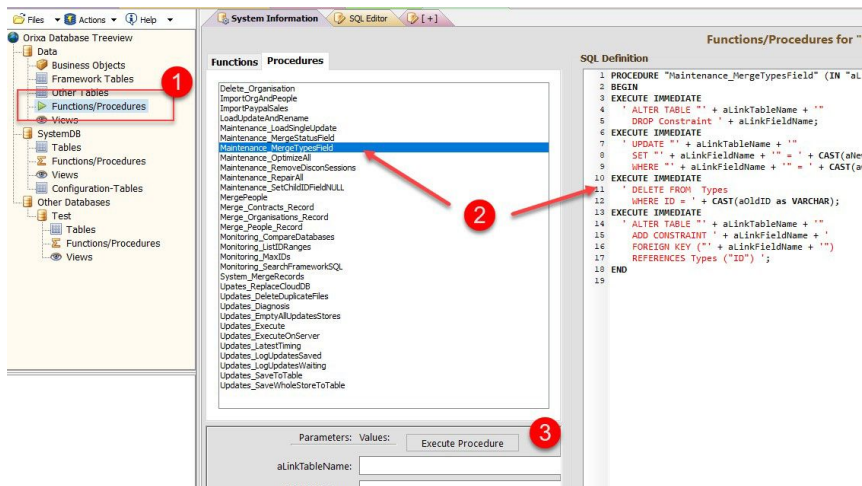


# Basics of SQL Functions and Procedures

SQL is a well developed procedural language for writing scripts which can interact with your database. There are many links on-line which give deep and detailed explanations of how to write Procedural SQL. This short document is intended to give a basic overview and introduction of some key parts of the SQL that make up a Function or Procedure to help Orixia Developers extend their Apps.

## Open the Database Management Utility to view details of a Procedure

To review and edit the Functions and Procedures in your App, go to the "System" menu, and select "View Database Management Utility", the screen showed in the image below should open.



Viewing the details of a Procedure

1. Select the "Functions/Procedures" heading in the Database Treeview.
2. Click on the name of a procedure (note that Procedures and Functions are held in separate lists, so click on the appropriate heading to see what you need). The SQL Definition of the procedure will then be shown at the far right-hand side of the screen.
3. Once a Procedure is selected a panel will list its parameters and allow you to Execute the procedure if you wish to.

## Basic Elements of a Procedure

All Procedures and Functions contain SQL enclosed by "BEGIN" "END" sections. It is normal to DECLARE a number of variables, PREPARE and OPEN Cursors and FETCH data from the cursor.

In SQL the end of a line is set with a semi-colon. If a SQL-string is added it is included in single-quotes as shown in the examples, and will display red in the editor. Note that once a single-quote has been added the SQL-string can continue over several lines. The end of a line does not end the SQL-string. To add local variables or parameters to a SQL-string use the plus sign (+) to concatenant parts of a SQL-string together, as shown in the examples.

A common error when first writing SQL-strings within Procedures occurs when the SQL-string itself contains a single quote. In such cases the single-quote must be typed twice. This is called "escaping" the single quote, and allows single quotes to be included as part of the SQL-string.

