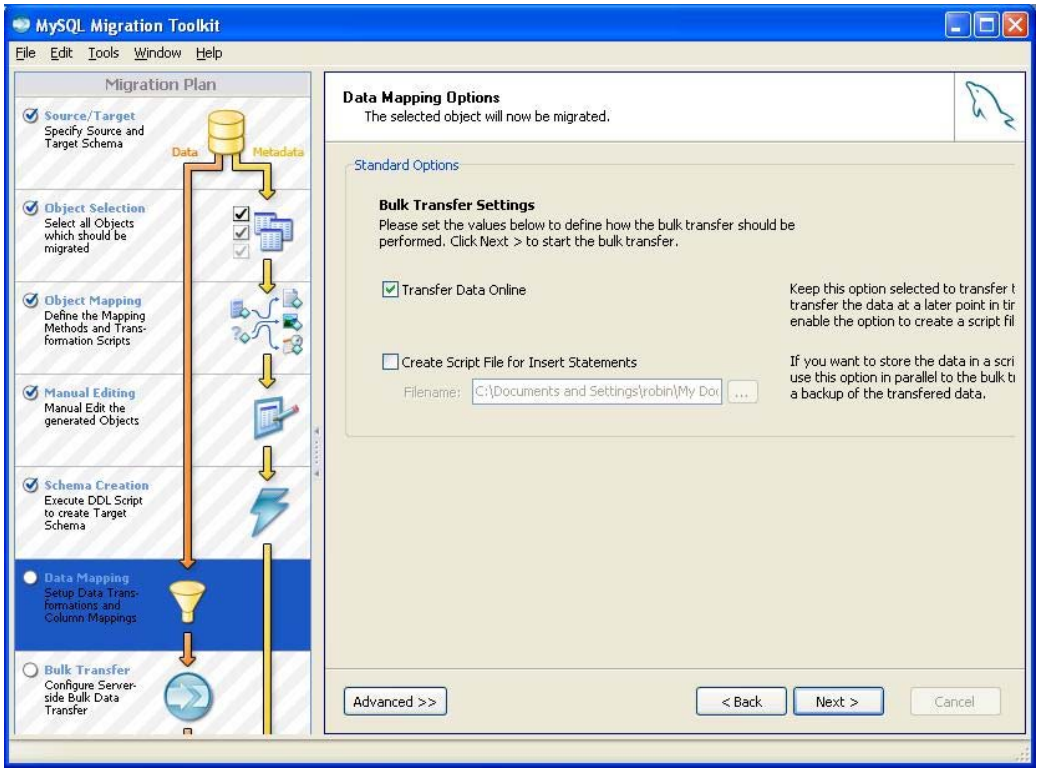
Step 6 – Create MySQL Target Objects:



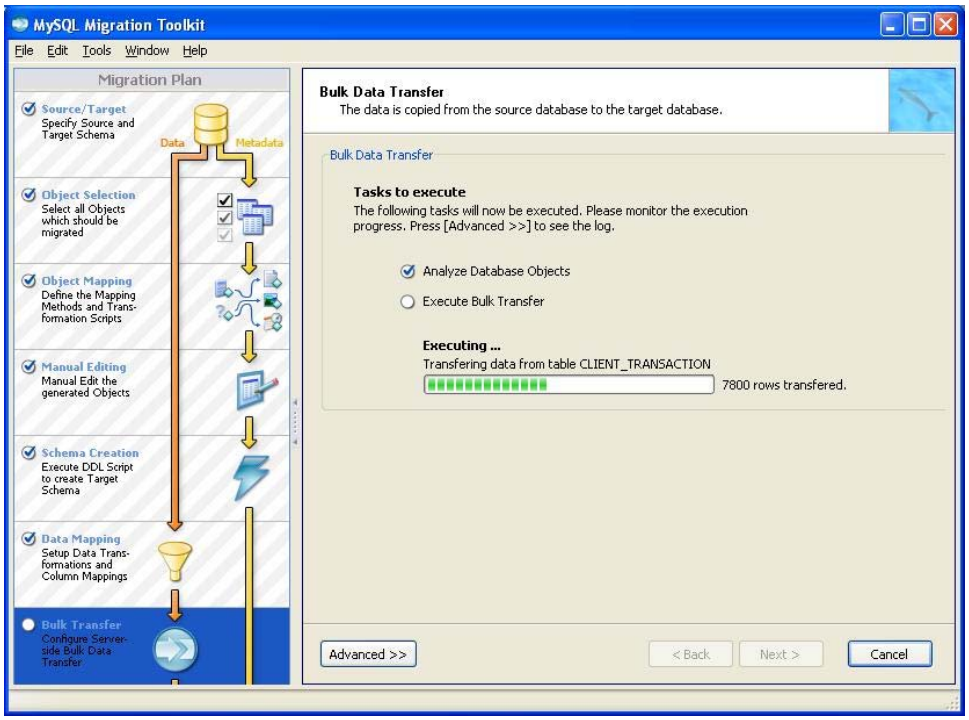
This step allows you to create the new MySQL database and object schema (with no data) online or save the SQL statements to a file. If you choose to proceed with the default online option, the Migration Toolkit will create the MySQL database and schema objects:



