

dbshow 宏包 v1.2* \Rightarrow English Version

李昌锴 <lichangkai225@qq.com>

2022 年 1 月 8 日

Contents

1	引言	2
1.1	数据类型	2
1.2	与 datatool 的区别	3
2	接口文档	3
2.1	创建、展示和清空数据库	3
2.2	\dbNewStyle 和样式选项	4
2.2.1	通用选项	4
2.2.2	装饰器	5
2.3	使用 \dbNewReviewPoints 定义复习点	6
2.4	在 dbFilters 环境中定义过滤器	6
2.5	使用 dbitem 环境存储数据	7
2.6	\dbsave 和 \dbuse	8
2.7	条件判别式	8
2.8	特殊命令	8
3	错题本示例	8
4	Introduction	9
4.1	Data Types	9
4.2	Comparison to datatool	9

*代码仓库: <https://github.com/ZhiyuanLck/dbshow>, QQ 群: 788706534

5	Interfaces	10
5.1	Create, Display and Clear Database	10
5.2	\dbNewStyle and Style Options	10
5.2.1	General Options	11
5.2.2	Decorators	12
5.3	Use \dbNewReviewPoints to Define Review Points	13
5.4	Define Filters inside dbFilters Environment	13
5.5	Store Data with dbitem Environment	14
5.6	\dbsave and \dbuse	15
5.7	Conditionals	15
5.8	Special Macros	15
6	Example of Flaw Sweeper Template	16
	Change History	20
	Index	21

1 引言

编写本宏包的动机来源于当前没有一个很好的错题本宏包，可以方便的根据各种条件对错题进行筛选、排序，然后以自定义的样式展示出来。dbshow 宏包实现了四个核心功能：数据存储和使用、数据筛选、数据排序、数据展示。

数据只需要存储一次，就可以通过预定义的筛选、排序条件和样式展示部分或全部的数据。如上所述，本宏包其实实现了一个非常简单的数据库，复习错题的功能只是其中一个应用，和其他数据库宏包比如 datatool 相比，dbshow 更专注于非图表类型的数据展示。

1.1 数据类型

宏包基于 expl3 的基础类型构建了 6 种类型：

- date** 基于宏包 datetime2 的日期类型，以 iso 标准存储，支持大小比较，排序（转换成字符串）。默认值为 \Today。
- str** 字符串类型，支持正则匹配，英文排序。默认值为空。
- tl** $\langle token list \rangle$ 类型，支持正则匹配。默认值为空。
- int** 整数类型，支持大小比较，排序。默认值为 0。
- fp** 浮点数类型，支持大小比较，排序。默认值为 0。
- clist** 逗号分隔的列表类型。默认值为空列表。

1.2 与 datatool 的区别

从核心功能上看, `dbshow` 和 `datatool` 实现了相同的功能。区别在于 `dbshow` 基于 `expl3` 实现, 支持字符串的正则匹配, 还支持多级排序。使用方式上更倾向于样式与内容分离, 所有的样式都可以通过选项提前定义好并且可以复用。`dbshow` 并没有实现从外部文件读取数据以及将数据持久化的功能, 我认为这些应该是更专业的外部程序的工作而不应该在 `LATEX` 中设计这些功能。因此, `dbshow` 只提供了一个运行时的临时数据库, 足够轻便且满足大部分正常需求。如果你想删除或修改数据库中某一条记录, 请去对应的位置删除或修改掉对应的 `dbitem` 环境, 而不是让宏包提供一个输出某一行记录的命令。某种意义上记录数据库的 `TEX` 源文件本身就是数据的一种持久化。

2 接口文档

2.1 创建、展示和清空数据库

```
\dbNewDatabase [<base database>] [<database>] {  
\dbNewDatabase*  
  <attr1> = <type spec1>,  
  <attr2> = <type spec2>,  
  ...  
}  
\dbNewDatabase* [<database>] {  
  <attr1> = <type spec1>,  
  <attr2> = <type spec2>,  
  ...  
}
```

新建一个数据库, 不带星号的版本可以指定一个数据库来继承其属性设置, 该版本总是会舍弃掉之前的定义。

带星号的版本不会舍弃之前已有的定义, 而是将新的选项添加到后面。

<attr> 为属性名称, *<type spec>* 负责声明属性类型和属性默认值:

