SQL Data Modeling Guide — Chapter 2

SQL on MarkLogic Server Quick Start

This chapter describes how to set up your MarkLogic Server for SQL. This chapter describes how to set up a typical development environment in which the SQL client and MarkLogic Server are configured on the same machine. For a production environment, you would typically configure your SQL client and MarkLogic Server on separate machines.

You must have the admin role on MarkLogic Server to complete the procedures described in this chapter.

The main topics in this chapter are:

- Setup MarkLogic Server
- Create Views
- Load the Data
- Enter SQL Queries to Test
- Using MLSQL

Setup MarkLogic Server

Install MarkLogic Server on the database server, as described in the *Installation Guide*. and follow these procedures:

- Create a Schema Database and a SQL Database
- Create Range Indexes for the Database
- Create a Field for the Database
- Create an ODBC App Server

Create a Schema Database and a SQL Database

How to create a database is described in detail in Creating a New Database in the *Administrator's Guide*. This section provides a quick-start procedure for creating the database used in this example.

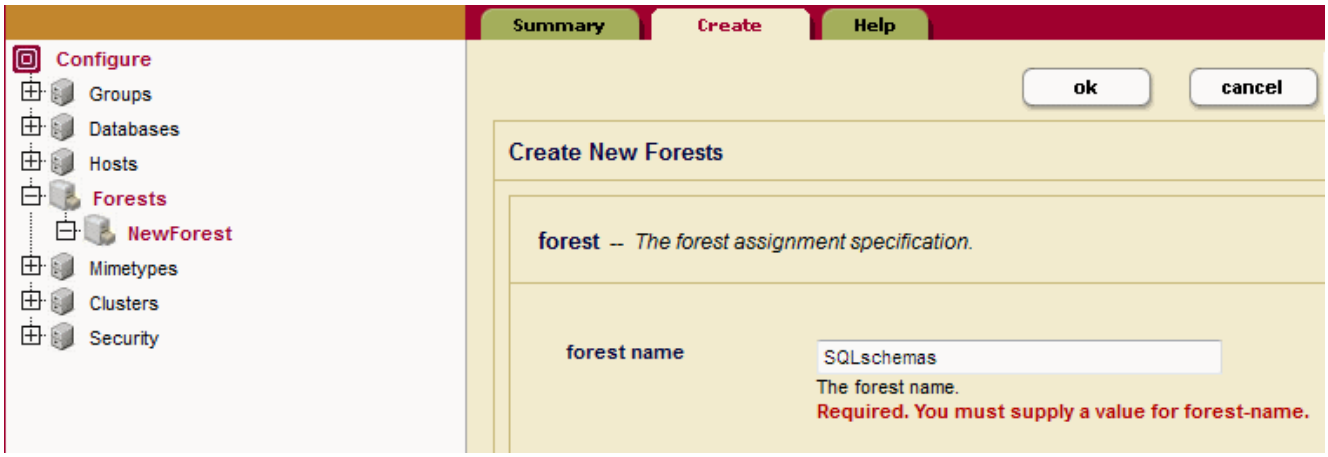
Every SQL database must have its own separate schema database.

1. Open your browser and navigate to the Admin Interface:

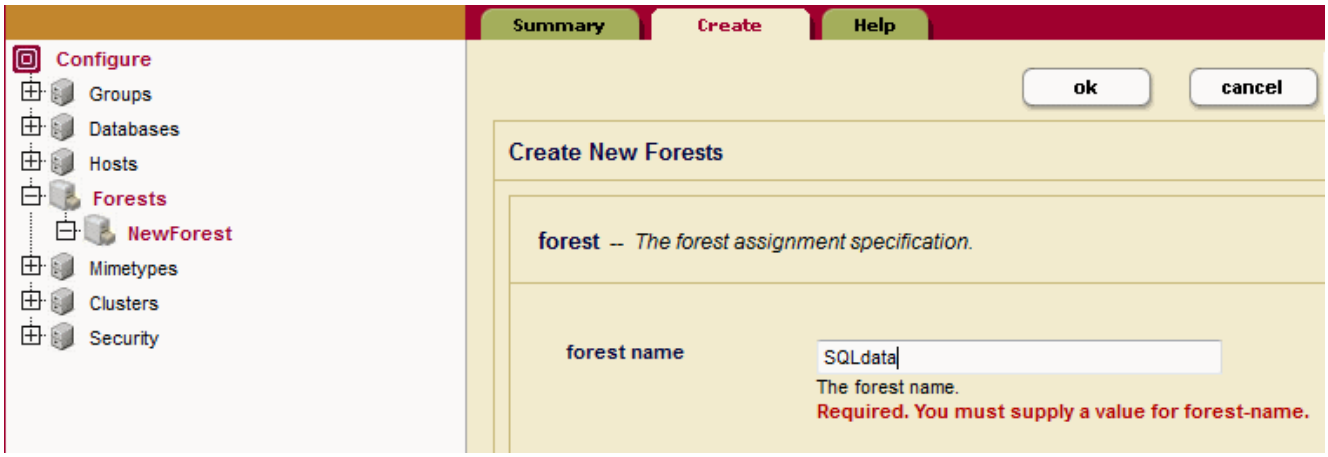
   `http://hostname:8001`

   Where *hostname* is the name of your MarkLogic Server host machine.

