MATLAB

Database Toolbox

What Is the Database Toolbox?

- The Database Toolbox is one of an extensive collection of toolboxes for use with MATLAB . The Database Toolbox enables you to move data (both importing and exporting) between MATLAB and popular relational databases .
- With the Database Toolbox, you can bring data from an existing database into MATLAB, use any of MATLAB's computational and analytic tools, and store the results back in the database or in another database. You read from the database, importing the data into the MATLAB workspace.
- For example, a financial analyst working on a mutual fund could import a company's financial data into MATLAB, run selected analyses, and store the results for future tracking. The analyst could then export the saved results to a database.

How Databases Connect to MATLAB?

- The Database Toolbox connects MATLAB to a database using MATLAB functions. Data is retrieved from the database as a string, parsed into the correct data types, and stored in a MATLAB cell array. At that point, you use MATLAB's extensive set of tools to work with the data. You can include Database Toolbox functions in MATLAB M-files. To export the data from MATLAB to a database, you use MATLAB functions.
- The Database Toolbox also comes with the Visual Query Builder (VQB), an easy-to-use graphical user interface for retrieving data from your database. With the VQB, you build queries to retrieve data by selecting information from lists rather than by entering MATLAB functions. The VQB retrieves the data into a MATLAB cell array so you then can process the data using MATLAB's suite of functions. With the VQB, you can display the retrieved information in relational tables, reports, and charts.

Features of the Database Toolbox

- Data types are automatically preserved in MATLAB – No data massaging or manipulation is required. The data is stored in cell arrays, which support mixed data types.
- Different databases can be used in a single session – Import data from one database, perform calculations, and export the modified or unmodified data to another database. Multiple databases can be open during a session.
- Database connections remain open until explicitly closed – Once connection to a database has been established, it remains open during the entire MATLAB session until you explicitly close it. This improves access and reduces the number of functions necessary to import/export data.
- Retrieval of large data sets or partial data sets – You can retrieve large data sets from a database in a single fetch or in discrete amounts using multiple fetches.

Features of the Database Toolbox

- Retrieval of database metadata – You do not need to know the table names, field names, and properties of the database structure to access the database, but can retrieve that information using Database Toolbox functions.
- Visual Query Builder – If you are unfamiliar with SQL, you can retrieve information from databases via this easy-to-use graphical interface.

Installing the Database Toolbox

- **Setting Up a Data Source :**

Before you can connect from the Database Toolbox to a database, you need to set up a data source, such as driver, directory, server, or network names. You assign a name to each data source.

- **The instructions for setting up a data source differ slightly depending on your configuration. Use one of these sets of instructions:**

- For MATLAB PC platforms whose database resides on that PC, use “Setting Up a Local Data for ODBC Drivers” on page 1-6.
- For MATLAB PC platforms whose database resides on another system to which the PC is networked, use “Setting Up a Remote Data Source for ODBC Drivers” on page 1-8.
- For MATLAB platforms that connect to a database via a JDBC driver, use “Setting Up a Data for JDBC Drivers” on page 1-12.

