Databases

Introduction

Mathematica provides access to standard SQL relational database systems such as Oracle and MySQL via its DatabaseLink Package. A different sort of database application is provided in the WorkLife FrameWork™.

Mathematica's DatabaseLink provides an interface to the SQL language used by most relational database systems (and achieves this through using Java and other technologies behind the scenes) and it allows the programming of these interactions to be cast in the *Mathematica* programming language. The sorts of database tables that can be created are those that are supported by the SQL database in question and, often, entries into tables must adhere to the specific types allowed in the table fields. DatabaseLink is a very powerful technology for extending the reach of *Mathematica* into the world's databases and extending the ability for applications to do arbitrarily sophisticated calculations through *Mathematica*.

On the other hand, the WorkLife FrameWork's™ database functionality is quite different. It is entirely written in *Mathematica* and does not require any interface with external systems. There are no necessary restrictions on the types that are placed into its tables, and the tables are explicit *Mathematica* lists—therefore these lists can be used independently of having the WorkLife FrameWork™. It is a very useful and accessible tool for tasks that are not enterprise level.

The WorkLife FrameWork's™ database tool is used by the WorkLife FrameWork™ itself to store and access data on your use of it and your use of *Mathematica*.

The databases that you create with the WorkLife FrameWork's™ database tool are simple rectangular tables, and relationships between tables can be programmatically queried in the way that one usually writes programmatic queries in *Mathematica*.

In the following sections we will show basic and then more sophisticated use of the WorkLife FrameWork's™ database tool.

The Database Palette

For the buttons and executable commands that are described n this section to work it is assumed that you have installed the WorkLife FrameWork™ and have loaded it. This can be done either from the **Load WorkLife Framework** button on the supplied palette, by executing the command `Needs["Diary`Diary`"]`, or by clicking on the following button: **Load WorkLife FrameWork™**

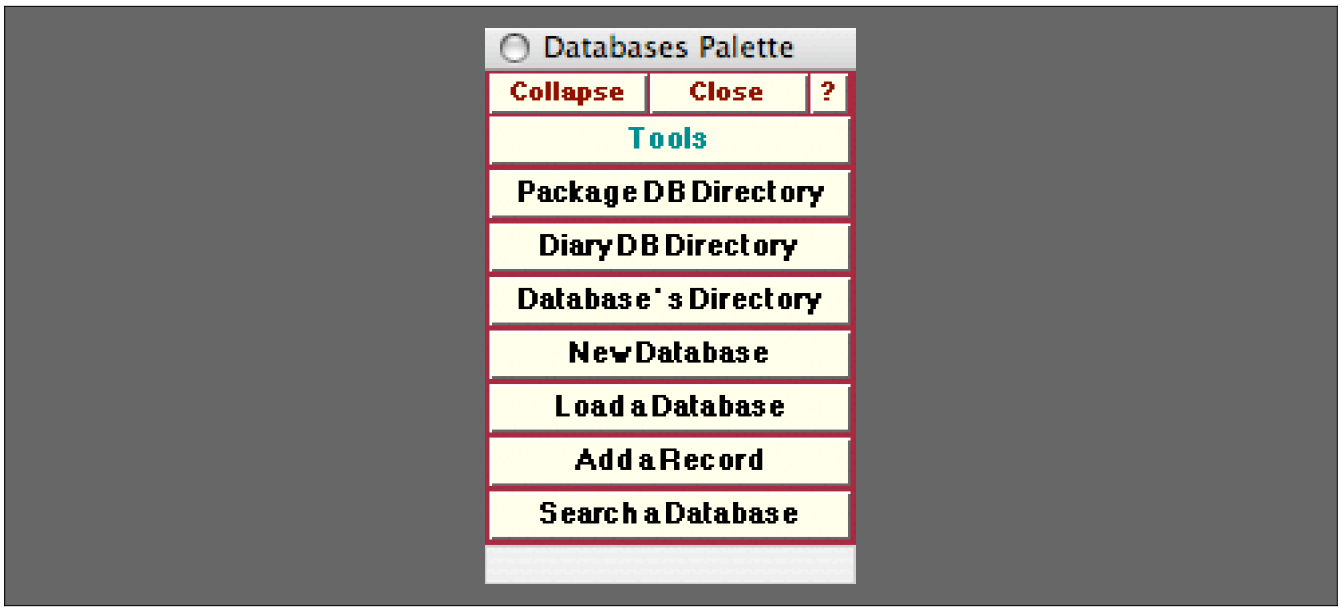
The Palette

The simplest interface to databases created in the WorkLife FrameWork™ is through the **Databases** palette. This can be accessed via its button on the **All Palettes** Palette, or by executing

```
DatabasesPalette[];
```

after the WorkLife FrameWork™ has been loaded.

The Palette looks like



The Databases Palette

The **Databases** Palette provides administrative buttons for managing Databases in the WorkLife FrameWork™ Package.

In this Palette each of the buttons below the **Databases** button, when clicked on, loads the indicated Database.

> *Although each Databases on this Palette is associated with a specific Diary (or group of Diaries that live in the same directory) the Databases listed in the* **Databases** *Palette are ones that are known to the* WorkLife FrameWork™ *Package. These are contained in the list* `$Databases`. *If a Database is not listed on this palette it can be loaded using the function* `LoadDatabase`.

The Palette Buttons

Package DB Directory

The WorkLife FrameWork™ Package has a global directory where Databases useful for the function of the package are located—the **Package DB Directory** button opens up this directory.

Diary DB Directory

The **Diary DB Directory** button opens the `Databases` subdirectory of the current Diary's directory that contains the directories of individual Dairies.

Database's Directory

This button opens a popup menu containing buttons listing the known databases that have been created. Clicking on one of these buttons will open the directory that that Database resides in.

This button opens a dialog for creating a **New Database**. See "Creating a Database" below.

This button opens a popup menu containing buttons listing the known databases that have been created. Clicking on one of these buttons will load that database.

This button opens a popup menu containing buttons listing the known databases that have been created. Clicking on one of these buttons will open a dialog Notebook that allows you enter a new record to add to the chosen database. When a button in the popup is clicked on the database will first be loaded if it has not been loaded yet.

This button opens a popup menu containing buttons listing the known databases that have been created. Clicking on one of these buttons will open a dialog Notebook that allows you enter information to base a search on for the chosen database. When a button in the popup is clicked on the database will first be loaded if it has not been loaded yet.

