

---

# Application of the Database Integration Toolbox

*The intent of this application example is to illustrate Maple™ techniques in a real world application context. Maple is a general-purpose environment capable of solving problems in any field that depends on mathematics and data. This application illustrates one possibility for this particular field. Note that there are many options within the Maple system to optimize the computations for specific problems.*

## Introduction

This worksheet demonstrates the use of the Database Integration Toolbox by loading a file into a database using Maple and then making use of database queries and the powerful technical analysis and programming tools offered in Maple to visualize the data and make predictions from the data.

The sample data comes from the Center for International Comparisons at the University of Pennsylvania<sup>1</sup>.

The following Maple techniques are highlighted.

- Loading a text file and extracting data from it
- Using the Database Toolbox to insert data into a database
- Using SQL queries to get information from a database
- Using advanced plotting functionality to visualize data
- Using the CurveFitting package to make predictions from the available information
- Creating a easy-to-use Maple™ interface for working with the data set

## Initialization

You can use this worksheet in two ways. You can browse the worksheet without executing it, which allows you to view it without setting up a database. Or, you can use the following instructions to set up a database and execute the worksheet's commands.

To execute this worksheet you will need to have a database server set up. From here, you can create a database to add the data to. You will also need to download the JDBC driver for your database. These are usually available for free download from your database vendor.

```
> restart;  
with(Database);
```

---

<sup>1</sup> Alan Heston, Robert Summers and Bettina Aten, *Penn World Table Version 6.1*, Center for International Comparisons at the University of Pennsylvania (CICUP), October 2002.  
<[http://pwt.econ.upenn.edu/php\\_site/pwt61\\_form.php](http://pwt.econ.upenn.edu/php_site/pwt61_form.php)>

*Sample Maple Application: Database  
Integration Toolbox*



Create a prepared statement to insert data into the database. A prepared statement creates an easily reusable SQL query into the table.

```
> addIndicator := connection:-CreatePreparedStatement(
    sprintf("INSERT INTO %s ( country, ind, indyr, data )
    VALUES( ?, ? , ?, ?)", tableName ) );
```

Finally, go through and insert the data into the database by executing the prepared statement.

```
> count := 0:
  for i from 1 to numLines-1 do
    curCountry := dd[i][1];
    for j from 3 to nops(dd[i] ) do
      # We check for ".." because this indicates
      # a data point which is unavailable
      if not ( dd[i][j] = ".." ) then
        addIndicator:-Execute( curCountry,
            indicators[j-2], dd[i][2], parse( dd[i][j] ) );
        count := count + 1;
      end if;
    end do;
  end do:
  connection:-Commit():
  print ( "Inserted " || count || " data elements into the database" );
  "Inserted 134107 data elements into the database"
```

## Read back the data

Now that all the information is in the database, use the database to query the available information. We will be calling `ExecuteQuery`, which calls a single SQL query on the table. We will be using the data to create a plot comparing several countries indicators. We will also use the data to call the `CurveFitting` package and find a prediction for the current indicator.

Read the countries that are in the database.

```
> countriesArray := connection:-ExecuteQuery(
    cat ( "SELECT DISTINCT ( country ) from ", tableName ),
    output=Array );
```