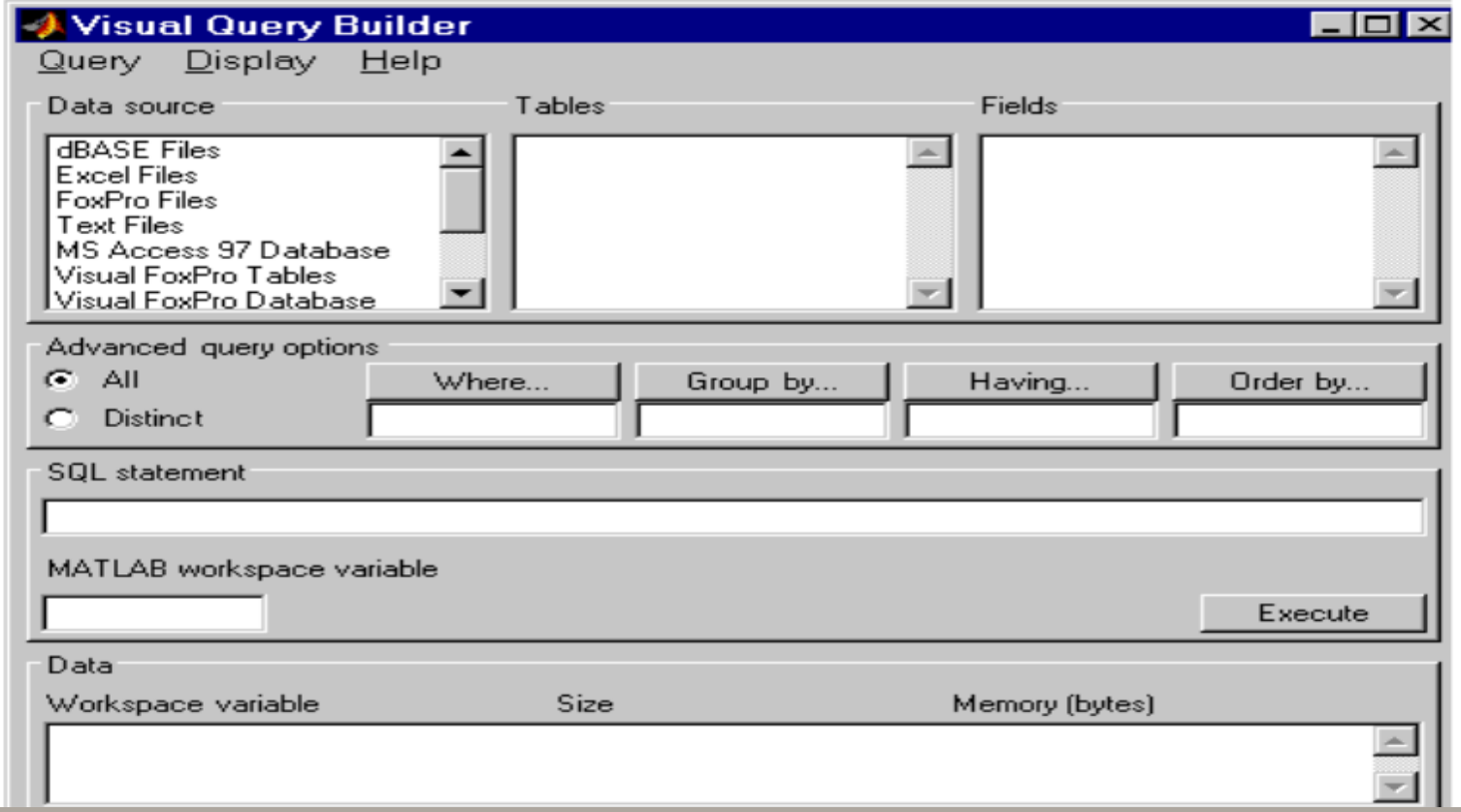
Starting the Database Toolbox

- To use the Database Toolbox functions, just type the function you want to use. For more information, see “Tutorial for Functions” on page 3-1.

- To start the Visual Query Builder, type query builder . For more information, see “Visual Query Builder Tutorial” on page 2-1.