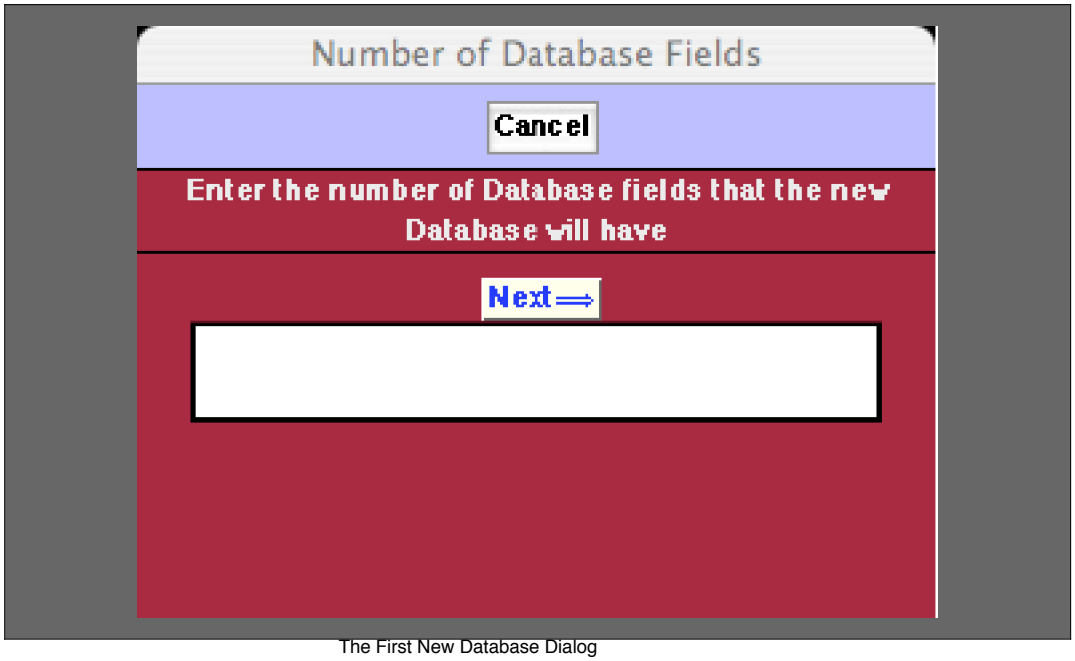
Creating a Database

For the buttons and executable commands that are described n this section to work it is assumed that you have installed the WorkLife FrameWork™ and have loaded it. This can be done either from the **Load WorkLife Framework** button on the supplied palette, by executing the command `Needs["Diary`Diary`"]`, or by clicking on the following button: **Load WorkLife FrameWork™**

...Via the Databases Palette

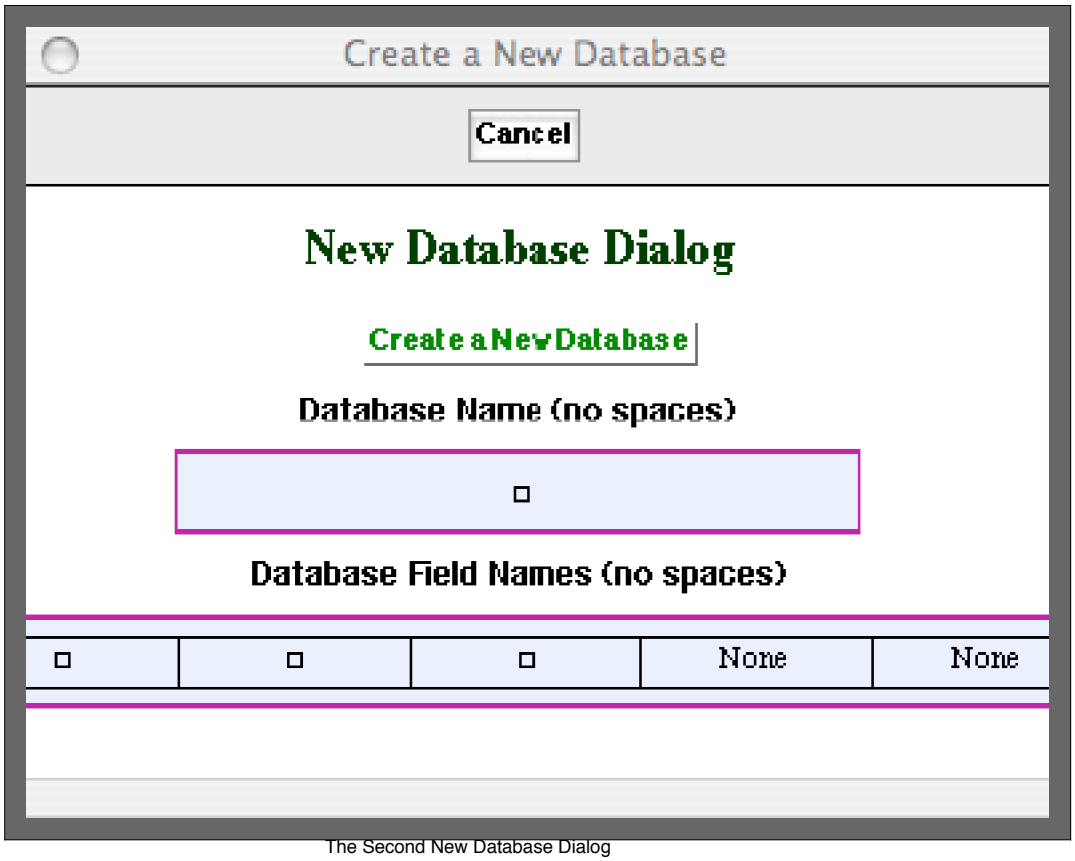
This button opens a dialog for creating a **New Database**. See "Creating a Database" below.

This dialog looks like:



The First New Database Dialog

After the number of Database fields is entered into this dialog and the Next⟹ button is pressed, a second dialog allows you to name the Database and enter the field names. In the following illustration of this second dialog the number of Database fields has been chosen to be 3.



The Second New Database Dialog

In this dialog you supply a name for the Database and Field Names for the indicated fields.

> *The boxes with* "None" *are just placeholders and generally should not be modified, though doing so will have no effect on the creation of the Database..*

When the **Create a New Database** button is clicked the new Database will be created in the Databases subdirectory of the current Diary's directory.

...Via the Function Interface

Although a database can be created via the **Databases** Palette and the dialogs that are presented after clicking on the **New Database** button, it's worth understanding the underlying functions that achieve these tasks so that you can understand the elements of a database.

To create a Database the WorkLife FrameWork™ must have been loaded and a Diary must be selected as the current Diary.