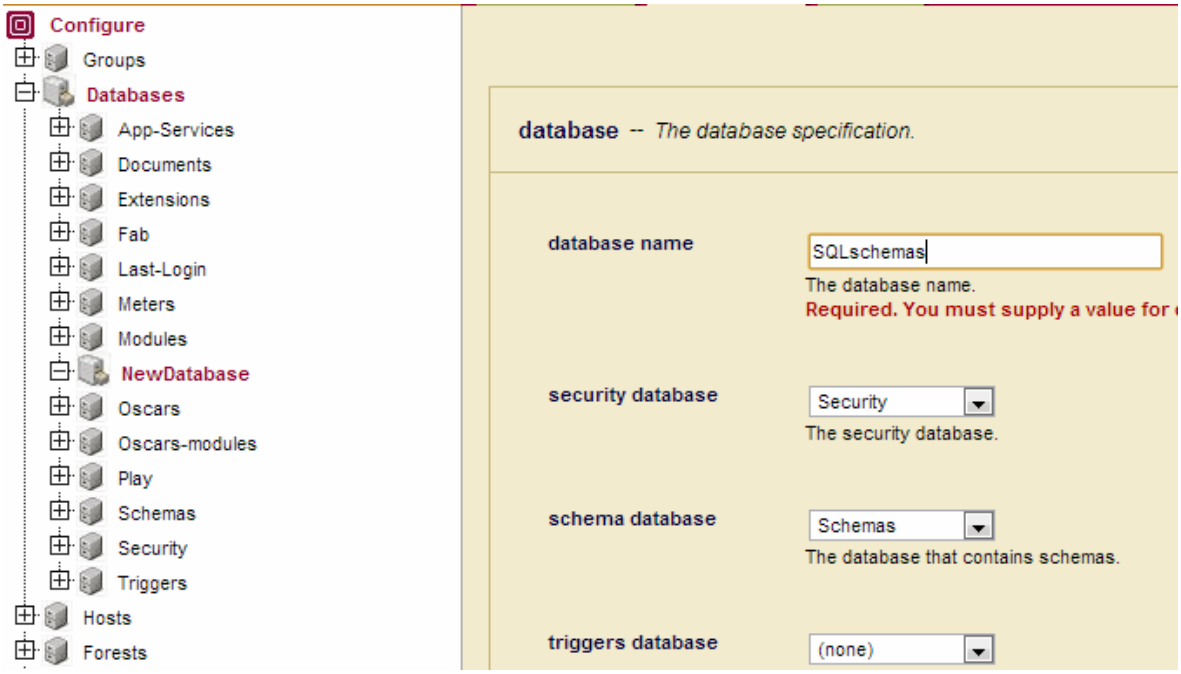
2. Click the Forests icon in the left frame.
3. Click the Create tab at the top right. The Create Forest page displays. Enter ‘SQLschemas' as the name of your forest in the Forest Name textbox. Click OK.



1. Click the Create tab at the top right. The Create Forest page displays. Enter ‘SQLdata' as the name of your forest in the Forest Name textbox. Click OK.

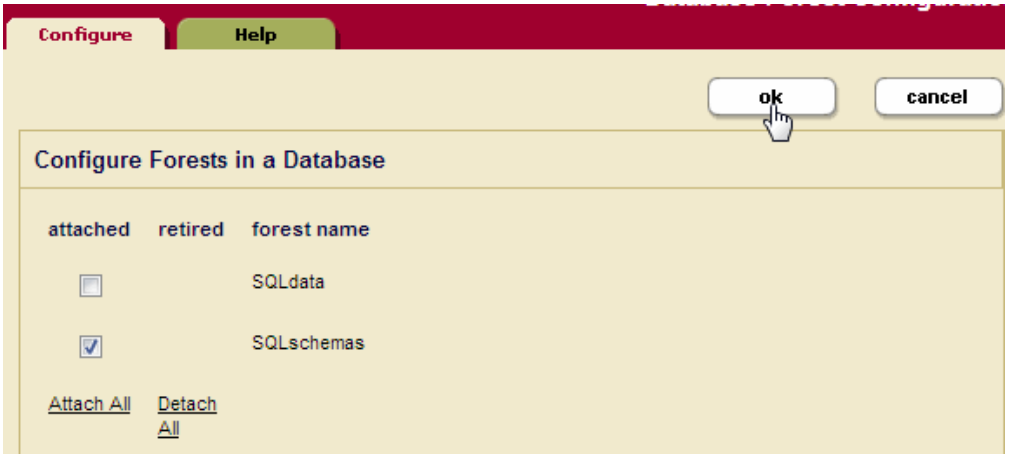


1. Click the Databases icon in the left tree menu.
2. Click the Create tab at the top right. The Create Database page displays. Enter ‘SQLschemas' as the name of the new database and click Ok:



3. At the top of the page click Database->Forests

4. Check the SQLschemas box to attach the SQLschemas forest. Click Ok:



5. Click the Create tab at the top right. The Create Database page displays. Enter ‘SQLdata' as the name of the new database and select ‘SQLschemas' as the Schema Database. Click Ok:

6. At the top of the page click Database->Forests

7. Check the SQLdata box to attach the SQLdata forest. Click Ok:

Create Range Indexes for the Database

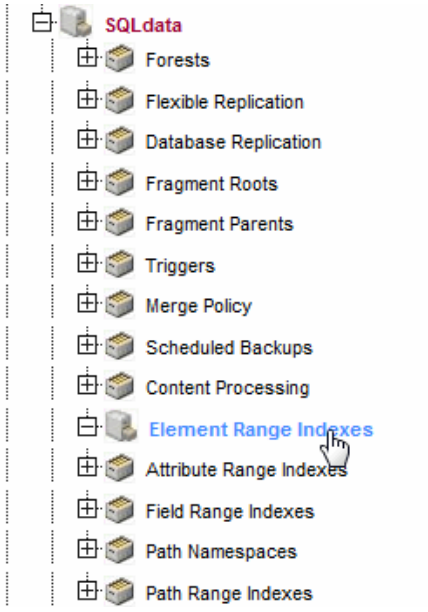
The sample JSON documents modeled in this chapter take the following form:

```
{ "Employee": {
      "EmployeeID": 1,
      "FirstName": "John",
      "LastName": "Widget",
      "Position": "Manager of Human Resources" }}
{ "Expenses": {
      "EmployeeID": 2,
      "Date": "2012-06-27",
      "Amount": 131.02}}
```