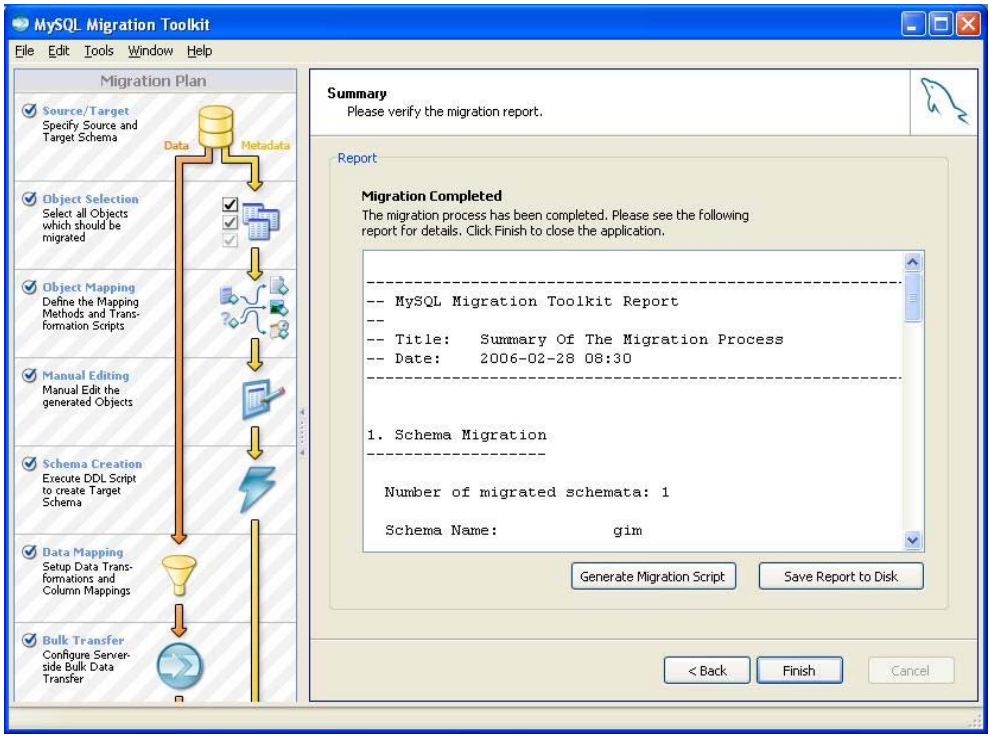
Step 7 – Migrate Existing Oracle Data to MySQL:



This step allows you to migrate the existing Oracle data to the new MySQL database schema. You can optionally choose to generate INSERT statements into a file for later execution. If you choose to proceed, the migration toolkit will move the data in bulk from Oracle to MySQL:



Step 8 – Review Migration



The final step is to review the migration process and ensure the desired results were obtained.