> *Databases are placed in the* Databases *subdirectory of a Diary's Directory. Hence, a Diary has to have been chosen as the current Diary in order for a Database to be created. Although a given database is located in a specific Diary's* Databases *subdirectory, all Databases that the* WorkLife FrameWork™ *is aware of are listed in the* **Databases** *Palette.*

The function to create a Database is `CreateDatabase`:

CreateDatabase[name, {records...}, fieldNames] creates a database with the name "name" containing
fieldNames, and this must be a list of distinct strings with length equal to the number of fields (colu
called "name" in the Database subdirectory of the current Diary directory. To create a new databas
containing a single record. Alternatively you can use CreateDatabase[name,fieldNames] to create a

Usage Message for `CreateDatabase`

To use this function you need to

- Name the database
- Decide how many fields its records will have, and
- Provide names for those records

As an example we will create a database called **`UserSurvey`**

```
CreateDatabase[UserSurvey,
  {"UserFirstName", "UserLastName", "UserAge", "RecordDate", "Rating"},
 FieldTypes → {_String, _String, _Integer?Positive, _?DateQ, _Integer?Positive}]
```

```
UserSurvey
```

> *Note that, in* `CreateDatabase`*, the Database's name is a symbol, but the FieldNames are all Strings. Be careful to not use, as a Database name, symbol that you either have already defined or that you plan to use. When the Database is created any earlier values for that symbol will be removed.*

The call to `CreateDatabase`, creates the database **`UserSurvey`** but does not add any records to it. The alternative form `CreateDatabase[name, {records...}, fieldNames]` seeds the database with the provided records. If the lengths of the records are not all the same, or if they are not the same length as the list of Field Names, then an error message is generated and the Database is not created.

In this example we have used the option `FieldTypes` to specify a list of patterns that each of the fields in a database record must match. The default, if the `FieldTypes` option is not specified, is for all of the field types to simply be the universal pattern `_`.

Interacting With a Database: Palette Interface

For the buttons and executable commands that are described n this section to work it is assumed that you have installed the WorkLife FrameWork™ and have loaded it. This can be done either from the **Load WorkLife Framework** button on the supplied palette, by executing the command **`Needs["Diary`Diary`"]`**, or by clicking on the following button: **Load WorkLife FrameWork™**

Loading the Database

When you click on the **Load a Database** button on the Databases Palette, a popup menu opens that lists the known databases that have been created. Clicking on one of these buttons will load that database. A database is known if it is contained in the list give by the parameter `$Databases`.

To load a database that is not listed in `$Databases` use the function `LoadDatabase` described below in the "Interacting With a Database: Function Interface" section of this document.

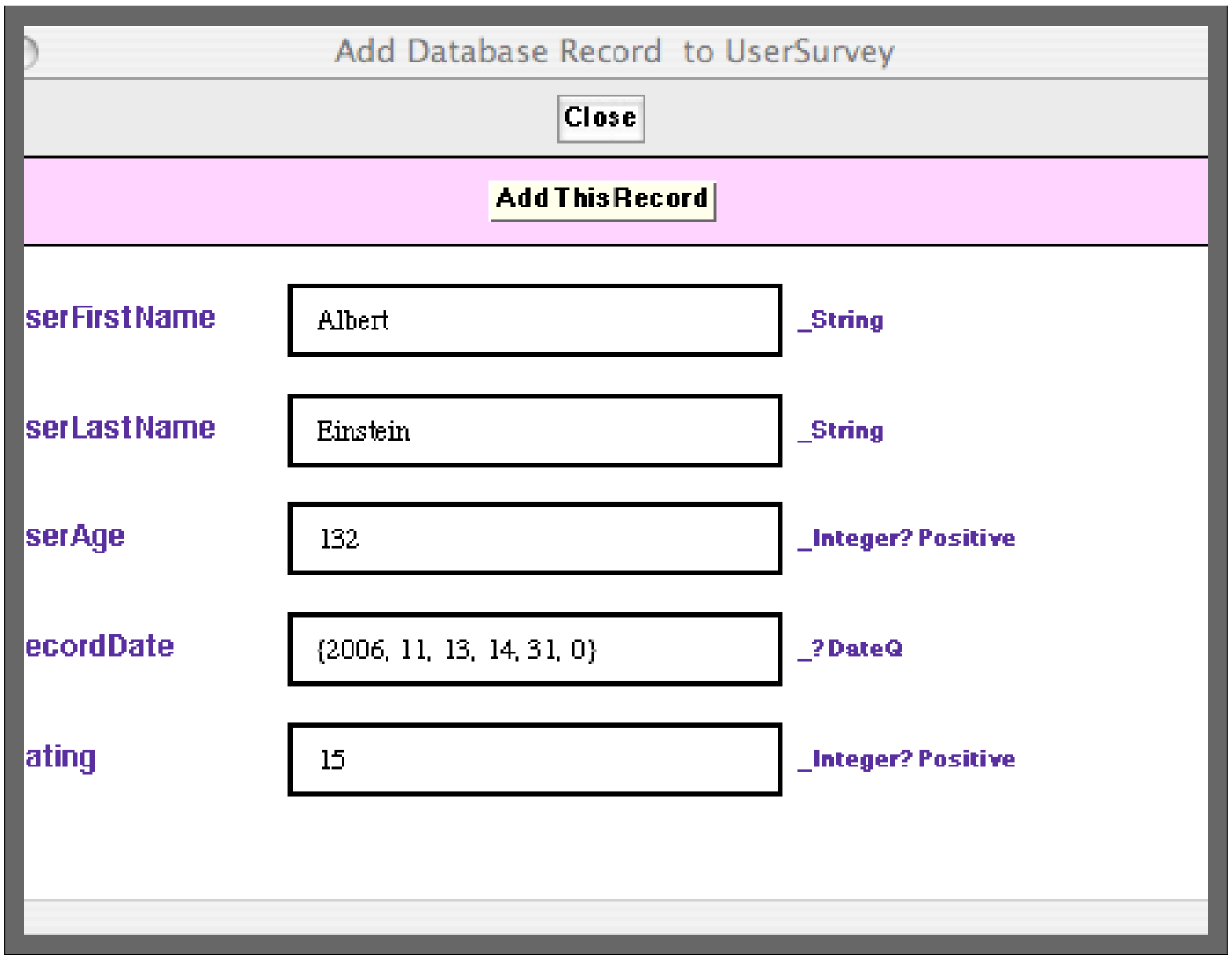
Adding records

When you click on the **Add a Record** button on the Databases Palette, a popup menu opens buttons listing the known databases that have been created. A database is known if it is contained in the list give by the parameter `$Databases`.

Clicking on one of these buttons will open a dialog Notebook that allows you enter a new record to add to the chosen database.