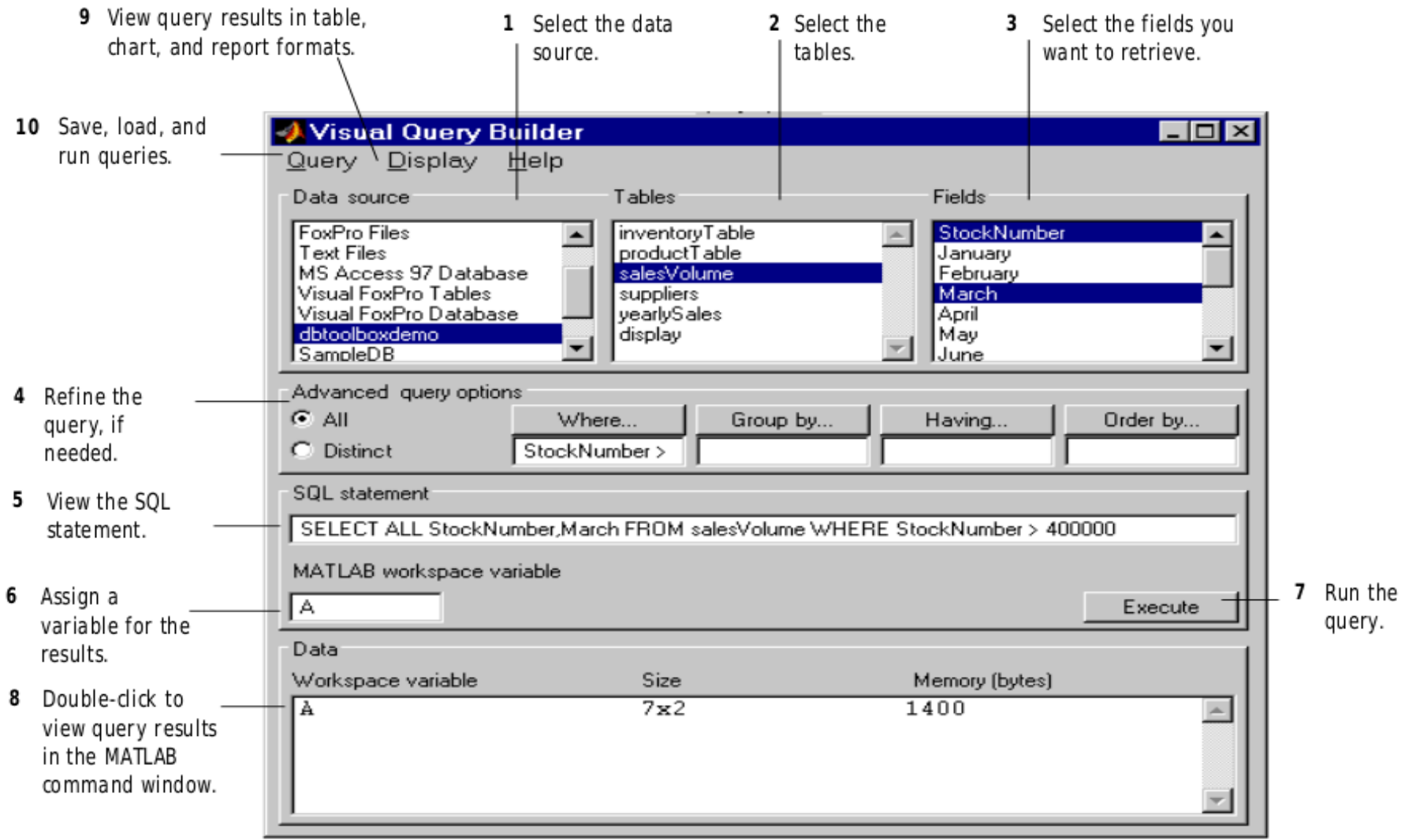
Starting the Visual Query Builder

The **Visual Query Builder** dialog box appears.



To Quit : from Query ,Select **EXIT**

Visual Query Builder Interface Tutorial



Visual Query Builder Interface Tutorial

262.62%

Command Window

File Edit View Web Window Help

```
>>
A =

    [125970]    [1400]    [1100]    [ 981]
    [212569]    [2400]    [1721]    [1414]
    [389123]    [1800]    [1200]    [ 890]
    [400314]    [3000]    [2400]    [1800]
    [400339]    [4300]    [ NaN]    [2600]
    [400345]    [5000]    [3500]    [2800]
    [400455]    [1200]    [ 900]    [ 800]
    [400876]    [3000]    [2400]    [1500]
    [400999]    [3000]    [1500]    [1000]
    [888652]    [ NaN]    [ 900]    [ 821]
```

Ready

Viewing Query Results

- After running a query in the Visual Query Builder, you can view :
- The retrieved data in the MATLAB command window, as described “Building, Running, and Saving a Query” .
- A “Relational Display of Data” on page 2-13.
- A “Chart Display of Results” on page 2-16; for example, a pie chart.
- A “Report Display of Results in a Table” on page 2-19.
- A “Display of Results in the Report Generator” on page 2-20