*countriesArray* :=  $\left[ \begin{array}{l} 1..195 \times 1..1 \text{ 2-D Array} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{array} \right]$

```

> countries := convert( LinearAlgebra:-Column (
    convert( countriesArray, Matrix ), 1 ), list );
countries := sort ( countries );
countries := ["Afghanistan", "Albania", "Algeria", "Angola", "Antigua", "Argentina",
    "Armenia", "Aruba", "Australia", "Austria", "Azerbaijan", "Bahamas", "Bahrain",
    "Bangladesh", "Barbados", "Belarus", "Belgium", "Belize", "Benin", "Bermuda",
    "Bhutan", "Bolivia", "BosniaandHerzegovina", "Botswana", "Brazil", "Brunei",
    "Bulgaria", "BurkinaFaso", "Burundi", "Cambodia", "Cameroon", "Canada",
    "CapeVerde", "CentralAfricanRepubl", "Chad", "Chile", "China", "Colombia",
    "Comoros", "CongoDem.Rep.", "CongoRepublicof", "CostaRica", "Coted'Ivoire",
    "Croatia", "Cuba", "Cyprus", "CzechRepublic", "Denmark", "Djibouti", "Dominica",
    "DominicanRepublic", "Ecuador", "Egypt", "ElSalvador", "EquatorialGuinea",
    "Eritrea", "Estonia", "Ethiopia", "Fiji", "Finland", "France", "FrenchPolynesia",
    "Gabon", "GambiaThe", "Georgia", "Germany", "Ghana", "Greece", "Greenland",
    "Grenada", "Guatemala", "Guinea", "Guinea-Bissau", "Guyana", "Haiti", "Honduras",
    "HongKong", "Hungary", "Iceland", "India", "Indonesia", "Iran", "Iraq", "Ireland",
    "Israel", "Italy", "Jamaica", "Japan", "Jordan", "Kazakhstan", "Kenya", "Kiribati",
    "KoreaRepublicof", "Kuwait", "Kyrgyzstan", "Laos", "Latvia", "Lebanon", "Lesotho",
    "Liberia", "Libya", "Lithuania", "Luxembourg", "Macao", "Macedonia",
    "Madagascar", "Malawi", "Malaysia", "Maldives", "Mali", "Malta", "MarshallIslands",
    "Mauritania", "Mauritius", "Mexico", "MicronesiaFed.Sts.", "Moldova", "Mongolia",
    "Morocco", "Mozambique", "Myanmar", "Namibia", "Nepal", "Netherlands",
    "NetherlandsAntilles", "NewCaledonia", "NewZealand", "Nicaragua", "Niger",
    "Nigeria", "Norway", "Oman", "Pakistan", "Palau", "Panama", "PapuaNewGuinea",
    "Paraguay", "Peru", "Philippines", "Poland", "Portugal", "PuertoRico", "Qatar",
    "Romania", "Russia", "Rwanda", "Samoa", "SaoTomeandPrincipe", "SaudiArabia",
    "Senegal", "Seychelles", "SierraLeone", "Singapore", "SlovakRepublic", "Slovenia",
    "SolomonIslands", "Somalia", "SouthAfrica", "Spain", "SriLanka", "St.Kitts&Nevis",
    "St.Lucia", "St.Vincent&Grenadine", "Sudan", "Suriname", "Swaziland", "Sweden",
    "Switzerland", "Syria", "Taiwan", "Tajikistan", "Tanzania", "Thailand", "Togo",
    "Tonga", "Trinidad&Tobago", "Tunisia", "Turkey", "Turkmenistan", "USA",
    "Uganda", "Ukraine", "UnitedArabEmirates", "UnitedKingdom", "Uruguay",
    "Uzbekistan", "Vanuatu", "Venezuela", "Vietnam", "VirginIslands(U.S.)",
    "WestBankandGaza", "Yemen", "Yugoslavia", "Zambia", "Zimbabwe"]

```

Count the number of operands in the list to determine the number of countries in the database.