10.3 Sample MySQL Schema Generated from Oracle

The following MySQL database was generated from the sample Oracle10g schema:

```
DROP TABLE IF EXISTS gim.broker;
CREATE TABLE  gim.broker (
  broker_id decimal(22,0) NOT NULL,
  office_location_id decimal(22,0) default NULL,
  broker_last_name varchar(40) character set latin1 collate latin1_bin NOT NULL,
  broker_first_name varchar(20) character set latin1 collate latin1_bin NOT NULL,
  broker_middle_initial char(1) character set latin1 collate latin1_bin default NULL,
  manager_id decimal(22,0) default NULL,
  years_with_firm decimal(3,1) NOT NULL,
  PRIMARY KEY  (broker_id),
  KEY broker_location (office_location_id),
  KEY broker_manager (manager_id),
  CONSTRAINT broker_manager FOREIGN KEY (manager_id) REFERENCES broker (broker_id) ON DELETE
CASCADE ON UPDATE NO ACTION,
  CONSTRAINT broker_office_location FOREIGN KEY (office_location_id) REFERENCES office_location
(office_location_id) ON DELETE CASCADE ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS gim.client;
CREATE TABLE  gim.client (
  client_id decimal(22,0) NOT NULL,
  client_first_name varchar(20) character set latin1 collate latin1_bin NOT NULL,
  client_last_name varchar(40) character set latin1 collate latin1_bin NOT NULL,
  client_gender char(1) character set latin1 collate latin1_bin NOT NULL,
  client_year_of_birth int(4) NOT NULL,
  client_marital_status varchar(20) character set latin1 collate latin1_bin default NULL,
  client_street_address varchar(40) character set latin1 collate latin1_bin NOT NULL,
  client_postal_code varchar(10) character set latin1 collate latin1_bin NOT NULL,
  client_city varchar(30) character set latin1 collate latin1_bin NOT NULL,
  client_state_province varchar(40) character set latin1 collate latin1_bin NOT NULL,
  client_phone_number varchar(25) character set latin1 collate latin1_bin NOT NULL,
  client_household_income decimal(30,0) default NULL,
```

```
  client_country varchar(40) character set latin1 collate latin1_bin default NULL,
  broker_id decimal(22,0) NOT NULL,
  PRIMARY KEY  (client_id),
  KEY client_broker (broker_id),
  CONSTRAINT client_broker FOREIGN KEY (broker_id) REFERENCES broker (broker_id) ON DELETE
CASCADE ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS gim.client_transaction;
CREATE TABLE  gim.client_transaction (
  client_transaction_id decimal(22,0) NOT NULL,
  client_id decimal(22,0) NOT NULL,
  investment_id decimal(22,0) NOT NULL,
  action varchar(10) character set latin1 collate latin1_bin NOT NULL,
  price decimal(12,2) NOT NULL,
  number_of_units decimal(22,0) NOT NULL,
  transaction_status varchar(10) character set latin1 collate latin1_bin NOT NULL,
  transaction_sub_timestamp datetime NOT NULL,
  transaction_comp_timestamp datetime NOT NULL,
  description varchar(200) character set latin1 collate latin1_bin default NULL,
  broker_id decimal(22,0) default NULL,
  broker_commission decimal(10,2) default NULL,
  PRIMARY KEY  (client_transaction_id),
  KEY client_transaction_broker (broker_id),
  KEY client_transaction_client (client_id),
  KEY client_transaction_investment (investment_id),
  CONSTRAINT client_transaction_broker FOREIGN KEY (broker_id) REFERENCES broker (broker_id) ON
DELETE CASCADE ON UPDATE NO ACTION,
  CONSTRAINT client_transaction_client FOREIGN KEY (client_id) REFERENCES client (client_id) ON
DELETE CASCADE ON UPDATE NO ACTION,
  CONSTRAINT client_transaction_investment FOREIGN KEY (investment_id) REFERENCES investment
(investment_id) ON DELETE CASCADE ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS gim.investment;
CREATE TABLE  gim.investment (
  investment_id decimal(22,0) NOT NULL,
  investment_type_id decimal(22,0) default NULL,
  investment_vendor varchar(30) character set latin1 collate latin1_bin default NULL,
  investment_name varchar(200) character set latin1 collate latin1_bin default NULL,
  investment_unit varchar(20) character set latin1 collate latin1_bin default NULL,
  investment_duration varchar(10) character set latin1 collate latin1_bin default NULL,
  PRIMARY KEY  (investment_id),
  KEY investment_investment_type (investment_type_id),
  CONSTRAINT investment_type FOREIGN KEY (investment_type_id) REFERENCES investment_type
(investment_type_id) ON DELETE CASCADE ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS gim.investment_type;
CREATE TABLE  gim.investment_type (
  investment_type_id decimal(22,0) NOT NULL,
  investment_type_name varchar(30) character set latin1 collate latin1_bin NOT NULL,
  PRIMARY KEY  (investment_type_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS gim.office_location;
CREATE TABLE  gim.office_location (
  office_location_id decimal(22,0) NOT NULL,
  office_name varchar(20) character set latin1 collate latin1_bin NOT NULL,
  office_address varchar(50) character set latin1 collate latin1_bin NOT NULL,
  office_city varchar(30) character set latin1 collate latin1_bin NOT NULL,
  office_state_province varchar(40) character set latin1 collate latin1_bin NOT NULL,
  office_postal_code varchar(10) character set latin1 collate latin1_bin NOT NULL,
  office_country varchar(40) character set latin1 collate latin1_bin default NULL,
  PRIMARY KEY  (office_location_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

10.4 Sample Code Migration

As was mentioned, no automatic conversion exists in terms of migrating Oracle PL/SQL code objects to MySQL. MySQL is in discussions with several partners on providing this functionality, but until a mechanized solution is reached, there are several options open to performing code migration:

1. Contact MySQL Professional Services group to help with the code migration. Staff at MySQL have assisted in code conversions from a number of platforms to MySQL.
2. Perform code migrations internally.

The difficulty level of migrating Oracle PL/SQL code to MySQL depends on the actual code itself. For example, the following Oracle PL/SQL procedure uses the sample schema presented above to create various business intelligence reports to the staff of the company using the database:

```
CREATE OR REPLACE PROCEDURE GIM_ANALYSIS(   START_DATE IN DATE,
                                            END_DATE IN DATE,
                                            B_CURSOR IN OUT SYS_REFCURSOR,
                                            O_CURSOR IN OUT SYS_REFCURSOR,
                                            I_CURSOR IN OUT SYS_REFCURSOR,
                                            T_CURSOR IN OUT SYS_REFCURSOR,
                                            C_CURSOR IN OUT SYS_REFCURSOR)
AS
BEGIN

        --
        -- display brokers ordered by highest commissions for time period
        --
        OPEN B_CURSOR FOR
        SELECT  A.BROKER_ID,
                A.BROKER_FIRST_NAME,
                A.BROKER_LAST_NAME,
                SUM(BROKER_COMMISSION) TOTAL_COMMISSIONS
        FROM    BROKER A,
                CLIENT_TRANSACTION B
        WHERE   B.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
                A.BROKER_ID = B.BROKER_ID
        GROUP BY A.BROKER_ID,
                A.BROKER_FIRST_NAME,
                A.BROKER_LAST_NAME
        ORDER BY 4 DESC;

         --
        -- display offices ordered by highest commissions for time period
        --
        OPEN O_CURSOR FOR
        SELECT  C.OFFICE_NAME,
                SUM(BROKER_COMMISSION) TOTAL_COMMISSIONS
        FROM    BROKER A,
                CLIENT_TRANSACTION B,
                OFFICE_LOCATION C
        WHERE   B.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
                A.BROKER_ID = B.BROKER_ID AND
                A.OFFICE_LOCATION_ID = C.OFFICE_LOCATION_ID
        GROUP BY C.OFFICE_NAME
        ORDER BY 2 DESC;

        --
        -- display top 20 invests ordered by highest invested dollars for time period
        --
        OPEN I_CURSOR FOR
        SELECT * FROM
```

```
        (SELECT B.INVESTMENT_VENDOR,
                B.INVESTMENT_NAME,
                SUM(PRICE) * SUM(NUMBER_OF_UNITS) TOTAL_INVESTED_DOLLARS
        FROM    CLIENT_TRANSACTION A,
                INVESTMENT B
        WHERE   A.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
                B.INVESTMENT_ID = A.INVESTMENT_ID AND
                A.ACTION = 'BUY'
        GROUP BY B.INVESTMENT_VENDOR,
                B.INVESTMENT_NAME
        ORDER BY 3 DESC)
        WHERE ROWNUM < 21
        ORDER BY 3 DESC ;

        --
        -- display top 20 types ordered by highest invested dollars for time period
        --
        OPEN T_CURSOR FOR
        SELECT C.INVESTMENT_TYPE_NAME,
                SUM(PRICE) * SUM(NUMBER_OF_UNITS) TOTAL_INVESTED_DOLLARS
        FROM    CLIENT_TRANSACTION A,
                INVESTMENT B,
                INVESTMENT_TYPE C
        WHERE   A.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
                B.INVESTMENT_ID = A.INVESTMENT_ID AND
                C.INVESTMENT_TYPE_ID = B.INVESTMENT_TYPE_ID AND
                A.ACTION = 'BUY'
        GROUP BY C.INVESTMENT_TYPE_NAME
        ORDER BY 2 DESC;