Viewing Query Results

- **1- Relational Display of Data :**

1 After executing a query, select Data from the Display menu. The query results appear in a figure window.

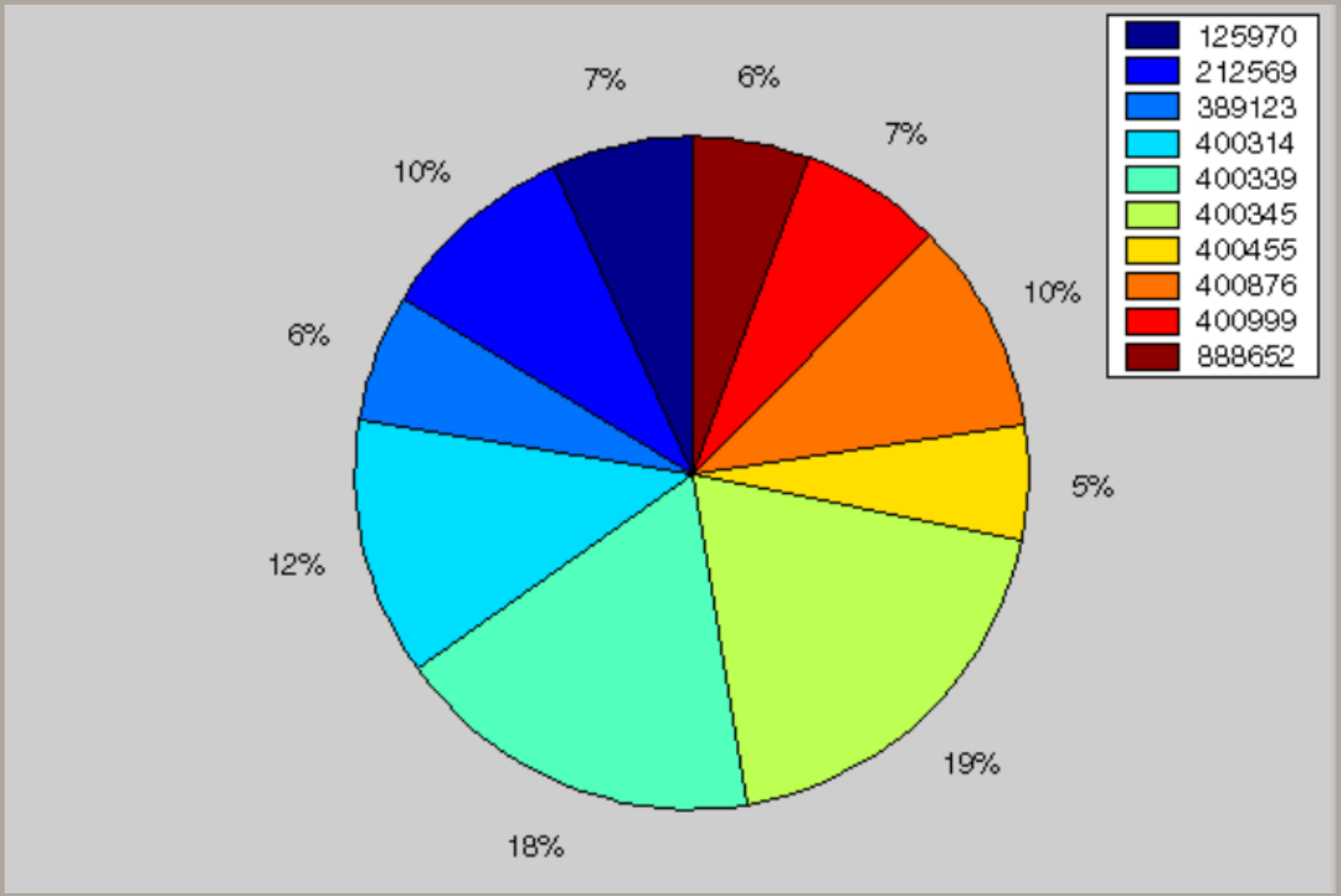
StockNumber	January	February	March
125970	0	0	800
212569	1200	900	821
389123	1400	1100	890
400314	1800	1200	981
400339	2400	1500	1000
400345	3000	1721	1414
400455	4300	2400	1500
400876	5000	3500	1800
400999			2600
388652			2800