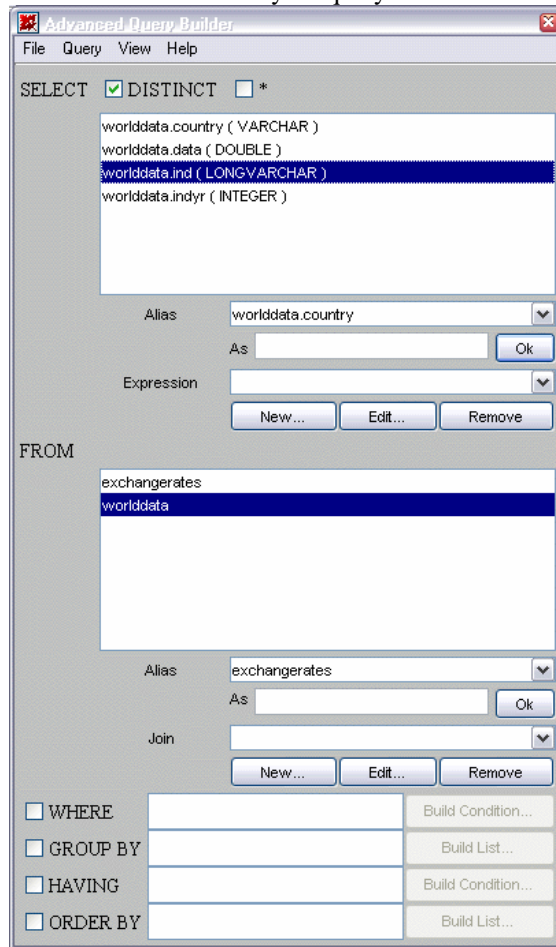
```
> nops ( countries );
```

195

Now we will extract all of the indicators that can be plotted using the Advanced Query Builder. This demonstrates the simple graphical methods the Database Toolbox provides for working with the database. To use the Query Builder for this example,

1. Execute **QueryBuilder(connection)**
2. Click **View**
3. Click **Advanced**
4. Click **DISTINCT**
5. Click the name of your table, in this case **worlddata**
6. Choose the column that you want to see, here it is **worlddata.ind**
7. Click **Query**
8. Click **Execute**
9. Click **To Array**

This returns an Array of all of the values that match your query.



Advanced Query Builder

```
> indicatorsArray := QueryBuilder(connection);
```

*indicatorsArray* := 1..23 x 1..1 2-D Array  
Data Type: anything  
Storage: rectangular  
Order: Fortran\_order

Sample Maple Application: Database  
Integration Toolbox

```

> indicators := convert( LinearAlgebra:-Column (
    convert( indicatorsArray, Matrix ), 1 ), list );
indicators := sort ( indicators );
indicators := [ "CGDPRelativeToTheUnitedStates(USD=100inCurrentPrices)" ,
    "ConsumptionShareOfCGPD(%inCurrentPrices)" ,
    "ConsumptionShareOfRGDPL(%in1996ConstantPrices)" ,
    "CurrentSavings(%inCurrentPrices)" , "ExchangeRate(USD=1)" ,
    "GovernmentShareOfCGDP(%inCurrentPrices)" ,
    "GovernmentShareOfRGDPL(%in1996ConstantPrices)" ,
    "InvestmentShareOfCGDP(%inCurrentPrices)" ,
    "InvestmentShareOfRGDPL(%in1996ConstantPrices)" ,
    "OpennessInCurrentPrices(%inCurrentPrices)" ,
    "OpennessinConstantPrices(%in1996ConstantPrices)" , "Population(1000s)" ,
    "PriceLevelOfConsumption(ppp/xrateInCurrentPrices)" ,
    "PriceLevelOfGovernment(ppp/xrateInCurrentPrices)" ,
    "PriceLevelOfGrossDomesticProduct(USD=100inCurrentPrices)" ,
    "PriceLevelOfInvestment(ppp/xrateInCurrentPrices)" ,
    "RatioOfGNPtoGDP(%inCurrentPrices)" ,
    "RealGDPChainPerEquivalentAdult(USDeq.AdultIn1996ConstantPrices)" ,
    "RealGDPChainPerWorker(USDworkerIn1996ConstantPrices)" ,
    "RealGDPperCapita(ConstantPrices:Chainseries)(USDin1996ConstantPrices)" ,
    "RealGDPperCapita(ConstantPrices:Laspeyres)(USDin1996ConstantPrices)" , "Re\
alGrossDomesticIncome(RGDPLadjustedForTermsOfTradechanges)(USDtermsO\
fTradein1996ConstantPrices)" ,
    "RealGrossDomesticProductPerCapita(USDinCurrentPrices)" ]