When a button in the popup is clicked on, the database will first be loaded if it has not been loaded yet.

For the example of the Database UserSurvey that we created in the preceding section, when the **Add a Record** button is clicked the following interface is opened:



An Example of the Interface to Add a Record to a Database

In this interface the FieldNames are listed along the left column. The boxed input fields accept the data for the respective field. And the right hand column shows the patterns that were assigned as the FieldTypes as a guide to what the form should be of the material entered into the input fields.

> *Note that, when a field's pattern is* `_String`*, you should not use the quotation characters around the field's entry in this dialog box interface. The entry will be assumed to be a string. If you do include quotation characters, they will be assumed to be part of the string itself.*

Searching a Database

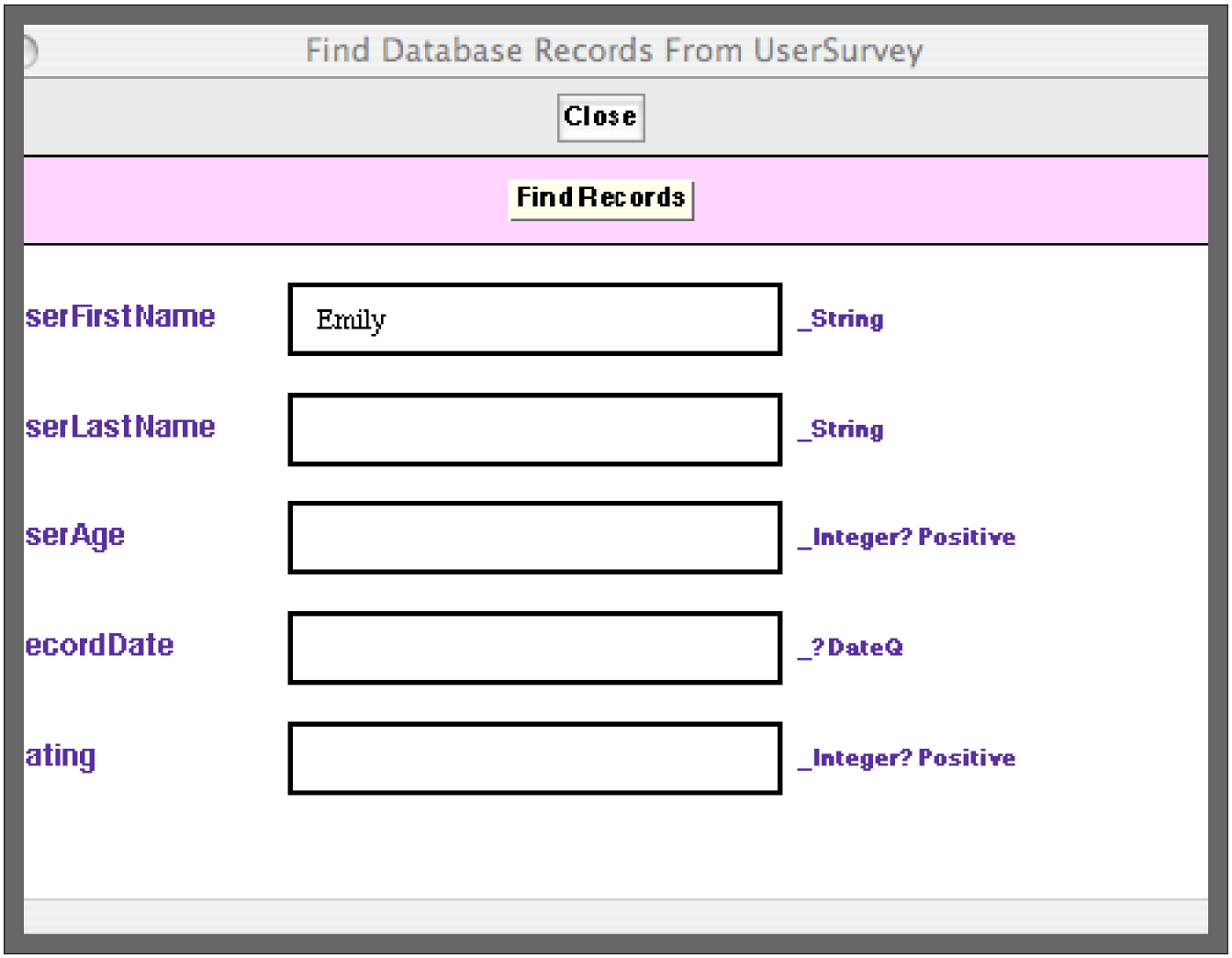
When you click on the **Search a Database** button on the Databases Palette, a popup menu opens buttons listing the known databases that have been created. A database is known if it is contained in the list give by the parameter `$Databases`.

Clicking on one of these buttons will open a dialog Notebook that allows you enter a new record to add to the chosen database.

When a button in the popup is clicked on, the database will first be loaded if it has not been loaded yet.

For the example of the Database UserSurvey that we created in the preceding section, when the **Search a Database** button is

clicked the following interface is opened:

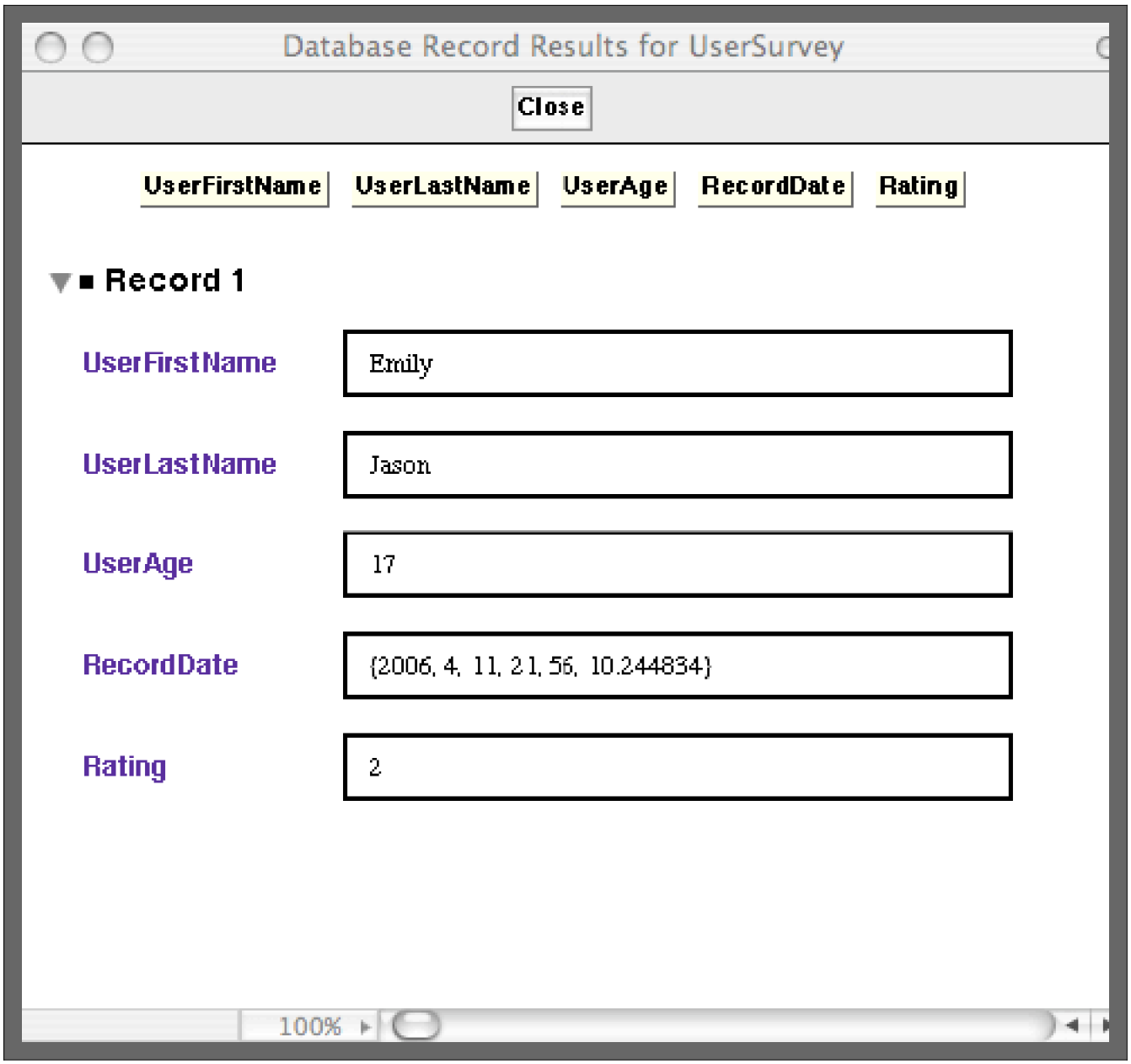


An Example of the Interface to Search for Records in a Database

The layout of this interface is similar to that for adding a record to a database with the columns having the same meaning. Database FieldNames are on the left, the input fields are in the boxes in the center, and the FieldTypes, as guidelines for your search, are in the column on the right.

> *Note again that, when a field's pattern is* `_String`*, you should not use the quotation characters around the field's entry in this dialog box interface. The entry will be assumed to be a string. If you do include quotation characters, they will be assumed to be part of the string itself.*

In this example we are looking for records where the user's first name is the String "Emily". The matching records are displayed in a notebook. For the UserSurvey database (with the records that are placed in this database in the next section) there is one record that matches this. Here is an example of the search results interface:



An Example of the Interface that Displays Search Results for a Database.

In this interface the buttons along the top, when clicked on, will open a notebook with the results for just the indicated field.

Interacting With a Database: Function Interface

For the buttons and executable commands that are described n this section to work it is assumed that you have installed the WorkLife FrameWork™ and have loaded it. This can be done either from the **Load WorkLife Framework** button on the supplied palette, by executing the command **Needs["Diary`Diary`"]**, or by clicking on the following button: **Load WorkLife FrameWork™**

Loading the Database

For a Database that has already been created, such as the **UserSurvey** Database that we created in the earlier section on Creating a Database, you load it using the `LoadDatabase` command.

LoadDatabase[name,file] loads a database residing in the file "file" and assigns it the name "name." I
it exists and is listed in $Databases. "name" must be a symbol without a value.

Usage Message for `LoadDatabase`

Here we load the **UserSurvey** Database:

```
LoadDatabase[UserSurvey]
```

Unloading the Database

If a database is already loaded and you want to unload it you can use the `UnloadDatabase` command.

UnloadDatabase[name] clears (unloads) the database "name" if it has been loaded. Database records a

Usage Message for `UnloadDatabase`

Here we unload the **UserSurvey** Database:

```
UnloadDatabase[UserSurvey]
```

```
UserSurvey
```

Reloading the Database

If a database is already loaded and you want to unload it you can use the `ReloadDatabase` command.

ReloadDatabase[name] reloads the database name if it has already been loaded.

Usage Message for `ReloadDatabase`

Here we load the **UserSurvey** Database and then Reload it:

```
LoadDatabase[UserSurvey]
```

```
UserSurvey
```

```
ReloadDatabase[UserSurvey]
```

```
UserSurvey
```

Generally there is no need to Reload a Database. However, reloading a Database has the effect of consolidating records that have been newly added and cleaning out any residual deleted records from the various files that make up the Database. In essence it has the effect of "cleaning house" for that Database.

Adding records

To add a record (or multiple records) to a Database use the `AddDatabaseRecords` function

AddDatabaseRecords[name,{records...}] adds the records to the database "name". These records are l
whereupon the records are added to the database file. Prior to that the new records are stored in an

Usage Message for `AddDatabaseRecords`

Since the Database **`UserSurvey`** is already loaded, we can add records to it. Here, for example, is a Record for Anne Gables:

{"UserFirstName","UserLastName","UserAge","RecordDate","Rating"}

```
AddDatabaseRecords[UserSurvey, {{"Anne", "Gables", 17, Date[], 3}}]
```

```
{{Anne, Gables, 17, {2006, 4, 11, 15, 39, 15.228215}, 3}}
```

Note several things

- The second argument to `AddDatabaseRecords` is a `List` of Records. In the case above we only added a single record. If we had (incorrectly used the form `AddDatabaseRecords[UserSurvey,{"Anne","Gables",17,Date[],3}]`—i.e., with out one of the sets of `{}` brackets—then an error message would have been generated.
- `AddDatabaseRecords` evaluates the list of Records before placing them in the Database. We took advantage of this to automatically insert into the Record's last field the date when the Record was added to the Database. This is because the function `Date[]` is evaluated by *Mathematica* at the point when `AddDatabaseRecords` is evaluated.
- When Database Records are found— for example by using `DatabaseFind`—the result that is returned is evaluated.
- If you want to add material to a Database without evaluating it you can wrap it in `Unevaluated`, `Hold`, or other similar functions. In these cases the results of, for example, `DatabaseFind` will also return unevaluated for those Database field entries that are wrapped in this way.
- The return value from the `AddDatabaseRecords` function is the list of Records that were added to the Database.