Click on a text object

Viewing Query Results

- **2- Chart Display of Results :**

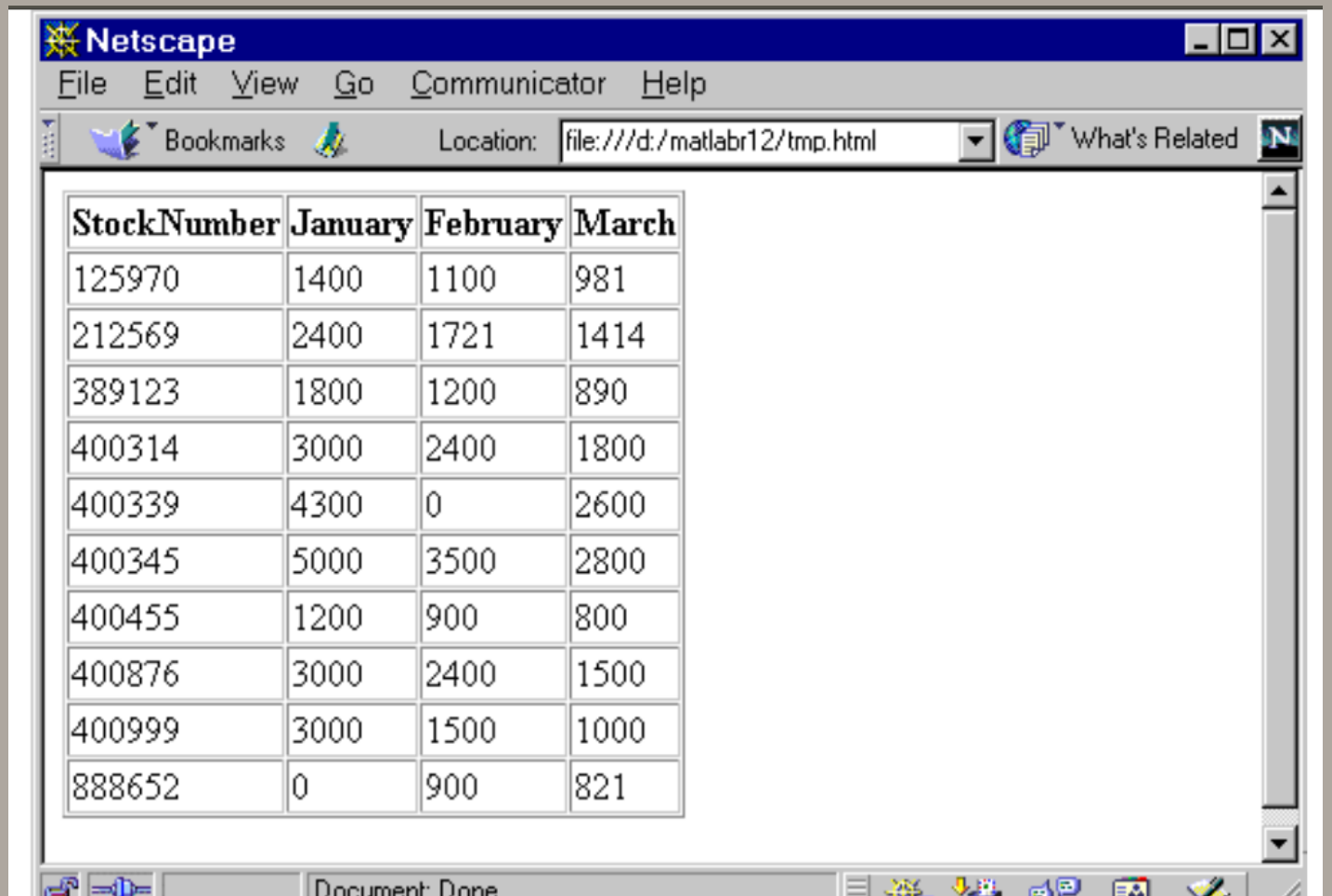
After executing a query, select Chart from the Display menu. The Charting dialog box appears.



