Mimer SQL

Migration guide from Oracle Rdb to Mimer SQL

Version 11.0

Mimer SQL, Migration guide from Oracle Rdb to Mimer SQL, Version 11.0, December 2025

Produced and published by Mimer Information Technology AB, Uppsala, Sweden.

Mimer SQL Web Sites:
https://developer.mimer.com
https://www.mimer.com

Contents

Chapter 1
Introduction

This document describes the steps needed to migrate a database and application code from Oracle Rdb, henceforth referred to as Rdb, to Mimer SQL.

Terms

This section discusses various terms referred to in this document. In particular, focus is on terms that are used somewhat differently in the two database systems.

Database

In Rdb a database is a collection of tables that logically belong together. An application may use one or several databases in an application. Tables may be joined between databases and the database name or alias is used as a schema name in SQL. For example:

```
select * from db1.table1 inner join db2.tab on table1.c1 = tab.c1
```

Here we are joining `table1` from database `db1` with table `tab` located in database `db2`.

In Mimer SQL, on the other hand, there is typically a single database containing several schemas. The equivalent use of the above SQL statement is to have one schema named `db1` and another schema named `db2`, and both of these would be located in the same Mimer SQL database.

A common case in Mimer SQL where you can have two databases is to have one for testing and one for production. The other case is if the tables in the two databases are completely separate and are never accessed together by an application.

When migrating from several Rdb databases to Mimer SQL, the tables in each schema can be owned by a single user having several schemas, or by several users. When different users own the tables, appropriate access privileges must be given to any user wishing to access the two tables together.

Storage area versus Databank

In Rdb there is a term *storage area* that describes files in the operating system used by Rdb. An Rdb database consist of many storage areas.

In Mimer SQL the corresponding term is a *databank*. In Mimer SQL there are four system databanks that are used to store data dictionary (SYSDB), handle transactions (TRANSDB), temporary storage (SQLDB) and recovery (LOGDB). The user adds his or her own databanks to the original set of system databanks. Typically, only one or a few databank files are added. The only time a databank is referenced is in create table when a table is placed in a databank.

Document Overview

Chapter 2, The conversion process contains the overall description of the conversion process to move a database and application from Rdb to Mimer SQL.

Chapter 3, Specifics goes into some depth on specific topics and considerations of the conversion process and how to design you Mimer SQL database.

Chapter 2

The conversion process

Overview

The following general steps are needed to perform a migration from Oracle/Rdb to Mimer SQL.

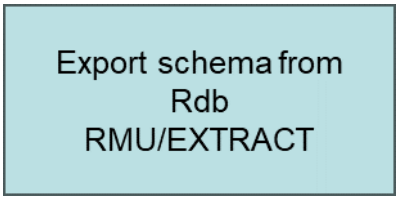
- Conversion of database schema
- Move existing data to Mimer SQL
- Conversion of application code
- Functionality review and testing
- Performance review and testing

Each of these topics will be considered in some detail in the rest of this chapter.

Conversion of database schema

Export schema

The database schema is exported from Rdb with the RMU utility. This step is repeated for each database in Rdb.




```
$ rmu/extract=inventory_rdb_meta.sql inventory
```

In this command `inventory` is the name of the database. The file `inventory_rdb_meta.sql` now contains the Rdb specific SQL statements needed to create the database.

The file contains all the application objects. However, it may also contain Oracle compatibility statements. These statements can be removed at this stage, or at a later stage.

The next step is to convert this file with the Mimer SQL translator. The translator takes the Rdb specific SQL statements and converts them to the corresponding Mimer SQL statements:

```
$ sqltranslator /rdb /mode=script inventory_meta.sql inventory_mimer_meta.sql
```

After running this command, open the file `inventory_mimer_meta.sql` in an editor and examine the results. In particular, you can search for `SYNTAX_ERROR`. This will show you the statements that `sqltranslator` was unable to handle. Either this will be statements you do not need in a Mimer SQL environment, or it is statements that must be converted manually. Contact Mimer if you find statements that would be helpful to include in the SQL translator.