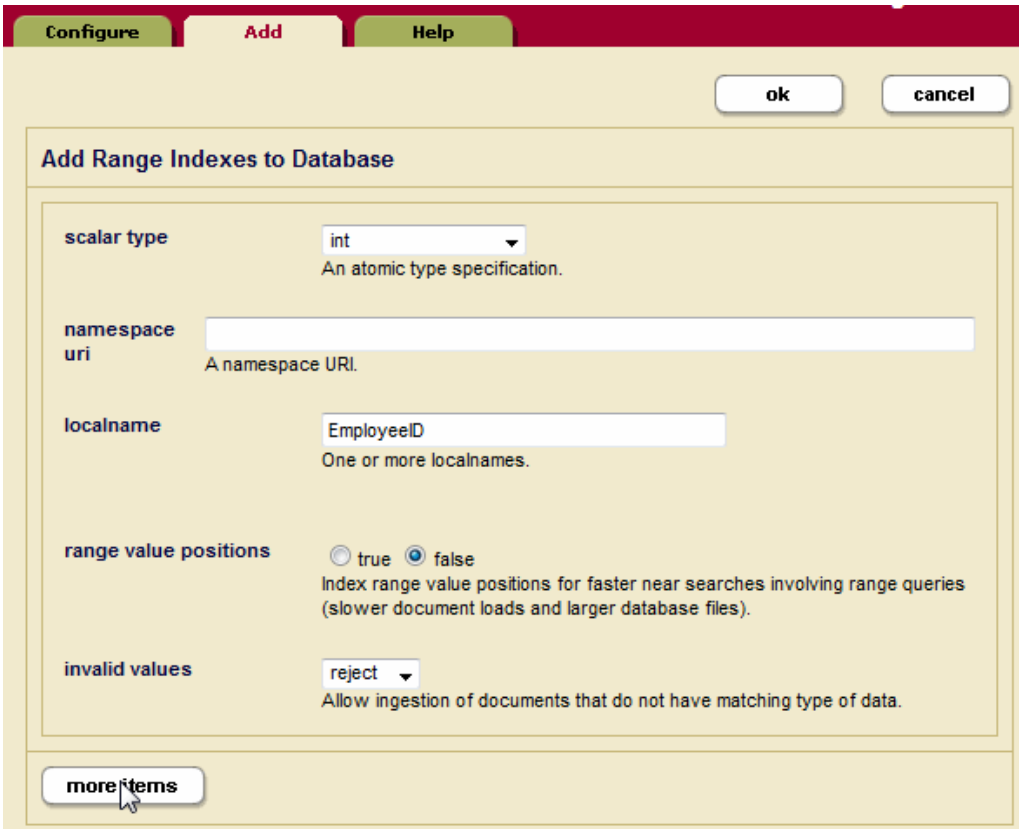
In order to model such documents as SQL tables, we need to create range indexes for each 'column' (`EmployeeID`, `FirstName`, `LastName`, `Date`, and `Amount`).

In the following section, Create a Field for the Database, you will create a field for the `Position` element.

1. Click the Element Range Indexes icon in the tree menu, under the 'SQLdata' database.



2. Click the Add tab. The Element Range Index Configuration page or the Element Word Lexicon Configuration page displays. Set the Scalar Type to `int` and enter `EmployeeID` in the localname field to create a range index for the `EmployeeID` element. Click More Items.



3. Create element range indexes for the remaining elements, as shown in the following table. Leave all other range index fields empty or unchanged with their default settings.

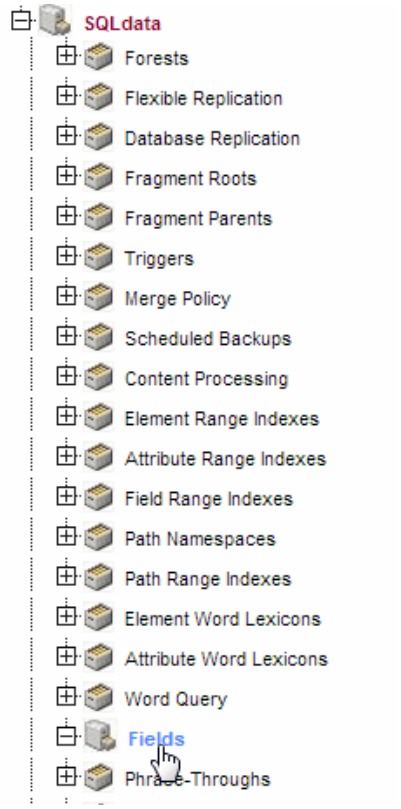
Local Name	Scalar Type
`FirstName`	`string`
`LastName`	`string`
`Date`	`date`
`Amount`	`decimal`