Viewing Query Results

- **3-Report Display of Results in a Table :**

The report display presents the results in your system's default Web browser.



StockNumber	January	February	March
125970	1400	1100	981
212569	2400	1721	1414
389123	1800	1200	890
400314	3000	2400	1800
400339	4300	0	2600
400345	5000	3500	2800
400455	1200	900	800
400876	3000	2400	1500
400999	3000	1500	1000
888652	0	900	821

DataBase ToolBox Functions

- **About Objects and Methods for the Database Toolbox :**

  - Cursor .
  - Database .
  - Database metadata .
  - Driver .
  - Drivermanager .
  - Resultset .
  - Resultset metadata .

Importing Data into MATLAB from a Database

- **database : conn = database('SampleDB', ' ' , ' ')** → you define a MATLAB variable, conn , to be the returned connection object. This connection stays open until you close it with the close function. For the database function, you provide the name of the database, which is the data source SampleDB for this example. The other two arguments for the database function are username and password . For this example they are empty strings because the SampleDB database does not username or password.

- **exec : curs = exec(conn, 'select country from customers')** → Open a cursor and execute an SQL statement – type ,In the exec function, conn is the name of the connection object. The second argument, select country from customers , is a valid SQL statement that selects the country column of data from the customers table. The exec command returns a cursor object. In this example, you assign the MATLAB variable curs to the returned cursor object. curs =
Attributes: [] , Data: 0 , DatabaseObject: [1x1 database] ,RowLimit: 0 ,SQLQuery: 'select country from customers' ,Message: [] , Type: 'Database Cursor Object' , ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet] , Cursor: [1x1 com.mathworks.toolbox.database.sqlExec] ,Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement] , Fetch: 0 .

- **logintimeout : logintimeout(5)** → set the maximum time, in seconds, you want to allow the MATLAB session to try to connect to a database. This prevents the MATLAB session from hanging up if a database connection fails.

- **ping : ping(conn)** → Check the connection status – type , MATLAB returns status information about the connection, indicating that the connection was successful. Database Product Name: 'ACCESS' ,Database Product Version: '3.50.0000' , JDBC Driver Name: 'JDBC-ODBC Bridge (odbcjt32.dll)' , JDBCDriver Version: '1.1001 (04.00.4202)' , Max Database Connections: 64 , Current UserName: 'admin' , DatabaseURL: 'jdbc:odbc:SampleDB' .

Importing Data into MATLAB from a Database

- **Fetch : curs = fetch(curs, 10)** → *Import data into MATLAB – type , fetch is the function that imports data. It has the following two arguments*

** in this example:*

– curs : the cursor object returned by exec ,

– 10 : the maximum number of rows you want to be returned by fetch . The RowLimit argument is optional. If RowLimit is omitted, MATLAB imports all remaining rows.

- *close(curs)*
- *close(conn)*

- *conn = database('SampleDB', ' ', ' ');*
- *curs = exec(conn, 'select country from customers');*
- *curs = fetch(curs, 10);*

Exporting Data from MATLAB to a New Record in a Database

- insert(conn, 'Avg_Freight_Cost', colnames , exdata) :
  - -conn is the connection object .
  - -Avg_Freight_Cost is the name of the table .