```

Count the number of operators in the list to determine the number of indicators in the database.

```
> nops ( indicators );
```

23

We can also make another easy SQL query to determine the minimum and maximum year that we have the data for.

```

> maxYear := connection:-ExecuteQuery( cat (
    "SELECT MAX( indyr ) FROM ", tableName ), output=Array )[1][1];
    maxYear := 2000

> minYear := connection:-ExecuteQuery( cat (
    "SELECT MIN( indyr ) FROM ", tableName ), output=Array )[1][1];
    minYear := 1950

```

# Analyze the data

Now we will make use of the extensive visualization, analysis and user interface tools in Maple to create a function, which will display a plot of the chosen indicator value for several countries. We will also write a Maplet interface for this function.

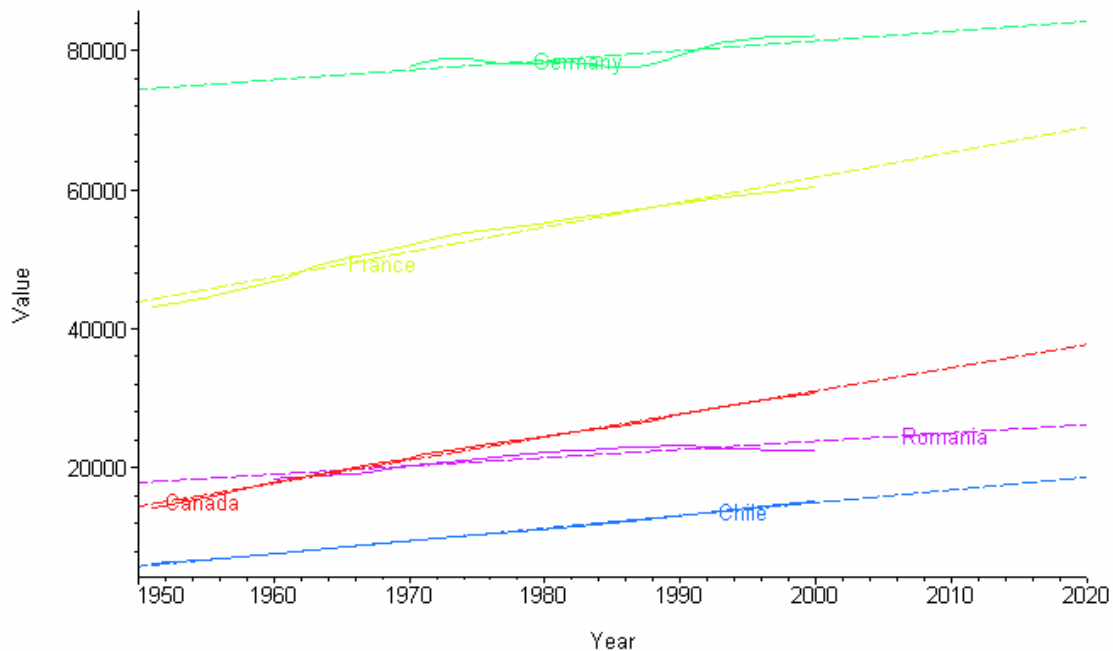
## WorldPlot

First we will write a function, which compares an indicator value for any number of countries. To use this function, pass a list of strings of country names, a string for the indicator to be plotted, the range, and a Boolean value to show the trend line. If the trend line is shown, it will be extended to cover the entire given range even if there is no data in that area.

The definition of the **WorldPlot** function has been hidden to simplify this example.