4. Click Ok when finished.

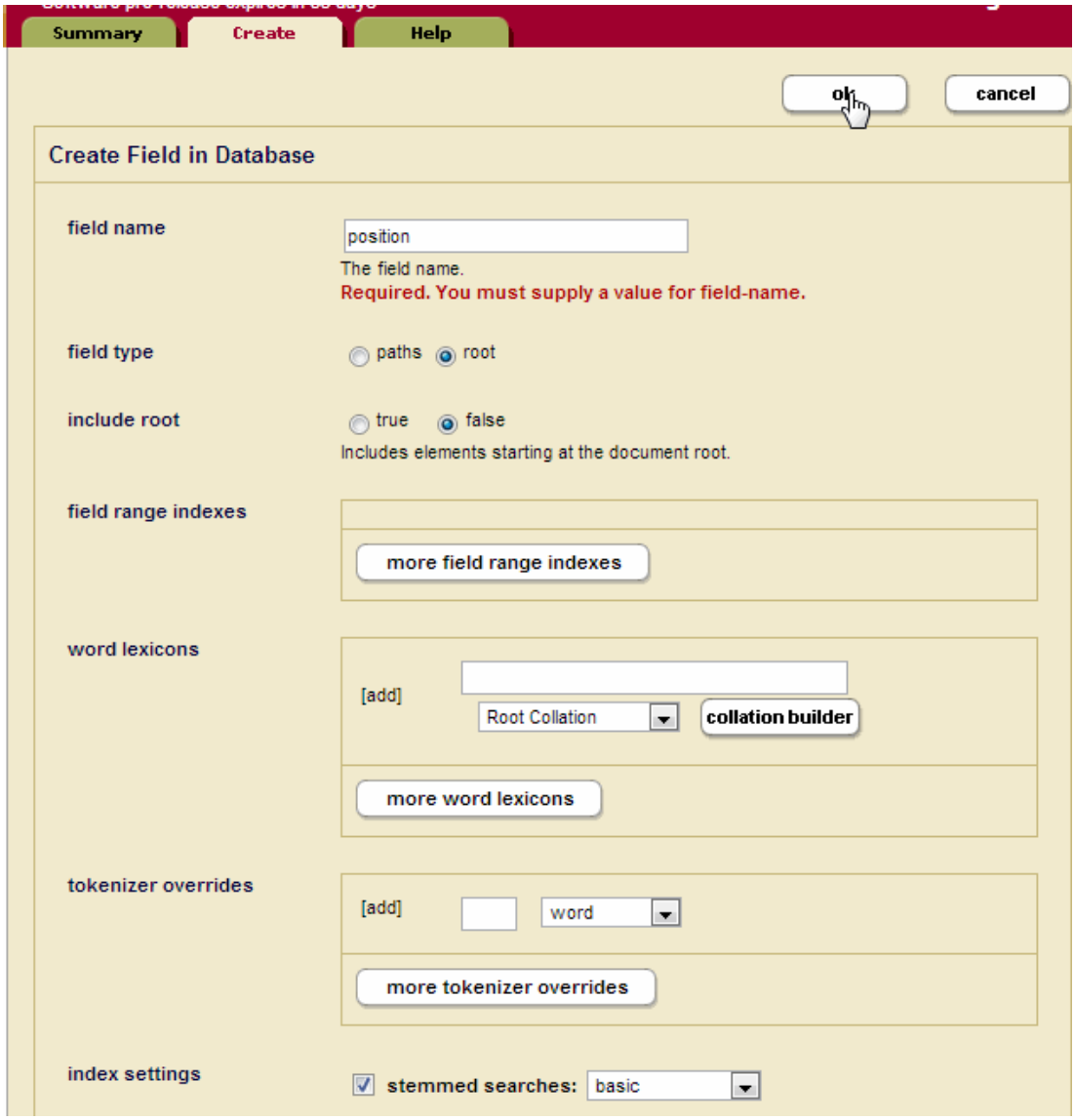
Create a Field for the Database

In MarkLogic Server, fields allow you to narrow searches to specific elements. Like the range indexes created in Create Range Indexes for the Database, the field created in this section will be bound to a view. Fields allow you to do more precise searches with the MATCH operator, as demonstrated in Using MLSQL and MATCH Operator.

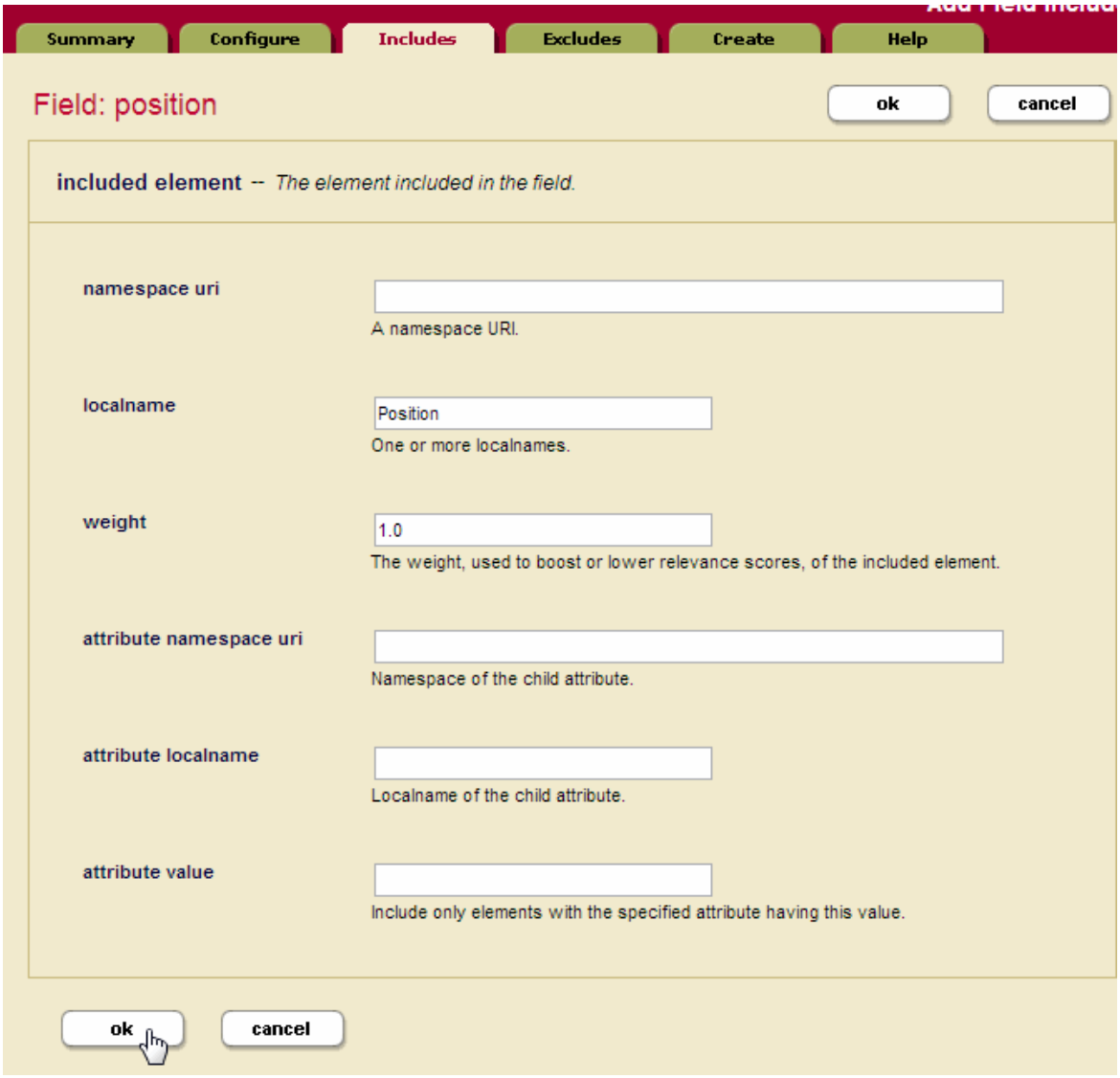
1. Click the Fields icon in the tree menu, under the ‘SQLdata' database.