Exporting Data from MATLAB, Replacing Existing Data in a Database

- update(conn, 'Avg_Freight_Cost', colnames, exdata, whereclause)

Examples

Example 1 – set rowLimit of Cursor

- conn=database('orcl','scott','tiger','oracle.jdbc.driver...OracleDriver','jdbc:oracle:thin:@144.212.33.228:1521:') ;

  curs=exec(conn, 'select * from EMP') ;

  set(curs, 'RowLimit' , 5) ;

  curs=fetch(curs) ;

Some phrases of Code and outputs

- datasource = 'MS SQL Server Auth';
- conn = database(datasource, ',' ')
- selectquery = 'SELECT * FROM inventoryTable';
- data = select(conn,selectquery);
- Data(1:3,:)

output :

Import all data from the table `inventoryTable` into MATLAB® using the `select` function
rows of data.

```
selectquery = 'SELECT * FROM inventoryTable';
data = select(conn,selectquery);
data(1:3,:)
```

```
ans =

    productNumber    Quantity    Price    inventoryDate
    _____________    ________    _____    _____________

    1                1700        15       '2014-09-23'
    2                1200         9       '2014-07-08'
    3                 356        17       '2014-05-14'
```

Some phrases of Code and outputs

Determine the highest quantity in the table.

```
max(data.quantity)
```

```
ans =

    9000
```

Close the database connection.

```
close(conn)
```

Some phrases of Code and outputs

Import all data from the `inventoryTable` using `conn`. Store the data in a cell array contained in the `Data` property of the `cursor` object. Display the data from `inventoryTable` in this property.

```
curs = exec(conn,'SELECT * FROM inventoryTable');
curs = fetch(curs);
curs.Data
```

```
ans =

    [ 1]    [1700]    [14.5000]    '2014-09-23 09:38...'
    [ 2]    [1200]    [      9]    '2014-07-08 22:50...'
    [ 3]    [ 356]    [     17]    '2014-05-14 07:14...'
    ...
```

Define a cell array containing the column name that you are updating.

```
colnames = {'Quantity'};
```

Define a cell array containing the new data `2000`.

```
data = {2000};
```

Update the column `Quantity` in the `inventoryTable` for the product with `productNumber` equal to `1`.

```
tablename = 'inventoryTable';
whereclause = 'WHERE productNumber = 1';

update(conn,tablename,colnames,data,whereclause)
```

Import the data again and view the updated contents in the `inventoryTable`.

```
curs = exec(conn,'SELECT * FROM inventoryTable');
curs = fetch(curs);
curs.Data
```

```
ans =

    [ 1]    [2000]    [14.5000]    '2014-09-23 09:38...'
    [ 2]    [1200]    [      9]    '2014-07-08 22:50...'
    [ 3]    [ 356]    [     17]    '2014-05-14 07:14...'
    ...
```

Some phrases of Code and outputs

Create an SQL script file named `salesvolume.sql` with this SQL query. This SQL query uses multiple joins to join these tables in the `dbtoolboxdemo` database:

- `producttable`
- `salesvolume`
- `suppliers`

The purpose of the query is to import sales volume data for suppliers located in the United States.

```
SELECT  salesvolume.January
,   salesvolume.February
,   salesvolume.March
,   salesvolume.April
,   salesvolume.May
,   salesvolume.June
,   salesvolume.July
,   salesvolume.August
,   salesvolume.September
,   salesvolume.October
,   salesvolume.November
,   salesvolume.December
,   suppliers.Country
FROM     ((producttable
INNER JOIN salesvolume
ON  producttable.stockNumber = salesvolume.StockNumber)
INNER JOIN suppliers
ON  producttable.supplierNumber = suppliers.SupplierNumber)
WHERE suppliers.Country LIKE 'United States%'
```

Run the SQL script file named `salesvolume.sql` using the `runsqlscript` function.

Some phrases of Code and outputs

Run the SQL script file named `salesvolume.sql` using the `runsqlscript` function.

```
results = runsqlscript(conn,'salesvolume.sql');
```

`results` is a `cursor` object array with the returned data from running the SQL query in the SQL script file.