Deleting records

To delete a record (or multiple records) from a Database use the `DeleteDatabaseRecords` function.

DeleteDatabaseRecords[name,{records...}] deletes the records from the database "name". These reco
reloaded, whereupon the records are deleted from the database file. Prior to that the deleted records

Usage Message for `DeleteDatabaseRecords`

Here we delete the Anne Gables record that we added earlier.

```
DeleteDatabaseRecords[UserSurvey,
 {{"Anne", "Gables", 17, {2006, 4, 11, 15, 39, 15.228215`7.935223984853651}, 3}}]
```

```
{{Anne, Gables, 17, {2006, 4, 11, 15, 39, 15.228215}, 3}}
```

Note several things

- The records that are supplied to `DeleteDatabaseRecords` must be the exact records that you want to Delete from the Database. If we had supplied `{"Anne","Gables",17,Date[],3}` then the record would not have been deleted because the date field would have a different value. If you want to delete records that fit a particular pattern, then you should first `DatabaseFind` to locate those records and then supply them to `DeleteDatabaseRecords`.
- Just like `AddDatabaseRecords`, `DeleteDatabaseRecords` evaluates its second argument. Therefore if the final argument had been $\frac{9}{3}$ instead of 3, the Record would have still been deleted.
- The return value from the `DeleteDatabaseRecords` function is the list of Records that were deleted from the Database.

Modifying records

To modify an existing record in a Database use the `ModifyDatabaseRecord` function.

ModifyDatabaseRecord[name,originalRecord,replacementRecord] replaces the record originalRecord
than one copy of originalRecord in the database then all of them are replaced with replacementRec

Usage Message for `ModifyDatabaseRecord`

As an example first add a record to the database:

```
AddDatabaseRecords[UserSurvey, {{"Anne", "Gables", 17, Date[], 3}}]
```

```
{{Anne, Gables, 17, {2006, 4, 11, 16, 17, 1.708113}, 3}}
```

Here we use ModifyDatabaseRecord to change the "RecordDate" and "Rating" fields in the original record.

```
ModifyDatabaseRecord[UserSurvey,
 {"Anne", "Gables", 17, {2006, 4, 11, 16, 17, 1.708113`6.985091586887222}, 3},
 {"Anne", "Gables", 17, Date[], 2}]
```

```
{{Anne, Gables, 17, {2006, 4, 11, 16, 21, 38.865950}, 2}}
```

Note several things

- `ModifyDatabaseRecord` modifies a single Record at a time. To modify multiple records you need to write a simple function to perform that task.

- `ModifyDatabaseRecord` evaluates its second and third arguments.
- The return value from the `ModifyDatabaseRecord` function is the updated Record.

Adding Records when the Database is not loaded

There is an additional function that allows you to add records to a database even if it has not been loaded using `LoadDatabase`. Cleverly enough, this function is called `AddDatabaseRecordsToClosedDatabase`. This function is useful when you want to write an application that only needs to place data in a Database, but not to access that data.

Information about Database

The WorkLife FrameWork™ provides several functions that give you information about a Database.

Databases

The function `Databases` gives you a list of Databases in the current Diary's Databases subdirectory along with the path to the database file.

Note that the file that is given is not the only file that comprises the Database. For example, it doesn't contain information on records that have been added or deleted since the last time the Database was loaded (or reloaded).

> Databases[] gives a list of the databases in the current Diary's Database directory. The list is in the sa
> list with each element a list of length two. The first element of each entry is the name of the databa
> Databases[] is equivalent to Databases[Diary]. The databases in the package database directory ca
> not listed.

Usage Message for `Databases`

```
Databases[]
```

```
{{UserSurvey, /Users/dreiss/Documents/Mathem
    docs/VariousPackages/DiaryDevelopment/Diary/Documentation/English/Docs
    Development/Databases/UserSurvey/UserSurveyDB.m}}
```

DatabaseFieldNames

The function `DatabaseFieldNames` gives you a list of the FieldNames in the given Database.

> DatabaseFieldNames[name] gives the list of names of the fields in the given database if it has been lo
> the database. If i>RecordLength[name] then DatabaseFieldNames[name,i] returns $Failed.

Usage Message for `DatabaseFieldNames`

```
DatabaseFieldNames[UserSurvey]
```

```
{UserFirstName, UserLastName, UserAge, RecordDate, Rating}
```

DatabaseFile

The function `DatabaseFile` gives you a list of Databases in the current Diary's Databases subdirectory along with the path to the database file.

> DatabaseFile["directory"] gives the full path to a database file if it exists within the directory. If there

database file is one of the form "*DB.m".

Usage Message for DatabaseFile

DatabaseDirectory

The function DatabaseDirectory gives you a list of Databases in the current Diary's Databases subdirectory along with the path to the database file.

DatabaseDirectory[] gives the database directory in the current Diary directory. DatabaseDirectory[na

NumberOfDatabaseFields

The function NumberOfDatabaseFields gives you a list of Databases in the current Diary's Databases subdirectory along with the path to the database file.

NumberOfDatabaseFields[name] gives the number of fields in the database. The field names can be

Usage Message for NumberOfDatabaseFields

```
NumberOfDatabaseFields[UserSurvey]
```

```
5
```

NumberOfDatabaseRecords

The function NumberOfDatabaseRecords gives you a list of Databases in the current Diary's Databases subdirectory along with the path to the database file.

NumberOfDatabaseRecords[name] gives the number of records in the database.

Usage Message for NumberOfDatabaseRecords

```
NumberOfDatabaseRecords[UserSurvey]
```

```
25
```

DatabaseFileInDirectoryQ

The function DatabaseFileInDirectoryQ gives you a list of Databases in the current Diary's Databases subdirectory along with the path to the database file.

DatabaseFileInDirectoryQ["directory"] determines whether a database file exists within the directory

Usage Message for DatabaseFileInDirectoryQ

Searching a Database

Initialization: Populating the Database

You can retrieve the material stored in a Database by using the function DatabaseFind and making use of any of the found Records as input to any appropriate *Mathematica* commands.

DatabaseFind[name,item,"fieldName"] looks for item in the field of the database "name" with the fiel
match in that field. DatabaseFind[name,item,j] looks for item in the jth field of the database "name
the form of a list that is NumberOfDatabaseFields[name] long. DatabaseFind[name, Function|Data
function or a DatabasePattern.

Usage Message for `DatabaseFind`

There are a number of different ways in which `DatabaseFind` can be used. To show examples of these we first load and then populate our example Database **`UserSurvey`**.