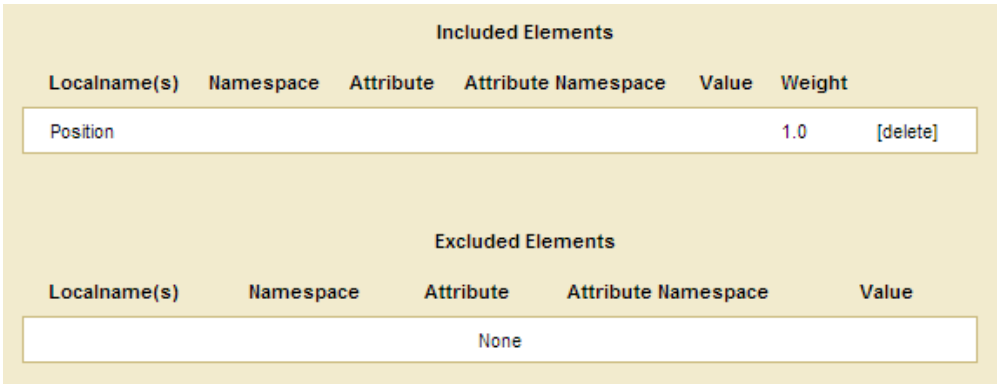
2. Click the Create tab. The Database Fields Configuration page displays. Select a field type of `root` and enter `position` for the field name. Leave all other settings as the defaults. Click Ok.



3. The tabs will expand to display additional tabs. Click on the Includes tab and enter `Position` in the localname field. Click Ok.



4. The bottom section of the Database Fields Configuration page for the `position` field will now display that it includes the `Position` field:



Create an ODBC App Server

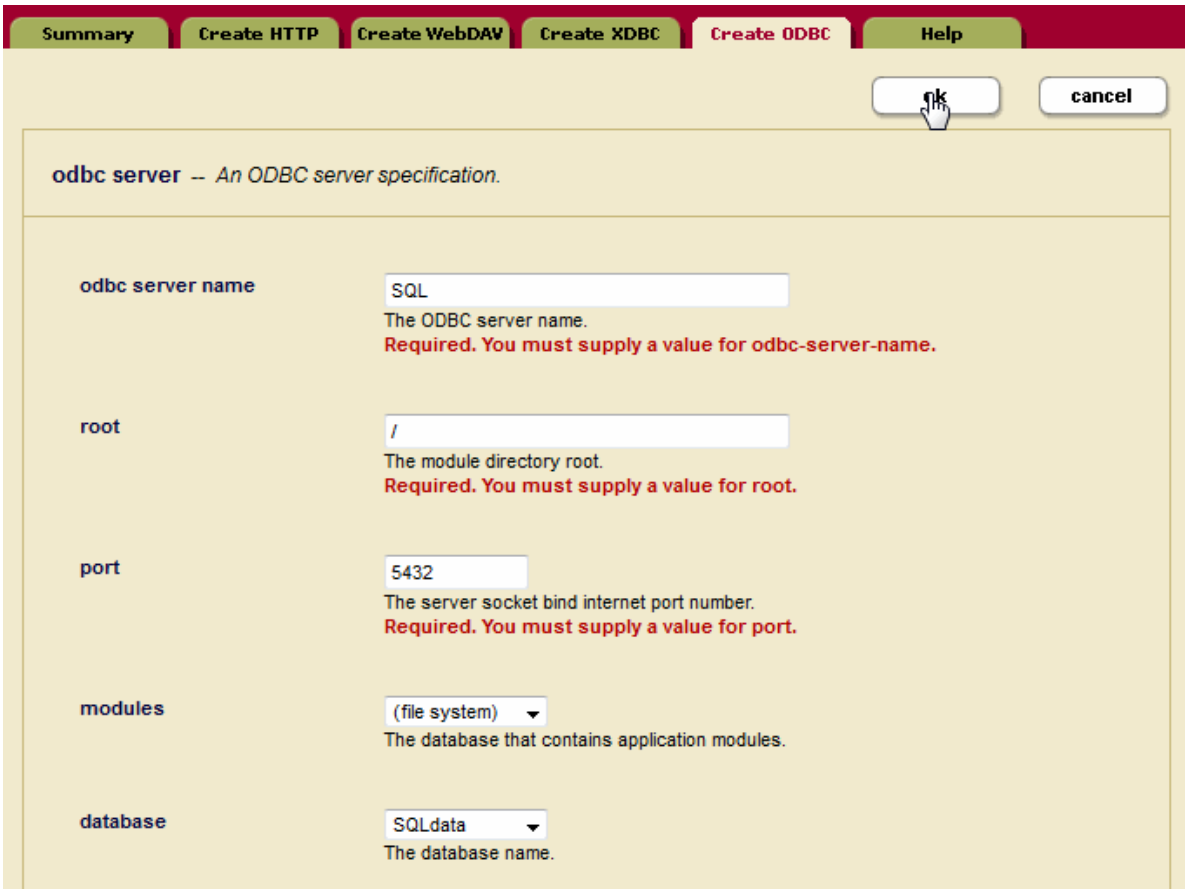
Schemas and views represent content stored in a MarkLogic Server database. Each content database used by a SQL client is managed by an ODBC App Server that accepts SQL queries from the SQL client and responds by returning MarkLogic Server data in tuple form. An ODBC App Server can manage only one content database. However, a single content database can be managed by multiple ODBC App Servers.

ODBC App Servers are described in detail in the ODBC Servers chapter in the *Administrator's Guide*.

Open the Admin Interface

To create a new server, complete the following steps:

1. Click the Groups icon in the left frame.
2. Click the group in which you want to define the ODBC server (for example, Default).
3. Click the App Servers icon on the left tree menu.
4. Click the Create ODBC tab at the top right. The Create ODBC Server page will display:



5. In the Server Name field, enter a shorthand name for this ODBC server. In this example, the name of the App Server is 'SQL.'
6. In the Root directory field, enter /.
7. In the Port field, enter the port number through which you want to make this ODBC server available. The default PostgreSQL listening socket port is 5432.
8. Leave the Modules field as `(file system)`.
9. In the Database field, select the 'SQLdata' database you created in Create a Schema Database and a SQL Database.

Create Views

This section describes how to use the REST management API to create the views used by SQL queries.