<attr> = *<type>* 将 *<attr>* 声明为 *<type>* 类型

<attr> = *<type>*|*<default>* 将 *<attr>* 声明为 *<type>* 类型, 并且将默认值设置为 *<default>*。

NOTE: 每个数据库都有一个默认的属性 `id` 用来存储数据的索引。

下面是定义一个错题数据库的示例, `question` 和 `answer` 属性用来存储问题和答案, `date` 属性存储日期, `info` 属性存储额外信息, `labels` 存储题目标签。

```
\dbNewDatabase{ques}{  
  question = tl,  
  answer = tl,  
  date = date,  
  info = tl,  
  labels = clist  
}
```

```
\dbshow [<style>] [<database>]  
使用 <style> 样式来展示 <database>。
```

```
\dbclear \dbclear {\<database>}
```

清空 $\langle database \rangle$ 里的所有内容。

2.2 \dbNewStyle 和样式选项

```
\dbNewStyle \dbNewStyle [{\<base styles>}] {\<style>} {\<database>} {\<opts>}
```

为 $\langle database \rangle$ 定义一个新的样式 $\langle style \rangle$ ，该样式可以基于已有的样式 $\langle base styles \rangle$ ，比如 `\dbNewStyle[base1, base2]{new-style}{ques}{}`。

2.2.1 通用选项

```
filter =  $\langle filter \rangle$  (initially -none-)
```

为当前样式设置由 `\dbCombineFilters` 所定义的过滤器

```
raw-filter =  $\langle cond\ combination \rangle$  (no value)
```

用已定义的条件为当前样式设置匿名过滤器。下面两个例子的作用是等同的。

```
% method 1
\begin{dbFilters}{db}
  \dbNewConditional{cond1}{int-attr}{\rval > 1}
  \dbNewConditional*{cond2}{str-attr}{\d+}
\end{dbFilters}
\dbNewStyle{style}{db}{raw-filter={cond1 \&\& cond2}}

% method 2
\begin{dbFilters}{db}
  \dbNewConditional{cond1}{int-attr}{\rval > 1}
  \dbNewConditional*{cond2}{str-attr}{\d+}
  \dbCombineFilters{filter}{cond1 \&\& cond2}
\end{dbFilters}
\dbNewStyle{style}{db}{filter=filter}
```

```
sort = {  $\langle attr\ spec1 \rangle$ ,  $\langle attr\ spec2 \rangle$ , ... } (no value)
```

为当前样式设置排序规则。支持根据 `str`, `date`, `int`, `fp` 类型的数据进行排序，支持多级排序。 $\langle attr \rangle$ 表示增序， $\langle attr \rangle^*$ 表示降序。下面例子中，使用 `sort-style` 展示数据时的顺序为先按 `level` 降序，`level` 相同的再按出生日期 `birth` 增序，以此类推。

```
\dbNewDatabase{sort-example}{
  name = str,
  birth = date,
  level = int,
  weight = fp,
}
\dbNewStyle{sort-style}{sort-example}{
  sort = { level*, birth, name, weight }
}
```

`item-code = <item code>` (no value)

该选项用来设置展示数据库中每条记录的代码。你可以使用 `\dbuse` 来展示属性的值。

```
<attr>/sep = <sep spec> (initially ,~)
<attr>/sep = <separator>
<attr>/sep = {
  <separator between two>,
  <separator between more than two>,
  <separator between final two>
}
```

该选项只适用于类型为 `clist` 的属性，用来设置列表间元素的间隔。第一个版本接受一个参数，将所有的元素间隔设置为 `<separator>`。第二个版本接受逗号分隔的三个参数，分别用来设置只有两个元素时的分隔符 `<separator between two>`，超过两个元素时的分隔符 `<separator between more than two>`，和最后两个元素之间的分隔符 `<separator between final two>`。

2.2.2 装饰器

下面这些选项在不同层次上装饰原有的展示代码，有些其实不必通过选项的形式来装饰，但这样做的好处是可以进一步使样式与内容分离。下面的例子中，`<style1>` 和 `<style2>` 是相同的样式，都用 `*` 将 `<attr1>` 包裹住了，但是如果你还想定义一个样式用 `=` 将 `<attr1>` 包裹住，如果用 `<style1>` 的方式，那就可能需要重复大片代码，用 `<style2>` 的方式则可以很轻松的继承 `<style1>` 中的代码。

```
\dbNewStyle{style1}{db}{
  item-code = {%
    *\rvuse{attr1}*\rvuse{attr2}
    % more code
  }
}
\dbNewStyle{base-style}{db}{
  item-code = {%
    \rvuse{attr1}\rvuse{attr2}
    % more code
  }
}
\dbNewStyle[base-style]{style2}{db}{
  attr1/before-code = { * },
  attr1/after-code = { * },
}
\dbNewStyle[base-style]{style3}{db}{
  attr1/before-code = { = },
  attr1/after-code = { = },
}
```