We will now execute the **WorldPlot** function by plotting the population of five different countries. This plot also shows a straight-line fit, which uses a Maple curve-fitting algorithm to fit the data and extend the line into the future. This simplistic “prediction” assumes a linear growth model; it does not take into account factors that affect population such as migration, economy, fertility, disease, or other factors that an in-depth analysis would consider. However, it serves to demonstrate how some of the analytical tools of Maple can be used in conjunction with this toolbox.

```
> WorldPlot( ["Canada", "France", "Germany", "Chile", "Romania"],  
             "Population(1000s)", minYear, maxYear + 20, true );  
Population(1000s)
```



Sample Maple Application: Database  
Integration Toolbox

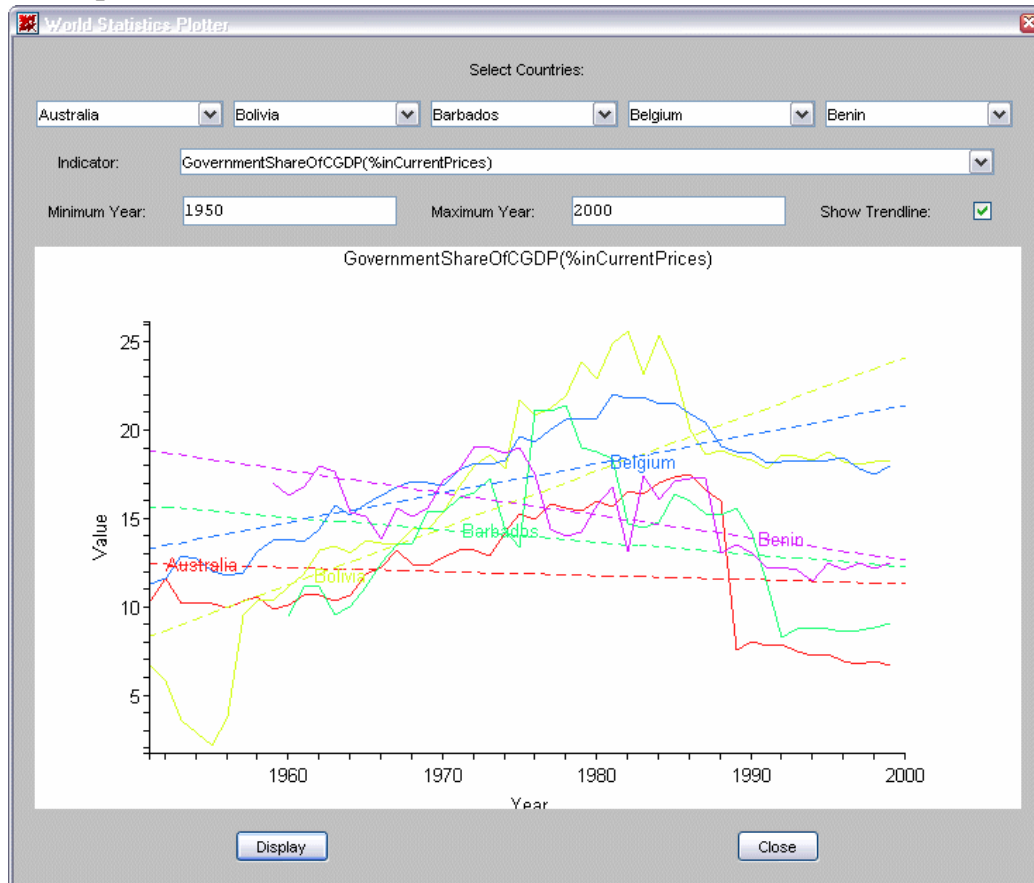
## WorldPlot Maplet Application

One of the great advantages of Maple is the simplicity of generating GUI interfaces to analyze information. We will now write a simple Maplet application, which uses the **WorldPlot** function to compare the different country's indicators.

The definition of the Maplet has been hidden to simplify this example.

Choose up to five countries to plot in the Maplet application, and pick any of the available indicators. You can choose what range you would like plotted and if the trend line should be shown. If you pick a year outside of the data range, the trend line will be extended to that year using the least squares curve fitting algorithm.

```
> WorldMaplet();
```



World Maplet

## Clean up the database

Remove the table from the database and close the connection.

```
> connection:-ExecuteUpdate( cat("DROP TABLE ", tableName) ):
connection:-Commit():
connection:-Close():
```