When exporting the schema, sometimes various Oracle compatibility code is also exported. These should typically be removed. They are called `ORA_OUTPUT`, `ORA_TRANS`, `ORA_MISC3` etc. In particular some errors occur when creating these objects, as some objects are created in the wrong order and as there are also references to system tables that are not present in Mimer SQL. You can safely ignore these errors.

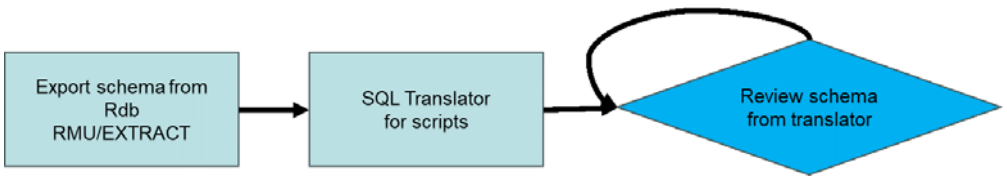
Create schema in Mimer SQL

At this point we want to create the exported and converted schema in Mimer SQL. To do so you must have a Mimer SQL database up and running. Please see the *Mimer SQL OpenVMS Guide*, section *Establishing a database* (https://docs.mimer.com/MimerOnVms/v110/html/VMS_Net/Establishing/Establishing.htm).

After setting up the Mimer SQL database server, run `BSQL` with the following commands:

```
$ BSQL
Username: SYSADM
Password:
SQL>CREATE IDENT APPOWNER AS USER USING 'APASSW';
SQL>CREATE DATABANK APPDB;
SQL>GRANT SCHEMA, SEQUENCE TO APPOWNER;
SQL>GRANT TABLE ON DATABANK APPDB TO APPOWNER;
SQL>GRANT SEQUENCE ON DATABANK APPDB TO APPOWNER;
```

Substitute `APPOWNER` above with the name of your choice and another password. Now log in to the user that will own the schemas and tables you migrate from Rdb.



If you, for example, have two Rdb databases called `INVENTORY` and `STOCK` you create each one under a separate schema in Mimer SQL.

```
$ BSQL
Username: APPOWNER
Password:
SQL>CREATE SCHEMA INVENTORY;
SQL>CREATE SCHEMA STOCK;
SQL>SET LC 0;
SQL>WHENEVER ERROR CONTINUE;
SQL>LOG INPUT, OUTPUT ON 'inventory_mimer_meta.log';
SQL>SET SCHEMA INVENTORY;
SQL>READ INPUT FROM 'inventory_mimer_meta.sql';
```

The `SET LC` (Line-Count) to zero is used when reading input files in `BSQL`.

The `SET SCHEMA` command will let unqualified references in the `inventory_mimer_meta.sql` file default to the `INVENTORY` schema.

So, for example, `CREATE TABLE ABC` will be treated as `CREATE TABLE INVENTORY.ABC`, and so on.

Index analysis

After the objects have been created it is time to do an index analysis. This is because index handling varies somewhat between Mimer SQL and Rdb.

In Mimer SQL many database objects implicitly create indexes. For example, if a table has a primary key there is no need to have a UNIQUE index on the primary key columns. In fact, this will result in loss of performance and disk space as the UNIQUE index is maintained unnecessarily.

The same is true for unique constraints and foreign keys. Indexes are implicitly created for these objects.

Mimer SQL has a utility called `DbAnalyzer` which goes through the database and looks at existing explicit and implicit indexes and recommends appropriate actions. The utility can also find when indexes are overlapping and recommend which index to keep, and which one to drop.

The output created by `DbAnalyzer` can actually be used as an input file to `BSQL` as follows:

```
$ DEFINE/USER SYS$OUTPUT ANALYZE.TXT
$ DBANALYZER /user=APPOWNER /password=APASSW
$ BSQL
Username: APPOWNER
Password:
SQL>SET LC 0;
SQL>WHENEVER ERROR CONTINUE;
SQL>LOG INPUT, OUTPUT ON 'ANALYZE.LOG';
SQL>READ INPUT FROM 'ANALYZE.TXT';
```

Note that any duplicate indexes are removed through this process.

`ANALYZE.TXT` should be reviewed, as it returns interesting information about the schema before applying the changes.

