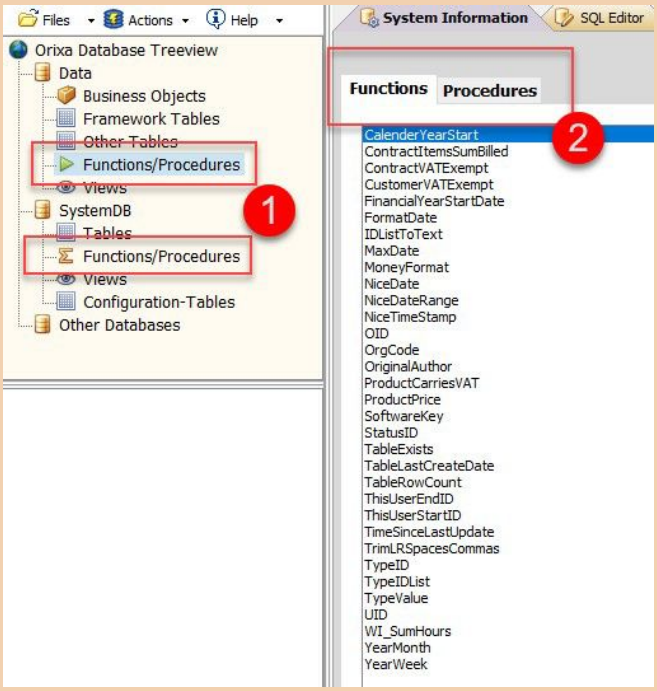
# CREATE FUNCTION, CREATE PROCEDURE

**Functions and Procedures can be written in your Orixa App, they can contain the full features of the SQL2003 syntax.**
**Functions run SQL script and return a value. They are most usually used to return data which is used in a GENERATED column in a table, or in data returned in a report or dashboard.**
**Procedures run code and do not return a value. They are most commonly used to perform data-management regular tasks.**

This guide cannot cover the full details of how to write Functions and Procedures, as this is a huge topic. The purpose of this page is to give basic guidence and some useful examples.

## Get up and running: Copy from existing procedures

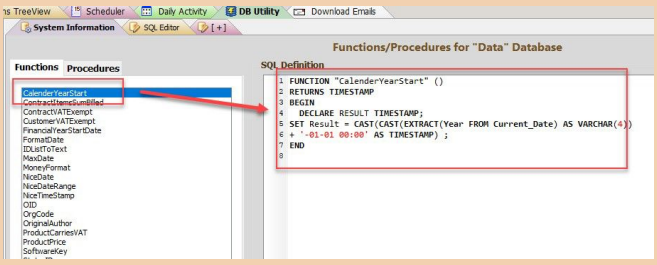
Your Orixa App will contain many functions and procedures. If you have Administrator priviledges you can access the DB Management Utility and view these. Take some time to study these, they show all the basic SQL syntax issues and possibilities of Functions and Procedures.




**Viewing example Functions and Procedures**

**In the DB Management Utility Access all existing Functions and Procedures**

1. Click on the "Functions/Procedures" item under either the "Data" database or the "SystemDB" item.
2. In the System Information tab, you will see lists of the Functions and Procedures on your system.




**Function example SQL**

Click on any Function or Procedure and it's SQL Definition will display.

You can copy and paste SQL from this window to use elsewhere in your Orixa system.