First Load the database

```
LoadDatabase[UserSurvey]
```

```
UserSurvey
```

Next, add records to the database

```
AddDatabaseRecords[UserSurvey,
  {
   {"Pete", "Barton", 54, Date[], 8},
   {"Shirah", "Marcus", 14, Date[], 2},
   {"Juliette", "Capulet", 14, Date[], 4},
   {"Mark", "Wilson", 32, Date[], 8},
   {"Stephen", "Wolfram", 45, Date[], 9},
   {"Albert", "Einstein", Date[][[1]] - 1879, Date[], 10},
   {"Emily", "Jason", 17, Date[], 2},
   {"Fitzpatric", "Downturn", 39, Date[], 7},
   {"Blea", "Flapjack", 73, Date[], 8},
   {"Fsbif", "Tridd", 34, Date[], 6},
   {"Peter", "White", 34, Date[], 7},
   {"Laurie", "Dorsey", 23, Date[], 4},
   {"Leslie", "Lebowitz", 50, Date[], 7},
   {"Secaucus", "Seven", 49, Date[], 7},
   {"Bea", "Bee", 43, Date[], 8},
   {"Fourmi", "Antoine", 78, Date[], 9},
   {"Downey", "Fir", 13, Date[], 2},
   {"Twice", "Stated", 44, Date[], 7},
   {"Leviathan", "Ishmael", 99, Date[], 9},
   {"Sacher", "Torte", 16, Date[], 3},
   {"Bubonic", "Plague", 88, Date[], 7},
   {"Persimmon", "Gladness", 25, Date[], 5},
   {"Lisa", "Perchance", 18, Date[], 3},
   {"Swallow", "Barn", 62, Date[], 7}
  }];
```

This is the total number of Database Records in the UserSurvey Database. It includes the Records that we just added to the Database along with the one record that we added earlier.

```
NumberOfDatabaseRecords[UserSurvey]
```

```
25
```

Simple Search of the Database

The simplest use of the `DatabaseFind` function is to search for an exact match to the value in a specific Field position in Records.

It is easy to remind oneself what the field names are with the function `DatabaseFieldNames`:

```
DatabaseFieldNames[UserSurvey]
```

```
{UserFirstName, UserLastName, UserAge, RecordDate, Rating}
```

Here we look for all records where the age of the survey respondent is 34

```
DatabaseFind[UserSurvey, 34, "UserAge"]
```

```
{{Fsbif, Tridd, 34, {2006, 4, 11, 21, 56, 10.244857}, 6},
 {Peter, White, 34, {2006, 4, 11, 21, 56, 10.244865}, 7}}
```

An alternative version of this is to use the index of the FieldName (i.e., its position in the Record).

Here again we look for all records where the age of the survey respondent is 34, but indicate the Field that we are looking in by its place in the record

```
DatabaseFind[UserSurvey, 34, 3]
```

```
{{Fsbif, Tridd, 34, {2006, 4, 11, 21, 56, 10.244857}, 6},
 {Peter, White, 34, {2006, 4, 11, 21, 56, 10.244865}, 7}}
```

Search of the Database Based on a Record Pattern

Rather than looking for an exact match in a specific Field of the Database's Records, you can look more generally for Records that match a a specific pattern. As an example of this we look for Records where the user is older than 30 and whose Rating is less than or equal to 6.

This looks for Records where the user is older than 30 and whose Rating is less than or equal to 6.

```
DatabaseFind[UserSurvey, {_, _, _?(# > 30 &), _, _?(# ≤ 6 &)}]
```

```
{{Fsbif, Tridd, 34, {2006, 4, 11, 21, 56, 10.244857}, 6}}
```

This looks for Records where the user's first and last names contain the lower case letter "a".

```
DatabaseFind[UserSurvey,
 {_?(StringMatchQ[#, "*a*"] &), _?(StringMatchQ[#, "*a*"] &), _, _, _}]
```

```
{{Shirah, Marcus, 14, {2006, 4, 11, 21, 56, 10.244779}, 2},
 {Blea, Flapjack, 73, {2006, 4, 11, 21, 56, 10.244849}, 8},
 {Leviathan, Ishmael, 99, {2006, 4, 11, 21, 56, 10.244927}, 9},
 {Lisa, Perchance, 18, {2006, 4, 11, 21, 56, 10.244957}, 3},
 {Swallow, Barn, 62, {2006, 4, 11, 21, 56, 10.244965}, 7}}
```

Search of the Database Based on a Pure Function

Another approach that does not require writing out a *Mathematica* pattern that represents a full Database Record is to use a Pure Function. In the preceding examples suing a Record Pattern, Pure Functions naturally appeared in the pattern's specification. In this alternative approach the Pure Functions that are used have formal parameters (`#1,#2,...`) that correspond to each of the Database Fields respectively.

Again, as an example of this approach, we look for Records where the user is older than 30 and whose Rating is less than or equal to 6.

This looks for Records where the user is older than 30 and whose Rating is less than or equal to 6. Note that the Pure Function that is used must evaluate to True for those Records that you want to find. As a general rule the Pure Function should be a Boolean expression.

```
DatabaseFind[UserSurvey, ((#3 > 30) && (#5 ≤ 6)) &]
```

```
{{Fsbif, Tridd, 34, {2006, 4, 11, 21, 56, 10.244857}, 6}}
```

This is another way to write this that is slightly easier to read.

```
DatabaseFind[UserSurvey, Function[((#3 > 30) && (#5 ≤ 6))]]
```

```
{{Fsbif, Tridd, 34, {2006, 4, 11, 21, 56, 10.244857}, 6}}
```

Here we search for cases where the cube of the user's rating is less than the user's age

```
DatabaseFind[UserSurvey, Function[#5^3 < #3]]
```

```
{{Shirah, Marcus, 14, {2006, 4, 11, 21, 56, 10.244779}, 2},
 {Emily, Jason, 17, {2006, 4, 11, 21, 56, 10.244834}, 2},
 {Downey, Fir, 13, {2006, 4, 11, 21, 56, 10.244911}, 2}}
```

Search of the Database Based on a Database Pattern

One further approach that sometimes is easier to read is through the use of Database Patterns. These are similar to Pure Functions, except that in stead of formal parameters such as #1, #2, and so on..., the Database's field names are used explicitly. To do this each of the field names are placed in a `FieldName` wrapper and the full Database Pattern

As an example, the case above where the Pure Function was

```
Function[((#3 > 30) && (#5 ≤ 6))]
```

In terms of a `DatabasePattern` this becomes

```
DatabasePattern[((FieldName["UserAge"] > 30) && (FieldName["Rating"] ≤ 6))]
```

So, rather than `DatabaseFind[UserSurvey,Function[((#3>30)&&(#5≤6))],` in terms of a DatabasePattern the search becomes

```
DatabaseFind[UserSurvey,
 DatabasePattern[((FieldName["UserAge"] > 30) && (FieldName["Rating"] ≤ 6))]]
```

```
{{Fsbif, Tridd, 34, {2006, 4, 11, 21, 56, 10.244857}, 6}}
```

Note that, as always, FieldNames must be Strings.