         --
        -- display top 20 clients ordered by highest invested dollars for time period
        --
        OPEN C_CURSOR FOR
        SELECT * FROM
        (SELECT B.CLIENT_FIRST_NAME,
                B.CLIENT_LAST_NAME,
                SUM(PRICE) * SUM(NUMBER_OF_UNITS) TOTAL_INVESTED_DOLLARS
        FROM    CLIENT_TRANSACTION A,
                CLIENT B
        WHERE   A.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
                B.CLIENT_ID = A.CLIENT_ID AND
                A.ACTION = 'BUY'
        GROUP BY B.CLIENT_FIRST_NAME,
                B.CLIENT_LAST_NAME
        ORDER BY 3 DESC)
        WHERE ROWNUM < 21
        ORDER BY 3 DESC ;

END;
/
```

The Oracle PL/SQL procedure can be translated to MySQL as follows:

```
CREATE PROCEDURE CORPORATE_ANALYSIS
(IN START_DATE DATETIME,
 IN END_DATE   DATETIME)

        BEGIN

        /*
        display brokers ordered by highest commissions for time period
       */
```

```
SELECT  A.BROKER_ID,
        A.BROKER_FIRST_NAME,
        A.BROKER_LAST_NAME,
        SUM(BROKER_COMMISSION) TOTAL_COMMISSIONS
FROM    BROKER A,
        CLIENT_TRANSACTION B
WHERE   B.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
        A.BROKER_ID = B.BROKER_ID
GROUP BY A.BROKER_ID,
        A.BROKER_FIRST_NAME,
        A.BROKER_LAST_NAME
ORDER BY 4;

/*
-- display offices ordered by highest commissions for time period
*/
 SELECT  C.OFFICE_NAME,
        SUM(BROKER_COMMISSION) TOTAL_COMMISSIONS
FROM    BROKER A,
        CLIENT_TRANSACTION B,
        OFFICE_LOCATION C
WHERE   B.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
        A.BROKER_ID = B.BROKER_ID AND
        A.OFFICE_LOCATION_ID = C.OFFICE_LOCATION_ID
GROUP BY C.OFFICE_NAME
ORDER BY 2 DESC;

/*
-- display top 20 invests ordered by highest invested dollars for time period
*/
SELECT  B.INVESTMENT_VENDOR,
        B.INVESTMENT_NAME,
        SUM(PRICE) * SUM(NUMBER_OF_UNITS) TOTAL_INVESTED_DOLLARS
FROM    CLIENT_TRANSACTION A,
        INVESTMENT B
WHERE   A.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
        B.INVESTMENT_ID = A.INVESTMENT_ID AND
        A.ACTION = 'BUY'
GROUP BY B.INVESTMENT_VENDOR,
        B.INVESTMENT_NAME
ORDER BY 3 DESC
LIMIT 20;

/*
-- display top types ordered by highest invested dollars for time period
*/

SELECT  C.INVESTMENT_TYPE_NAME,
        SUM(PRICE) * SUM(NUMBER_OF_UNITS) TOTAL_INVESTED_DOLLARS
FROM    CLIENT_TRANSACTION A,
        INVESTMENT B,
        INVESTMENT_TYPE C
WHERE   A.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
        B.INVESTMENT_ID = A.INVESTMENT_ID AND
        C.INVESTMENT_TYPE_ID = B.INVESTMENT_TYPE_ID AND
        A.ACTION = 'BUY'
GROUP BY C.INVESTMENT_TYPE_NAME
ORDER BY 2 DESC;

/*
-- display top 20 clients ordered by highest invested dollars for time period
*/
SELECT  B.CLIENT_FIRST_NAME,
```

```
        B.CLIENT_LAST_NAME,
        SUM(PRICE) * SUM(NUMBER_OF_UNITS) TOTAL_INVESTED_DOLLARS
FROM    CLIENT_TRANSACTION A,
        CLIENT B
WHERE   A.TRANSACTION_COMP_TIMESTAMP BETWEEN START_DATE AND END_DATE AND
        B.CLIENT_ID = A.CLIENT_ID AND
        A.ACTION = 'BUY'
GROUP BY B.CLIENT_FIRST_NAME,
        B.CLIENT_LAST_NAME
ORDER BY 3 DESC
LIMIT 20;

END;
```