1. Use POST /manage/v2/databases/{idlname}/view-schemas to create a schema, named 'main'.

```
curl -X POST  --anyauth --user admin:admin \
--header "Content-Type:application/json" -d '{"view-schema-name": "main"}' \
http://gordon-2:8002/manage/v2/databases/SQLdata/view-schemas?format=json
```

1. Use POST /manage/v2/databases/{idlname}/view-schemas/{schema-name}/views to create a view in the 'main' schema, named 'employees'. Specify a scope on the 'Employee' element and columns for the 'firstname', 'lastname', and 'employeeid' range indexes. Specify the 'position' field as a searchable field.

```
curl -X POST  --anyauth --user admin:admin \
--header "Content-Type:application/json" \
-d '{
     "view-name": "employees",
     "element-scope":{"namespace-uri":"", "localname":"Employee"},
     "column": [
       {
         "column-name": "employeeid",
         "element-reference": {
            "namespace-uri": "",
```

```
            "localname": "EmployeeID",
            "scalar-type": "int"
          }
        },
        {
          "column-name": "firstname",
          "element-reference": {
            "namespace-uri": "",
            "localname": "FirstName",
            "scalar-type": "string",
            "collation": "http://marklogic.com/collation/"
          }
        },
        {
          "column-name": "lastname",
          "element-reference": {
            "namespace-uri": "",
            "localname": "LastName",
            "scalar-type": "string",
            "collation": "http://marklogic.com/collation/"
          }
        }
      ],
      "field": [{"field-name": "position",
                 "localname": "Position",
                 "namespace-uri": ""}]
    }' \
http://gordon-2:8002/manage/v2/databases/SQLdata/view-schemas/main/views?format=json
```

1. Use POST /manage/v2/databases/{id|name}/view-schemas/{schema-name}/views to create a second view in the ‘main' schema, named ‘expenses', with a scope on the ‘Expenses' element and columns for the ‘EmployeeID', ‘Date', and ‘Amount' range indexes.

```
curl -X POST  --anyauth --user admin:admin \
--header "Content-Type:application/json" \
-d '{
      "view-name": "expenses",
      "element-scope":{"namespace-uri":"", "localname":"Expenses"},
      "column": [
        {
          "column-name": "employeeid",
          "element-reference": {
            "namespace-uri": "",
            "localname": "EmployeeID",
            "scalar-type": "int"
          }
        },
        {
          "column-name": "date",
          "element-reference": {
            "namespace-uri": "",
            "localname": "Date",
            "scalar-type": "date"
          }
        },
        {
          "column-name": "amount",
          "element-reference": {
            "namespace-uri": "",
            "localname": "Amount",
            "scalar-type": "decimal"
          }
        }
      ]
    }' \
http://gordon-2:8002/manage/v2/databases/SQLdata/view-schemas/main/views?format=json
```

1. List the views in the ‘main' schema.

```
curl -v -X GET --anyauth -u admin:admin \
--header "Content-Type:application/json" \
http://gordon-2:8002/manage/v2/databases/SQLdata/view-schemas/main/views?format=json
```

If you change a range index used by a column, you need to delete and recreate the view in order for the view to use the new index.

Load the Data

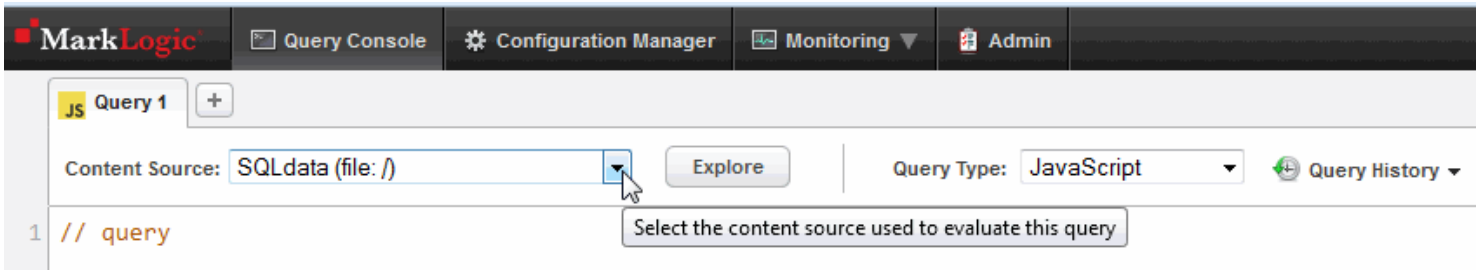
This section describes the procedure for loading the sample documents.

1. Go to the following URL to open Query Console:

   `http://hostname:8000/qconsole/`

Where *hostname* is the name of your MarkLogic Server host.

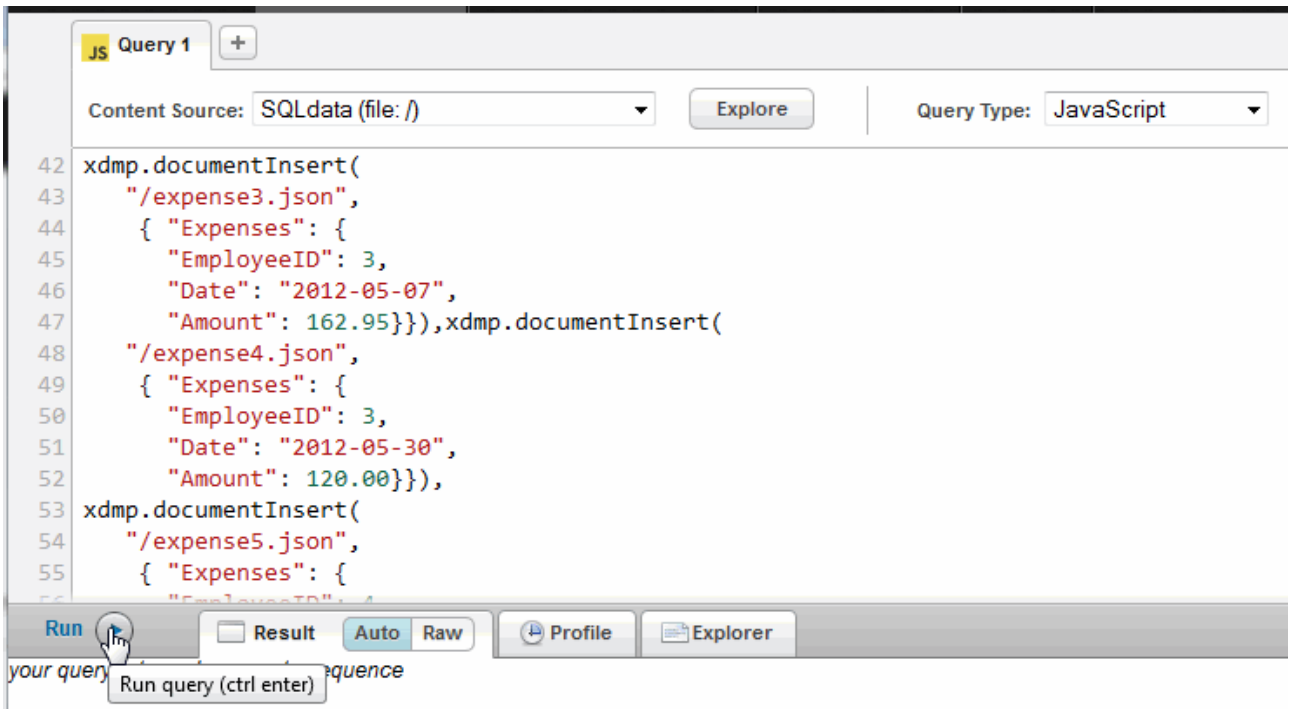
2. Select the SQLdata database from the Content Source pulldown menu and JavaScript from the Query Type menu.