`DbAnalyzer` by default analyzes all objects created by the login user. The switch `/SCHEMA` can be used to look at a specific schema owned by the user. For example:

```
$ DBANALYZER /user=APPOWNER /password=APASSW /schema=inventory
```

Move existing data to Mimer SQL

```
$ rmu/unload/RECORD_DEFINITION=(FORMAT=DELIMITED_TEXT, PREFIX="",
SUFFIX="|", separator="", null="\_|") INVENTORY ABC ABC.TXT
```

In this command `INVENTORY` is the Rdb database name, `ABC` is the table name, and `ABC.TXT` is the text file with the data from the table.

Note: This command will only work if there is no data with the delimiter character '`|`'. If you use the character in your data, pick another delimiter character that is not used.

Use `mimload` to load the data into Mimer SQL:

```
$ mimload -uAPPOWNER -p"APASSW" "load from loadabc.txt', 'abc.txt' log
'abc_load.log' "
```

The file `loadabc.txt` contains the following:

```
#data
    column separator '|'
    null indicator '\-'
    using insert into inventory(c1,c2,c3) values(?,?,?);
```

This process is repeated for each table migrated. The insert command can be moved to the `mimload` command line. However, command line lengths may interfere with long insert statements in this case. The USING syntax in `mimload` is then used.

Rdb migration scripts

To ease the migration from Rdb to Mimer SQL a set of DCL scripts can be provided. These scripts automate the extraction of schema and data from Rdb and the creation and optimization of the schema in Mimer SQL as well as loading the data into Mimer SQL.

The entire migration can be performed on a single machine that has both Mimer SQL and Rdb installed, or it can be done on separate machines.

The script `unload_rdb.com` is executed on the OpenVMS machine that have Rdb installed. To execute it, run `@unload_rdb <path to Rdb database> <schema>`, for example `@unload_rdb DKA0:[db]inventorydb inventory`.

When the unload is finished, the migration and loading of the schema and data into Mimer SQL is performed by running:

```
@load_mimer <SYSADM password> <schema> [<Mimer SQL user> <Mimer SQL password]
```

For the specified Mimer SQL user, a schema will be created, and all database objects will be created within that schema.

Multiple Rdb databases can be unloaded and loaded using the same Mimer SQL user but with different schema names. The schema corresponds to the name given by the `declare alias` statement used with the Rdb database.

To handle objects that need to be manually migrated or to execute other custom SQL, the `load_mimer.com` script will look for sql files in the `[.extra_sql]` directory and execute them if found. This can be used, for example, to create triggers that could not be automatically converted to Mimer SQL.

The `load_mimer.com` script will perform the following steps:

1. Run sqltranslator on the Rdb SQL schema to make it compatible with Mimer SQL.
2. Create the Mimer SQL user if it does not exist.
3. Create a databank where database objects will be stored.
4. Create the Mimer SQL schema for the migrated Rdb database.
5. Execute the translated SQL schema file using Mimer SQL.
6. Run dbanalyzer and apply the suggested changes on the created schema to optimize the database structure.
7. If `[.extra_sql]<schema>-system-after-create.sql` or `[.extra_sql]< schema>-after-create.sql` exist, execute them to run custom SQL, such as changing table or databank definitions.
8. Load each table that contains data.
9. If `[.extra_sql]<schema>-after-load.sql` exists, execute it to run custom SQL, such as creating manually converted triggers.
10. Update database statistics for the Mimer SQL database to ensure efficient query execution.

For more details, see the README file in the script distribution.

Conversion of application code

The SQL translator used to convert the SQL script generated by RMU can also be used to convert embedded SQL application code. In addition, if module SQL is used, the SQL contained in the modules SQL file can also be converted by the SQL translator.

The intention is for the SQL translator to be used once to convert the source code. The output file will be your new main source that you continue to work and develop.

The different languages are covered in the following sections.