1. Use one or more DECLARE statements to make variables available in your procedure. These variables can be simple data types (like FLOATs and VARCHARs) identical to the data types of a database, or they can be Statements and Cursors. Statements and Cursors are variables which can be used to SELECT and FETCH data.
2. Once a Statement has been declared, the SQL syntax:  
PREPARE [StatementName] FROM

```

1 PROCEDURE "Monitoring_MaxIDs" ()
2 BEGIN
3   DECLARE Crsr CURSOR WITH RETURN FOR Stmt;
4   DECLARE TablesCrsr CURSOR FOR TablesStmt;
5   DECLARE TableName VARCHAR;
6   DECLARE SQLStr VARCHAR;
7   PREPARE TablesStmt FROM
8   ' SELECT
9     TableName
10    FROM Information.TableColumns
11    WHERE Name = 'ID'
12    ORDER BY TableName ';
13   OPEN TablesCrsr;
14   FETCH FIRST FROM TablesCrsr('TableName') INTO TableName;
15   SET SQLStr = ' SELECT
16     ''' + TableName + ''' as TableName,
17     MAX(ID) as MaxID,
18     COUNT(*) as RecordCount
19    FROM ''' + TableName + ''' ';
20   WHILE NOT EOF(TablesCrsr) DO
21     FETCH NEXT FROM TablesCrsr('TableName') INTO TableName;
22     SET SQLStr = SQLStr + ' UNION ' + ' SELECT
23       ''' + TableName + ''' as TableName,
24       MAX(ID) as MaxID,
25       COUNT(*) as RecordCount
26    FROM ''' + TableName + ''' ';
27   END WHILE;
28   CLOSE TablesCrsr;
29   PREPARE Stmt FROM SQLStr;
30   OPEN Crsr;
31 END
32

```

#### DECLARE, PREPARE, FETCH

- ' SELECT [Fields] FROM [Tables]';  
Can be used, coupled with the statement:  
OPEN [CursorName];  
Once this step has been done a Cursor will be available in the Procedure which the programmer can use to manipulate data.
- If you want to manipulate data returned by a Cursor, use the syntax:  
FETCH FIRST FROM  
[CursorName]([FieldName]) INTO  
[VariableName].  
You can combine this with:  
WHILE NOT EOF([CursorName]) DO  
FETCH NEXT FROM ...  
To iterate through all the records returned by a Cursor. Note that "EOF" stands for "End Of File".
- Finally, at the end of this procedure, the SQL Syntax "OPEN Crsr" is used. In such a situation the procedure will return a dataset, which can be viewed by the developer.

```

1 PROCEDURE "Updates_SaveToTable" (IN "aFileName" VARCHAR, IN "aStoreName"
2 BEGIN
3 EXECUTE IMMEDIATE
4 ' CREATE TEMPORARY TABLE SaveUpd
5   FROM Updates ''' + aFileName + ''' IN STORE ''' + aStoreName + ''';
6
7 IF NOT aKeepExistingRecords THEN
8 EXECUTE IMMEDIATE
9 ' DELETE FROM SystemDB.SavedUpdates ';
10 END IF;
11 EXECUTE IMMEDIATE
12 ' INSERT INTO SystemDB.SavedUpdates
13   (FileName, TableName, UpdateType,
14    DateCreated, Manifest, KeyData,
15    RowData)
16   SELECT
17     ''' + aFileName + ''' as FileName,
18     TableName, UpdateType,
19     UpdateTimeStamp as DateCreated, Manifest, KeyData,
20     RowData FROM SaveUpd ';
21 EXECUTE IMMEDIATE
22 ' DROP TABLE SaveUpd ';
23 END
24

```

#### Parameters, EXECUTE IMMEDIATE

It is often useful to pass inputs from a user into a Procedure or Function. To do this add parameters to the Procedures definition

- Parameters are declared directly after the Procedure name, wrapped in brackets.  
Parameters are given a data-type when they are declared, and also marked as "IN" "OUT" or "INOUT". An "IN" parameter will need to be provided by the user, and OUT parameter will be returned by the procedure.
- It is common to use the SQL Syntax:  
EXECUTE IMMEDIATE  
' [SOME SQL Statement] ';

```

PREPARE Stmt FROM
' SELECT
  Name,
  CreatedOn
FROM Configuration."Updates"
ORDER BY CreatedOn ';
OPEN Crsr;
FETCH FIRST FROM Crsr ('Name') INTO FileName;
WHILE NOT EOF(Crsr) DO
  IF NOT (FileName IS NULL) OR NOT (FileName='') THEN
    --iterate the file-list and copy / rename the tables.
    EXECUTE IMMEDIATE
      'COPY FILE ''' + FileName + '.EDBUpd'
      IN STORE Uploads TO ''' + FileName + '.EDBUpd'
      IN STORE ServerUploads';
    EXECUTE IMMEDIATE
      'RENAME FILE ''' + FileName + '.EDBUpd'
      IN STORE Uploads TO ''' + FileName + '.OLDUpd'';
  END IF;
  FETCH NEXT FROM Crsr ('Name') INTO FileName;
END WHILE;
CLOSE Crsr;