1. Cut and paste the following JavaScript into Query Console:

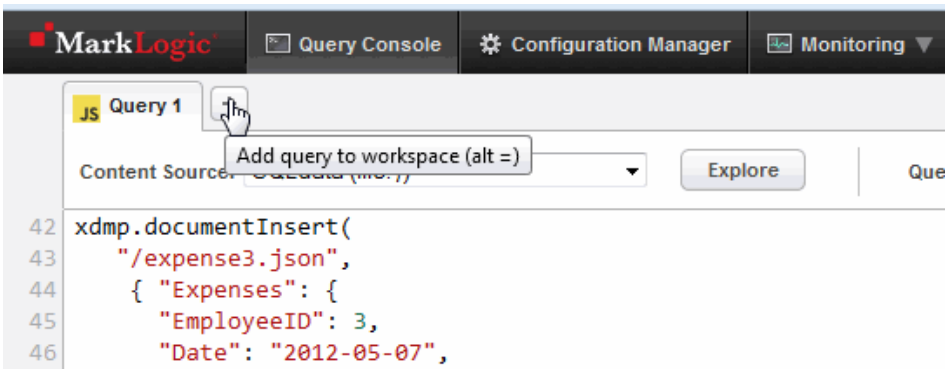
```
declareUpdate();
xdmp.documentInsert(
   "/employee1.json",
    { "Employee": {
      "EmployeeID": 1,
      "FirstName": "John",
      "LastName": "Widget",
      "Position": "Manager of Human Resources" }}),
xdmp.documentInsert(
   "/employee2.json",
    { "Employee": {
      "EmployeeID": 2,
      "FirstName": "Jane",
      "LastName": "Lead",
      "Position": "Manager of Widget Research" }}),
xdmp.documentInsert(
   "/employee3.json",
    { "Employee": {
      "EmployeeID": 3,
      "FirstName": "Steve",
      "LastName": "Manager",
      "Position": "Senior Technical Lead" }}),
xdmp.documentInsert(
   "/employee4.json",
    { "Employee": {
      "EmployeeID": 4,
      "FirstName": "Debbie",
      "LastName": "Goodall",
      "Position": "Senior Widget Researcher" }}),
xdmp.documentInsert(
   "/expense1.json",
    { "Expenses": {
      "EmployeeID": 2,
      "Date": "2012-06-27",
      "Amount": 131.02}}),
xdmp.documentInsert(
   "/expense2.json",
    { "Expenses": {
      "EmployeeID": 1,
      "Date": "2012-08-03",
      "Amount": 59.95}}),
xdmp.documentInsert(
   "/expense3.json",
    { "Expenses": {
      "EmployeeID": 3,
      "Date": "2012-05-07",
      "Amount": 162.95}}),
xdmp.documentInsert(
   "/expense4.json",
    { "Expenses": {
      "EmployeeID": 3,
      "Date": "2012-05-30",
      "Amount": 120.00}}),
xdmp.documentInsert(
   "/expense5.json",
    { "Expenses": {
      "EmployeeID": 4,
      "Date": "2012-03-23",
      "Amount": 155.55}}),
xdmp.documentInsert(
   "/expense6.json",
    { "Expenses": {
      "EmployeeID": 4,
      "Date": "2012-06-05",
      "Amount": 104.29}})
```