Choosing the Entire Database

The form `DatabaseFind[name,All]` returns the full Database.

This returns the full database

```
DatabaseFind[UserSurvey, All]
```

```
{{Anne, Gables, 17, {2006, 4, 11, 21, 56, 10.244734}, 3},
 {Pete, Barton, 54, {2006, 4, 11, 21, 56, 10.244770}, 8},
 {Shirah, Marcus, 14, {2006, 4, 11, 21, 56, 10.244779}, 2},
 {Juliette, Capulet, 14, {2006, 4, 11, 21, 56, 10.244788}, 4},
 {Mark, Wilson, 32, {2006, 4, 11, 21, 56, 10.244796}, 8},
 {Stephen, Wolfram, 45, {2006, 4, 11, 21, 56, 10.244804}, 9},
 {Albert, Einstein, 127, {2006, 4, 11, 21, 56, 10.244825}, 10},
 {Emily, Jason, 17, {2006, 4, 11, 21, 56, 10.244834}, 2},
 {Fitzpatric, Downturn, 39, {2006, 4, 11, 21, 56, 10.244842}, 7},
 {Blea, Flapjack, 73, {2006, 4, 11, 21, 56, 10.244849}, 8},
 {Fsbif, Tridd, 34, {2006, 4, 11, 21, 56, 10.244857}, 6},
 {Peter, White, 34, {2006, 4, 11, 21, 56, 10.244865}, 7},
 {Laurie, Dorsey, 23, {2006, 4, 11, 21, 56, 10.244872}, 4},
 {Leslie, Lebowitz, 50, {2006, 4, 11, 21, 56, 10.244880}, 7},
 {Secaucus, Seven, 49, {2006, 4, 11, 21, 56, 10.244888}, 7},
 {Bea, Bee, 43, {2006, 4, 11, 21, 56, 10.244896}, 8},
 {Fourmi, Antoine, 78, {2006, 4, 11, 21, 56, 10.244903}, 9},
 {Downey, Fir, 13, {2006, 4, 11, 21, 56, 10.244911}, 2},
 {Twice, Stated, 44, {2006, 4, 11, 21, 56, 10.244919}, 7},
 {Leviathan, Ishmael, 99, {2006, 4, 11, 21, 56, 10.244927}, 9},
 {Sacher, Torte, 16, {2006, 4, 11, 21, 56, 10.244934}, 3},
 {Bubonic, Plague, 88, {2006, 4, 11, 21, 56, 10.244942}, 7},
 {Persimmon, Gladness, 25, {2006, 4, 11, 21, 56, 10.244950}, 5},
 {Lisa, Perchance, 18, {2006, 4, 11, 21, 56, 10.244957}, 3},
 {Swallow, Barn, 62, {2006, 4, 11, 21, 56, 10.244965}, 7}}
```

Using *Mathematica*'s `Part` function you can easily extract columns of the Database.

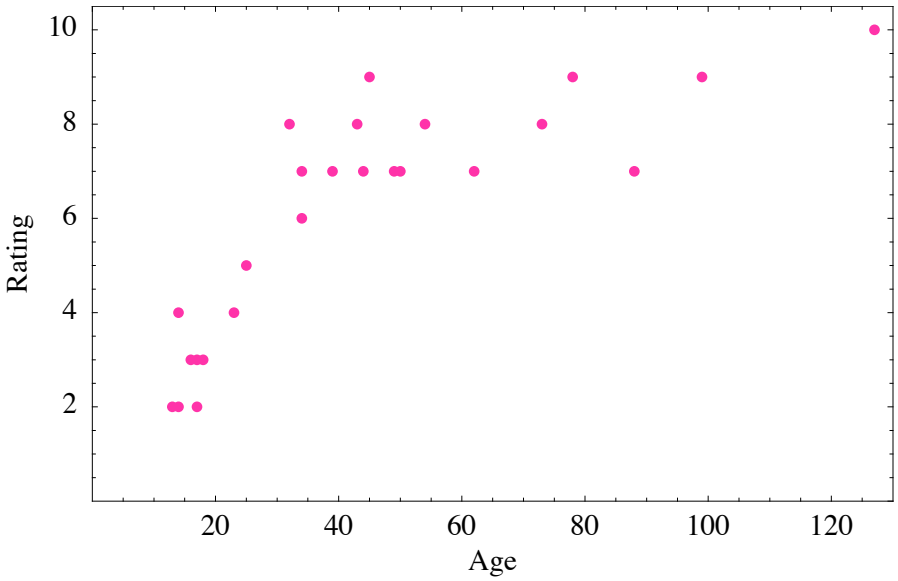
This gives the age—rating pairs

```
DatabaseFind[UserSurvey, All][[All, {3, 5}]]
```

```
{{17, 3}, {54, 8}, {14, 2}, {14, 4}, {32, 8}, {45, 9}, {127, 10}, {17, 2},
 {39, 7}, {73, 8}, {34, 6}, {34, 7}, {23, 4}, {50, 7}, {49, 7}, {43, 8},
 {78, 9}, {13, 2}, {44, 7}, {99, 9}, {16, 3}, {88, 7}, {25, 5}, {18, 3}, {62, 7}}
```

One can visualize this data using *Mathematica*'s `ListPlot`. There is a clear qualitative relationship between Age and Rating.

```
ListPlot[DatabaseFind[UserSurvey, All][[All, {3, 5}]],
  Frame → True,
  FrameLabel → {"Age", "Rating"},
  PlotStyle → {AbsolutePointSize[4], Hue[.9]},
  PlotRange → {{0, 130}, {0, 10.5}}];
```



The Output of DatabaseFind as a Notebook

`DatabaseFind` has an option, `GenerateNotebook`, that when set to `True` causes the records that were found to be displayed in a Notebook.

Backing up a Database

To back up the current state of a Database you can use the function `BackupDatabase`.

BackupDatabase[name] backs up the given database. The database must be currently loaded to do thi

Usage Message for `BackupDatabase`

`BackupDatabase` creates a copy of the Database in the same directory as the original Database. The copy's directory name is the same as that of the original Database, but with the characters "BU" appended along with an integer that repre-

sents a time stamp.