```

#### Configuration tables, COPY FILE

Procedures can be used to manage external data outside the database. To help with this there are a number of "Configuration" and "Information" Tables which can be queried with Statements and Cursors, to provide access to files on-disk.

SQL Syntax includes a number of keywords which can be used to manipulate, copy, delete and alter files on-disk. Full details of this are beyond the scope of this document.

- The statement references the Configuration "Updates" system-table, which contains a list of Updates waiting to run on the database.
- The statement uses the COPY FILE syntax to move an Update file from a Local Store to a Server Store on the cloud.

#### SQL DEMO

```

1 FUNCTION "OID" (IN "aLinkTable" VARCHAR COLLATE "ANSI")
2 RETURNS INTEGER
3 BEGIN
4   DECLARE Crsr CURSOR FOR Stmt;
5   DECLARE Result INTEGER;
6   DECLARE ID VARCHAR;
7
8   PREPARE Stmt FROM
9   ' SELECT
10    CAST(ID as VARCHAR) as ID,
11    OID
12    FROM "SystemDB"."OIDGenerator"
13    WHERE LinkTable = ? ';
14   OPEN Crsr USING aLinkTable;
15   IF ROWCOUNT(Crsr) = 0 THEN
16     EXECUTE IMMEDIATE
17     'INSERT INTO "SystemDB"."OIDGenerator"
18     (OID, LinkTable)
19     VALUES
20     (1000, '' + aLinkTable + '');
21   END IF;
22
23   FETCH FIRST FROM Crsr ('ID') INTO ID;
24   FETCH FIRST FROM Crsr ('OID') INTO Result;
25
26   EXECUTE IMMEDIATE
27   ' UPDATE "SystemDB"."OIDGenerator"
28   SET OID = OID + 1
29   WHERE ID = ' + ID ;
30   RETURN Result;
31 END
32

```

Using Parameters and Variables

Example showing the use of parameters and variables.

1. Parameter "aLinkTable" is passed into the procedure. Note how the Statement is **prepared** with a Question Mark. The Cursor is then opened with the SQL "OPEN [CursorName] USING [ParameterName];"
2. Variable "ID" is declared, and then set with FETCH FIRST FROM [CursorName] INTO [VariableName]; Once a value has been added to the ID it can be used in a later EXECUTE IMMEDIATE statement.

## Basic Language Elements of Procedural SQL

1. **BEGIN ...END;** Used to demark the start and end of a function or procedure.
2. **IF [Boolean Statement] THEN [DoSomething] ELSE [DoAnotherThing] END IF;** Used to create forking logic dependent on data and variables.
3. **WHILE [Boolean Statement] DO [DoSomething] END WHILE;** Repetition logic, to do something. Remember with a While statement you **must** provide an exit, or the loop will run forever, so the [Boolean Statement] must be acted on within the While statement to reset it.
4. **EXECUTE IMMEDIATE [SQL Statement];** Allows one or more SQL statements to be run in a procedure.
5. **DECLARE Result [DataType]; [DoSomething] RETURN Result;** This SQL syntax must be used within a Function to ensure that it returns the required value.
6. **FETCH FIRST/NEXT FROM [CursorName](['FieldNames']) INTO [VariableName];** Used to pull data out of a Cursor into a variable so it can be accessed by the Procedure.
7. **OPEN Crsr;** Used to Open a data-set based on a previously prepared Statement. If this is the last line of a Procedure, then the procedure will return a data-set which will be viewed in your App.