C

Embedded Rdb C files typically use extension `.SC`. Mimer SQL Embedded C uses extension `.EC`.

```
$ sqltranslator /C sourcefile.SC sourcefile.EC
$ esql /C sourcefile.EC
$ CC sourcefile.C
```

DECLARE SECTION

When running the embedded SQL preprocessor, you may find that the host variables referenced in the SQL statements in the file will not be found. The reason is that `BEGIN DECLARE SECTION` and `END DECLARE SECTION` are missing.

These have to be added to the `.EC` file at appropriate places to guide the preprocessor. For example:

```
EXEC SQL BEGIN DECLARE SECTION;
    double dblvar;
EXEC SQL END DECLARE SECTION;
```

These statements can surround any number of variable declarations as long as they are understood and supported by the esql preprocessor.

Cobol

The SQL Translator and the Mimer SQL Embedded preprocessor can handle Cobol source both in *standard format*, and in the so-called *terminal format*.

On OpenVMS, Cobol files are by default in terminal format.

Embedded Rdb Cobol files typically use extension `.SCO`. Mimer SQL Embedded uses extension `.ECO`.

If you need to run the SQL translator several times on the same file, it is also possible to add the statements to the original `.SCO` file rather than the `.ECO` file.

Terminal format

When terminal Cobol format is used, Area A begins in position 1 and Area B begins in position 5, and the source program records do not have line numbers.

So, for terminal format the following is used:

```
$ sqltranslator /COBOL /FORMAT=TERMINAL sourcefile.SCO newsourcefile.ECO
$ esql /COBOL /FORMAT=TERMINAL newsourcefile.ECO
$ cobol /NOANSI_FORMAT sourcefile.COB
```

Standard format

When standard Cobol format is used, positions 1-6 are used for sequence numbers, position 7 for indicators, and 72-80 is the identification area.

For standard format the following would be used:

```
$ sqltranslator /COBOL /FORMAT=FIXED sourcefile.SCO targetfile.ECO
$ esql /COBOL /FORMAT=FIXED sourcefile.ECO
$ cobol /ANSI_FORMAT sourcefile.COB
```

DECLARE SECTION

When running the embedded SQL preprocessor, you may find that the host variables, referenced in the SQL statements in the file, will not be found.The reason is that `BEGIN DECLARE SECTION` and `END DECLARE SECTION` are missing.

These have to be added to the `.ECO` file at appropriate places to guide the preprocessor. For example:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
   03 VARIABLEX PIC 9(08).
EXEC SQL END DECLARE SECTION END-EXEC.
```

These statements can surround any number of variable declarations as long as they are understood and supported by the esql preprocessor.

Fortran

Embedded Rdb Fortran files typically use extension .SFO. Mimer SQL Embedded Fortran uses extension .EFO.

```
$ sqltranslator /FORTRAN sourcefile.SFO sourcefile.EFO
$ esql /FORTRAN sourcefile.EFO
$ fortran sourcefile.EFO
```

DECLARE SECTION

When running the embedded SQL preprocessor, you may find that the host variables referenced in the SQL statements in the file will not be found. The reason is that `BEGIN DECLARE SECTION` and `END DECLARE SECTION` are missing.

These have to be added to the .EFO file at appropriate places to guide the preprocessor. For example:

```
EXEC SQL BEGIN DECLARE SECTION
CHARACTER*36 ALPNUM
EXEC SQL BEGIN DECLARE SECTION
```

These statements can surround any number of variable declarations as long as they are understood and supported by the esql preprocessor.

Module SQL

When Module SQL is used the SQL statements are stored in a separate file. The application calls the statements for each operation, such as `open cursor`, `fetch`, and so on.

In Mimer SQL, the Module SQL file is converted to Embedded C and then to C. So, a C compiler is required to use Module SQL with Mimer SQL.

Module SQL can thus be used for languages that are able to call routines in C. The calling language is declared in the Module SQL file.

Functionality review

At this point both schema and data have been moved from Rdb to Mimer SQL. The application code has been converted and it is time to test it.

If there exists unit tests or other test frameworks these are, of course, used to test the integration with Mimer SQL.

Testing the application requires all SQL statements to be executed.

In particular error handling should be reviewed. This is discussed further in *Error handling* on page *13*.

A difference with regard to concurrency is that in Rdb applications a deadlock may occur at any statement. The Mimer SQL equivalent is that a transaction is aborted due to a conflict with another transaction executing concurrently. In Mimer SQL, this error occurs at commit time. The application should thus test for this error code (-10001) after commit, and retry the transaction when this occurs.

Performance review

When an application works as intended, it is also important to look at the performance of the application.

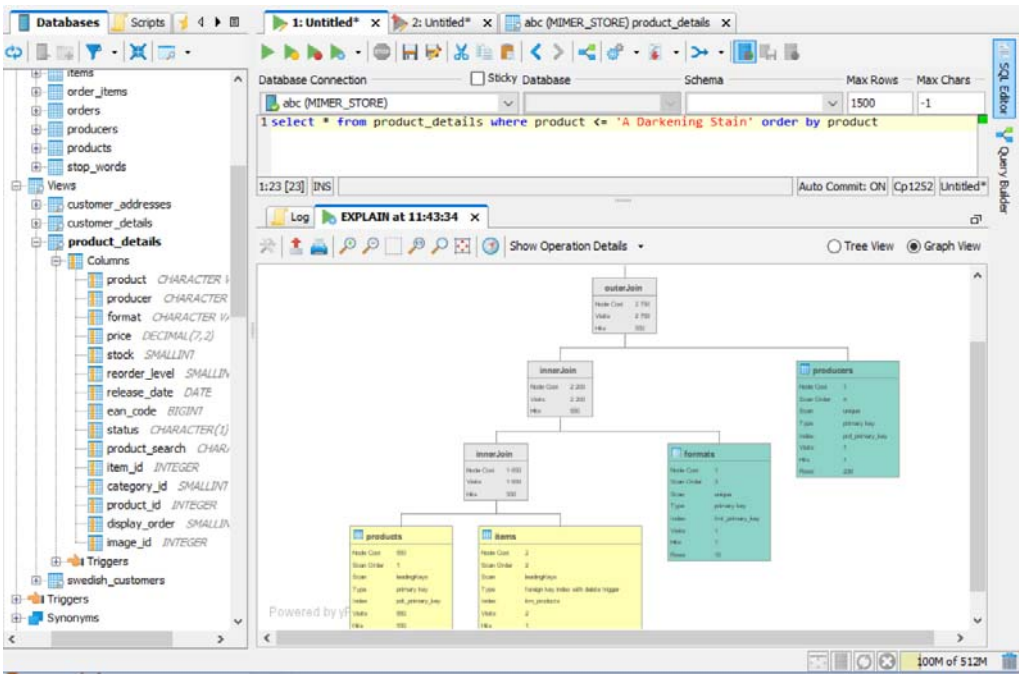
Explain

In particular, each SQL statement containing a query should be reviewed with regards to performance.

Note: After the data has been loaded into Mimer SQL you should run the `UPDATE STATISTICS` statement. This will give the SQL optimizer information about the data in the database. This is used to select indexes and greatly affects the performance of SQL statements. This is discussed further in the *Mimer SQL System Management Handbook*, chapter *Database Statistics* (https://docs.mimer.com/MimerSqlManual/v110/html/Manuals/Statistics/Statistics.htm).

Apart from running each SQL statement and measuring how long it takes to execute, it is also possible to look at the optimizer's explain plan for each query.

This can be done in both a graphical environment using DbVisualizer (on Linux or Windows etc), or in a command-line based environment with the `BSQL` utility.



In the DbVisualizer picture above green means complete primary key access, and yellow means leading keys use indexed access. If anything is red, it means expensive and may include a sequential scan of the whole table. It this is not intentional an index or a reformulated query may be needed to speed up the query. The gray nodes are different type of join nodes.

In `BSQL` the query plan is shown after `SET EXPLAIN ON` is given. The plan is displayed in XML format.

Please see *Explain for Mimer SQL 11*, https://developer.mimer.com/article/explain-for-mimer-sql-11/, for more information.

Monitoring

There are both graphical and command-line based tools to monitor the SQL execution in the Mimer SQL server. So, when an application is running it is possible to examine which SQL statements are using the most resources. The command line tool is called `sqlmonitor` and the graphical tool is called Mimer SQL Monitor and is installed with the Mimer SQL Data Provider on Windows.

In addition, DbAnalyzer, with the `/STATISTICS` option, can be used to view index usage statistics.

Chapter 3
Specifics

In this chapter we discuss various topics in detail. They may only be of interest in some conversion scenarios depending on the functionality used by the application.

Storage layout and databanks

The layout of storage is rather complex in Rdb with many files etc. In Mimer SQL on the other hand, the layout is simple.

A table and associated indexes are stored in a databank. A databank corresponds to a file in the operating system. Many tables are typically stored in the same databank.

A databank is a unit of backup and restore. So, all tables in a databank are backed up together.

A databank may, if it is large, be stored in several files. This allows the databank to be stored on more than one disk thus allowing greater I/O throughput. When several files are used, the data distribution between the files are handled automatically by Mimer SQL.

Which databank is used is decided in the `CREATE TABLE` command. For example:

```
create databank DBNAME set file 'DISK1:[DIR]DBNAME.DBF' option LOG;
create table TAB1(C1 int) in DBNAME;
```

This means that this kind of structuring is a good idea to perform when migrating the data. If no databank is specified the system will pick one for you.

For more information, please see:

`CREATE DATABANK` -
https://docs.mimer.com/MimerSqlManual/v110/html/Manuals/SQL_Statements/CREATE_DATABANK.htm

`ALTER DATABANK` -
https://docs.mimer.com/MimerSqlManual/v110/html/Manuals/SQL_Statements/ALTER_DATABANK.htm

`CREATE TABLE` -
https://docs.mimer.com/MimerSqlManual/v110/html/Manuals/SQL_Statements/CREATE_TABLE.htm

Error handling

In Rdb the routine `SQL$GET_ERROR_TEXT` is used to retrieve error messages. Mimer provides the source for an implementation of this routine towards Mimer SQL. The routine uses the SQL Standard `GET DIAGNOSTICS` statement to retrieve the error texts.

Applications that test specific error codes must be updated to use the corresponding error code in Mimer SQL. Both systems use sqlcode for this. You can inspect you code by looking for sqlcode to see if explicit error codes are used.

Here are some interesting Mimer SQL error codes that may be used in applications:

Error code	Explanation
-10001	Transaction aborted due to conflict with other transaction. Can occur after: EXEC SQL COMMIT
-10101	PRIMARY KEY constraint violated, attempt to insert duplicate key in table
-10105	Referential constraint violated INSERT/UPDATE operation not valid for table
-10106	Referential constraint violated UPDATE/DELETE operation not valid for table
-10110	UNIQUE constraint violated for table
0	Success
100	End of table/query

For more error code texts, please see the M*imer SQL Programmer's Manual*, *Appendix Return Code*, (https://docs.mimer.com/MimerSqlManual/v110/html/Manuals/App_Return_Codes/App_Return_Codes.htm

Triggers

Triggers in Rdb are limited in what they can do. Because of this, triggers are sometimes used with external stored procedures. These are written in a native language.

With Mimer SQL, external procedures are not allowed to execute within the database server. So, external procedures need to be rewritten as Stored SQL procedures. Mimer SQL does not have any limitations with regard to the SQL that can be used in a stored procedure.

The same reasoning applies to external stored procedures in general, and not only when they are used as triggers in Rdb.

Mimer SQL supports before row triggers and before and after statement triggers. *Before* triggers, both **row** and **statement** triggers, are only allowed to query other tables or views in Mimer SQL.

After statement triggers can be used to both query and modify objects in the database. So, the general conversion rules are:

- Any trigger in Rdb that modifies the database with insert, update, and/or delete statements is converted to *after* **statement** triggers.
- Before and after row triggers in Rdb are converted to *before* **row** triggers in Mimer SQL.

Dynamic SQL

Much of the discussions in this document focus on how to translate from the Rdb SQL dialect to Mimer SQL. If the SQL is constructed by the application in runtime, the SQL may need to be adjusted to conform to the Mimer SQL syntax. For the most part, this is easy, as the two products actually overlap to a very high degree.

One way of doing this is to run the application and extract the generated SQL with a debugger or similar. These SQL statements can then be fed into the SQL translator to determine the correct syntax. The application can then be modified appropriately so it generates correct Mimer SQL syntax directly.

Table record length

Rdb has maximum record length of 65 272 bytes. All the data in a record must fit within this restriction.

Mimer SQL has a maximum record length of 32 000 bytes. However, this does not include large object columns of type CLOB, NCLOB or BLOB (character/national character/binary large object). Each LOB column only occupies 19 bytes of the record length.

So even if the maximum record length is somewhat larger in Rdb, Mimer SQL can handle longer records than Rdb with appropriate use of large object columns.