## Some basic Aspects of the Syntax of Functions and Procedures

1. They can accept **parameters** which can be **input, output or both input and output.** Parameters are values of data. Input parameters accept data from the user, output parameters are returned as the result of a procedure. All functions return a **result** which is a special kind of output parameter.
2. Within functions and procedures a wide range of new SQL keywords are used, which will be unfamiliar to Developers who have mainly written SELECT statements. The
3. The developer can create a **cursor** using SQL statement, if this statement is a SELECT statment, once open

3. the developer can iterate through it and operate on the data. If the statement is an UPDATE or INSERT, the developer can write code to pass in parameters.
4. The developer can **declare** variables at the start of the SQL which can hold values to use in the script.
5. SQL syntax ends lines with the semi-colon character (;).
6. Strings start and end with single quotes ('), and can cross multiple lines. If the string statement contains a single quote this must be doubled, as shown in examples below. If the developer wants to combine strings with variables or parameters the plus sign (+) can be used for concatenation, with a single quote before and afterwards, as shown in examples below.
7. Basic programming constructs like **if, while, case** all work in similar ways to the way they work in other programming languages.

## Some Syntax with Examples and Explanations

**DECLARING VARIABLES**

```
DECLARE Crsr CURSOR FOR Stmt;
DECLARE Stmt2 STATEMENT;
DECLARE iCounter INTEGER;
DECLARE sName VARCHAR(100);
DECLARE Result DECIMAL(19,4);
```

Variables can be declared with any data-type (identical to the data-types used in field definitions).

The specialized variables **CURSOR** and **STATEMENT**. Can be used for data manipulation.

When a cursor is declared it is always associated with a statement which will be used to query the database.

An example Cursor declaration is shown on the left.

**OPENING STATEMENTS**

```
PREPARE Stmt FROM
' SELECT
    SUM(BilledValue) as SumBilled
    FROM ContractItems
  WHERE ContractPaymentsID = ? ';
OPEN Crsr USING aID;
[some other lines of code ...]
CLOSE Crsr;
```

The example  to the left shows a simple statement being prepared, and opened.

Note the use of the "?" character to represent a holder position for a parameter in the statement.

In the example the "aID" keyword may refer to a variable declared earlier in the procedure, or a parameter passed in by the caller of the procedure.

Once the Cursor has been opened it becomes available in the procedure and values from the returned records can be used in code.

**USING AND USING CURSORS**

```
OPEN Crsr USING aID;
FETCH FIRST FROM Crsr('AuthorID') INTO
Result;
RETURN Result;
```

```
OPEN Crsr USING aID;
IF ROWCOUNT(Crsr) = 0 THEN
  CLOSE Crsr;
  ELSE IF
  FETCH FIRST FROM Crsr('ID') INTO aID;
  [some other lines of code ...]
  WHILE NOT EOF(Crsr) DO
    [some other lines of code ...]
    FETCH NEXT FROM Crsr('ID') INTO aID;
    END WHILE;
  END IF;
```

The key processes that need to be understood to use Cursors are the keywords
FETCH FIRST FROM CursorName('FieldName') INTO VariableName.

Note that the FieldName is enclosed in single quotes, and more than one FieldName and VariableName can be contained in comma-separated lists.

Once values have been extracted from the cursor, it is common to use the WHILE statement to iterate the remaining rows.

WHILE NOT EOF(CursorName) DO

EOF is short for "End Of File", meaning that the subsequent code will continue until all records in the cursor have been dealt with.

**USING EXECUTE IMMEDIATE**

```
EXECUTE IMMEDIATE
 ' DELETE FROM Organisations
   WHERE ID = ' + CAST(aID as VARCHAR);
```

The EXECUTE IMMEDIATE keywords do exactly what you would expect. They allow a SQL Statement to be made and immediately run. Note how the "aID" variable, which is declared as an INTEGER must be CAST to a VARCHAR in the statement on the left in order for the statement to parse correctly.

**INPUT AND OUTPUT PARAMETERS**

```
CREATE PROCEDURE "DoSomething" (IN "aID"
INTEGER, IN "aFieldname" VARCHAR, IN
"aMasterTable" VARCHAR)
```

The statement on the left shows a procedure which declares three input parameters. Note the syntax:

IN "ParameterName" DATA-TYPE.

[Parameter Type] [ParameterName] [DataType]

The Parameter Type can be "IN", "OUT" or "INOUT"