Display the data in the `cursor` object containing the returned data.

```
results(1).Data
```

```
ans =

  Columns 1 through 8

    [5000.00]    [3500.00]    [2800.00]    [2300.00]    [1700.00]    [1400.00]    [1000.00]    [900.00]
    [2400.00]    [1721.00]    [1414.00]    [1191.00]    [ 983.00]    [ 825.00]    [ 731.00]    [653.00]
    [1200.00]    [ 900.00]    [ 800.00]    [ 500.00]    [ 399.00]    [ 345.00]    [ 300.00]    [175.00]
    ...

  Columns 9 through 13

    [1600.00]    [3300.00]    [12000.00]    [20000.00]    'United States'
    [ 723.00]    [ 790.00]    [ 1400.00]    [ 5000.00]    'United States'
    [ 760.00]    [1500.00]    [ 5500.00]    [17000.00]    'United States'
    ...
```

Display the column names for the returned data.

```
columnnames(results(1))
```

Some phrases of Code and outputs

Display the column names for the returned data.

```
columnnames(results(1))
```

```
ans =

'January','February','March','April','May','June','July','August',
'September','October','November','December','Country'
```

Close Database Connection

Close the `cursor` object array and database connection.

```
close(results)
close(conn)
```

Some phrases of Code and outputs

Export Data to New Record in Database

R2017a

This example does the following:

1. Retrieves sales data from a `salesVolume` table.
2. Calculates the sum of sales for 1 month.
3. Stores this data in a cell array.
4. Exports this data to a `yearlySales` table.

This example assumes that you are connecting to a Microsoft® Access™ database that contains tables named `salesVolume` and `yearlySales`. The table `salesVolume` contains the column names for each month. The table `yearlySales` contains the column names `Month` and `salesTotal`.

To access the code for this example, see `matlab\toolbox\database\dbdemos\dbinsertdemo.m`.

1. Create a database connection `conn` to the Microsoft Access database. For example, the following code assumes that you are connecting to a data source named `dbtoolboxdemo` with blank user name and password.

```
conn = database('dbtoolboxdemo','','');
```

2. Set the format for retrieved data to `numeric` by using `setdbprefs`.

```
setdbprefs('DataReturnFormat','numeric')
```

3. Execute the SQL query `sqlquery` using `conn` to import data for the `March` column from the `salesVolume` table. The `cursor` object `curs` contains the executed query. Import the data from the executed query using the `fetch` function.

```
sqlquery = 'SELECT March FROM salesVolume';

curs = exec(conn,sqlquery);
curs = fetch(curs);
```

4. The `Data` property of `curs` contains the imported data. Assign the data to the MATLAB® workspace variable `AA`. Display the data.

Some phrases of Code and outputs

4. The `Data` property of `curs` contains the imported data. Assign the data to the MATLAB® workspace variable AA. Display the data.

```
AA = curs.Data
```

```
AA =

981
1414
890
1800
2600
2800
800
1500
1000
821
```

5. Calculate the sum of the March sales. Assign the result to the MATLAB workspace variable `sumA`. Display the sum.

```
sumA = sum(AA(:))
```

```
sumA =

   14606
```

6. To export the data to the database, assign the month and sum of sales to a cell array. Put the month in the first cell of cell array `exdata`. Put the sum in the second cell of `exdata`.

```
exdata(1,1) = {'March'};
exdata(1,2) = {sumA}
```

```
exdata =
    'March'    [14606]
```

7. Define the names of the columns. Assign the cell array containing the column names to the MATLAB workspace variable `colnames`.

Some phrases of Code and outputs

7. Define the names of the columns. Assign the cell array containing the column names to the MATLAB workspace variable `colnames`.

```
colnames = {'Month','salesTotal'};
```

8. Determine the status of the `AutoCommit` database flag using `get`. This status determines if the exported data automatically commits to the database. If the flag is `off`, you can undo an insert. If the flag is on, data automatically commits to the database.

```
get(conn,'AutoCommit')
```

```
ans =
        on
```

The `AutoCommit` flag is set to on. The exported data automatically commits to the database.

9. Export the data into the `yearlySales` table using these arguments:

- Database connection `conn`
- Table name `yearlySales`
- Column names `colnames`
- Export data `exdata`

```
datainsert(conn,'yearlySales',colnames,exdata)
```

`datainsert` appends the data as a new record at the end of the `yearlySales` table.

10. In Microsoft Access, view the `yearlySales` table to verify the results.

yearlySales

Month	salesTotal	Revenue
March	14606	$0.00

11. After you finish working with the `cursor` object, close it.

```
close(curs)
```

The End

Thanks