2. In the control bar below the query window, click Run:

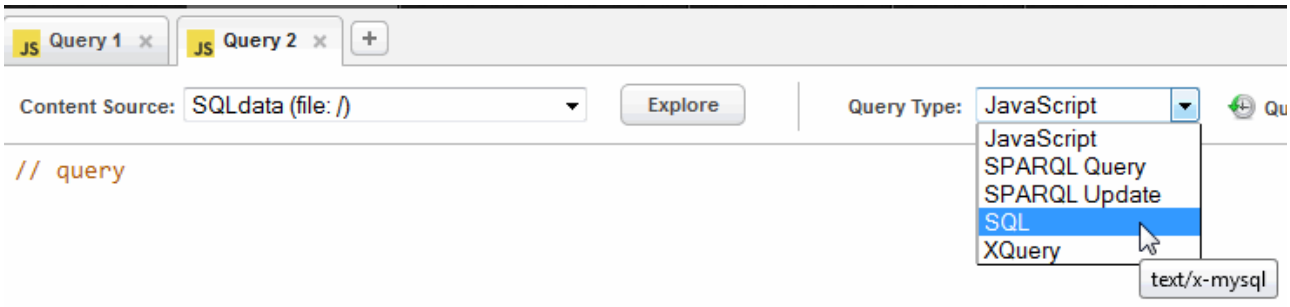


Enter SQL Queries to Test

1. To test that everything is working correctly, click + to open another query window:



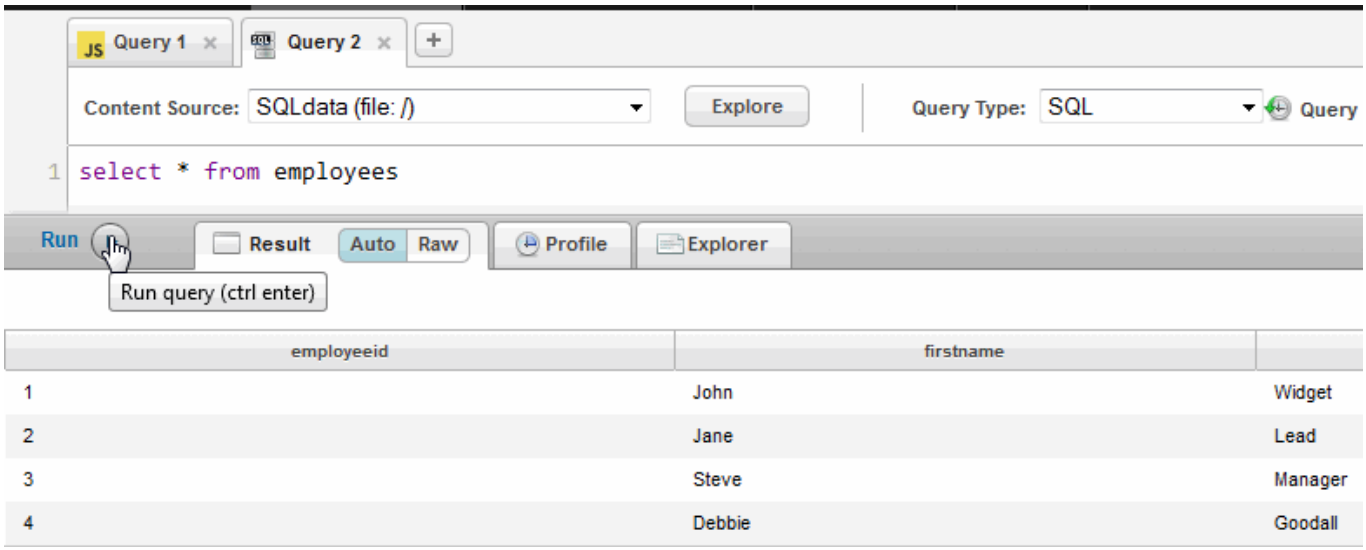
2. In the new query window, make sure you have 'SQLdata' selected in the Content Source pull-down menu. Select a Query Type of SQL:



Enter the following query:

```
select * from employees
```

1. In the control bar below the query window, select Run.
2. You should see results that look like the following:



MarkLogic Server treats SQL as case insensitive. Uppercase and lowercase characters are treated the same.

Using MLSQL

The MLSQL tool is a command line interface for issuing SQL statements. The executable MLSQL file is located in a MarkLogic installation at the following location:

- Windows: `c:\Program Files\MarkLogic\mlsql.exe`
- Linux/Unix: `/opt/MarkLogic/bin/mlsql`

You must be assigned the `sql-execution` role on MarkLogic Server to use MLSQL.

To use the MLSQL tool, open a shell window and enter:

```
mlsql -h hostname -p 5432 -U username
```

Enter your password when you see a prompt like:

```
username=>
```

Enter a few SQL queries, like the following:

```
username=> SELECT * FROM employees;
username=> SELECT employees.FirstName, employees.LastName,
SUM(expenses.Amount) AS ExpensesPerEmployee
FROM employees, expenses
WHERE employees.EmployeeID = expenses.EmployeeID
GROUP BY employees.FirstName, employees.LastName;
username=> SELECT employees.FirstName, employees.LastName,
SUM(expenses.Amount) AS ExpensesPerEmployee
FROM employees JOIN expenses
ON employees.EmployeeID = expenses.EmployeeID
GROUP BY employees.FirstName, employees.LastName
ORDER BY ExpensesPerEmployee;
```

A semicolon (;) is used in MLSQL to designate the end of a SQL query.

To demonstrate the purpose of the Searchable Field in the view, try the following queries:

```
username=> SELECT * from employees WHERE employees MATCH "Manager";
username=> SELECT * from employees WHERE employees
MATCH "position:Manager";
```

The first query searches for the word 'Manager' in all of the document elements. The `position:Manager` specification in the second query narrows the search for 'Manager' to the elements included in the `position` field, which in this case is the `Position` element.

To exit MLSQL, enter: `\q`

If you get results from the SQL queries, you can proceed to connecting your BI tool to MarkLogic Server, as described in Connecting Cognos to MarkLogic Server and Connecting Tableau to MarkLogic Server.

If you add or change the contents of a view, database, or documents, you must exit and restart MLSQL.

The MLSQL session commands are:

Command	Description
\copyright	Returns distribution terms.
\h	Returns list of available SQL commands.
\?	Returns list of available psql commands.
\g	Re-executes the last query.
\q	Exits MLSQL.

The syntax of a MLSQL command is:

```
mlsql [OPTION]... [DBNAME [USERNAME]]
```

Where:

Connection options:

Option	Description
`-h, --host=HOSTNAME`	Database server host or socket directory (default: "local socket")
`-p, --port=PORT`	ODBC server port (default: "5432")
`-U, --username=USERNAME`	Database user name (default: "username")
`-w, --no-password`	Never prompt for password
`-W, --password`	Force password prompt (should happen automatically)

General options:

Option	Description
`-c, --command=COMMAND`	Run only single command (SQL or internal) and exit
`-d, --dbname=DBNAME`	Database name to connect to (default: "gfurbush")
`-f, --file=FILENAME`	Execute commands from file, then exit
`-l, --list`	List available databases, then exit
`-v, --set=, --variable=NAME=VALUE`	Set psql variable NAME to VALUE
`-X, --no-psqlrc`	Do not read startup file (~/.psqlrc)
`-1` ("one"), `--single-transaction`	Execute command file as a single transaction
`--help`	Show help, then exit
`--version`	Output version information, then exit

Input and output options:

Option	Description
`-a, --echo-all`	Echo all input from script
`-e, --echo-queries`	Echo commands sent to server
`-E, --echo-hidden`	Display queries that internal commands generate

`-L, --log-file=FILENAME`	Send session log to file
`-n, --no-readline`	Disable enhanced command line editing (readline)
`-o, --output=FILENAME`	Send query results to file (or |pipe)
`-q, --quiet`	Run quietly (no messages, only query output)
`-s, --single-step`	Single-step mode (confirm each query)
`-S, --single-line`	Single-line mode (end of line terminates SQL command)

Output format options:

Option	Description
`-A, --no-align`	Unaligned table output mode
`-F, --field-separator=STRING`	Set field separator (default: "|")
`-H, --html`	HTML table output mode
`-P, --pset=VAR[=ARG]`	Set printing option VAR to ARG (see \pset command)
`-R, --record-separator=STRING`	Set record separator (default: newline)
`-t, --tuples-only`	Print rows only
`-T, --table-attr=TEXT`	Set HTML table tag attributes (e.g., width, border)
`-x, --expanded`	Turn on expanded table output