CHAR, VARCHAR, BINARY and VARBINARY columns have a limit of 15 000 bytes. NCHAR and NVARCHAR can be at most 5 000 characters, while BLOB, CLOB and NCLOB can have any size as maximum.

I.e. the data of a single record can have virtually any size using large objects.

Character sets

In Mimer SQL there are two character data groups. The first one is CHAR, VARCHAR and CLOB. The second one is NCHAR, NVARCHAR and NCLOB.

Data in the CHAR-group can be used to store Latin-1 data. So, only the 8-bit characters in the ISO 8859-1 standard can be stored in columns with this data type. See https://en.wikipedia.org/wiki/ISO/IEC_8859-1.

In the NCHAR-group, any Unicode data can be stored. This means that characters from any language can be stored in these columns.

When data is moved to or from the application and the database server the data is converted. For example, assume the application is using a Greek character set for single byte characters in the application. When storing these characters in Mimer SQL, NVARCHAR would be used to store the data in the database as Greek has a number of letters that are outside Latin-1. If the data is retrieved from a Windows system the Greek letters will be automatically displayed correctly there.

However, it is not possible to set the locale to German and access the Greek data from the NVARCHAR column. The letters have no representation in this locale. Fortunately, the application can use a wide character type, such as wchar_t in C, to access any data in an NVARCHAR column.

In summary, this means that character sets are not part of the SQL language in Mimer SQL. It is much easier to simply reference NCHAR data and not have to convert between different incompatible character sets.

If character sets are used by the Rdb application that is converted, it is likely that character columns should be converted to NVARCHAR rather than VARCHAR to be able to hold the wanted characters.

Applications that only use a single locale can retain their use of char-data with the correct locale in the client. To handle multiple locales, wchar_t or a single byte character set based on Unicode, such as UTF-8, are the logical choices when moving ahead.

Primary keys

The storage mechanism in Mimer SQL is particularly good at handling primary keys. It is therefore better to have a primary key than, for example, a unique index.

The difference between the two from an application perspective, is that primary key columns are NOT NULL.

So, if you have tables with no primary key, but with a unique index, you should consider whether the index columns can all be made NOT NULL as this would allow the index to be a primary key instead.

The DbAnalyzer will suggest converting UNIQUE to PRIMARY KEY if the null attributes allow this.

DATE VMS

The Mimer SQL data type `BUILTIN.DATE_VMS` is used to support the Rdb data type `DATE VMS`. To get full benefit of this type, both the client and the server must support `BUILTIN.DATE_VMS`. When an older client version is used, the type is treated the same as TIMESTAMP(2) by the client.

The current implementation supports the following languages: English, Austrian, Canadian, Danish, Dutch, Finish, German, Italian, Portuguese, Spanish, Swedish, Swiss German and Swiss French.

On OpenVMS the language is set by the `SYS$LANGUAGE` logical name. The name can be overridden with the `MIMER_LANGUAGE` logical name. If no language is selected ENGLISH is used as default. English is also always available as input language in addition to the selected language.

On other platforms than OpenVMS, the environment variable `MIMER_LANGUAGE` is used to select the language. Use upper case language name, with underscore for `SWISS_GERMAN` and `SWISS_FRENCH`.

In the current implementation the same language should be used in both server and client. In practice this means only a single language can be selected for a given database. In the future, the client setting will affect the language used by the server for this connection, thus allowing language to be selected on a client basis.

In version 11.0 this data type is fully supported by the embedded SQL client. Other clients treat the data type as a TIMESTAMP(2) column.

Index