`before-code = <before code>` (no value)

该选项用来设置在展示整个数据库之前需要执行的代码。

`after-code = <after code>` (no value)

该选项用来设置在展示整个数据库之后需要执行的代码。

`record-before-code = <before code>` (no value)

该选项用来设置在展示当前记录之前需要执行的代码。

`record-after-code = <after code>` (no value)

该选项用来设置在展示当前记录之后需要执行的代码。

`<attr>/before-code = <before code>` (no value)

该选项用来设置展示数据库中属性 `<attr>` 对应数据之前需要执行的代码。`\dbuse` 会在展示属性数据前执行此代码。

`<attr>/after-code = <after code>` (no value)

该选项用来设置展示数据库中属性 `<attr>` 对应数据之后需要执行的代码。`\dbuse` 会在展示属性数据后执行此代码。

`<attr>/item-before-code = <before code>` (no value)

该选项只适用于类型为 `clist` 的属性，用来设置展示列表每个元素前需要执行的代码。

`<attr>/item-after-code = <after code>` (no value)

该选项只适用于类型为 `clist` 的属性，用来设置展示列表每个元素后需要执行的代码。

2.3 使用 `\dbNewReviewPoints` 定义复习点

`\dbNewReviewPoints` `\dbNewReviewPoints {<name>} {<points>}`

定义名为 `<name>` 的复习点。这是专门为错题本或复习所定制的功能，`<points>` 是一系列整数，现在假设每道错题你都将写错时的日期记录在了 `date` 属性中，并且你希望每隔 2, 5, 15 天复习一次。下面的代码给出了一个实现示例。

```
\dbNewReviewPoints{review-point}{2, 5, 15}                    % 定义复习点
\begin{dbFilters}
  \dbNewConditional{cond1}{date}{review-point|\Today} % 定义复习条件
  \dbCombineConditionals{filter1}{cond1}                    % 定义过滤器
\end{dbFilters}
\dbNewStyle{review-style}{ques}{filter=filter1}            % 定义展示样式
```

2.4 在 `dbFilters` 环境中定义过滤器

`dbFilters` `\begin{dbFilters}{<database>}`
 `<code>`
 `\end{dbFilters}`

`dbFilters` 用来定义过滤器，此环境中定义了 `\dbNewConditional` 命令用来定义条件和 `\dbCombineConditionals` 命令用来组合条件定义过滤器。过滤器独立于每个 `<database>`，这意味着你可以在不同数据库中定义名称相同的过滤条件和过滤器。

<hr/>	<code>\dbNewConditional</code>	<code>{<name>}</code>	<code>{<attr>}</code>	<code>{<cond spec>}</code>
<hr/>	<code>\dbNewConditional*</code>	<code>{<name>}</code>	<code>{<attr>}</code>	<code>{<cond spec>}</code>
	<code>\dbNewConditional</code>	<code>{<name>}</code>	<code>{<int/fp attr>}</code>	<code>{<relation>}</code>
	<code>\dbNewConditional*</code>	<code>{<name>}</code>	<code>{<int/fp attr>}</code>	<code>{<relation>}</code>
	<code>\dbNewConditional</code>	<code>{<name>}</code>	<code>{<str/tl attr>}</code>	<code>{<regex expr>}</code>
	<code>\dbNewConditional*</code>	<code>{<name>}</code>	<code>{<str/tl attr>}</code>	<code>{<regex expr>}</code>
	<code>\dbNewConditional</code>	<code>{<name>}</code>	<code>{<clist attr>}</code>	<code>{<val list>}</code>
	<code>\dbNewConditional*</code>	<code>{<name>}</code>	<code>{<clist attr>}</code>	<code>{<val list>}</code>
	<code>\dbNewConditional</code>	<code>{<name>}</code>	<code>{<date attr>}</code>	<code>{<review points> <date>}</code>
	<code>\dbNewConditional*</code>	<code>{<name>}</code>	<code>{<date attr>}</code>	<code>{<relation>}</code>

`\dbNewConditional` 用来定义名为 `<name>` 的条件, `<attr>` 指定条件所绑定的属性, 在 `<cond spec>` 中可以用 `\dbval` 指代属性的值。

对于类型为 `int` 和 `fp` 的属性, 两个版本的定义是一致的, `<relation>` 可以是单个关系式, 比如 `\dbval > 3`, 也可以是组合关系式, 比如 `\dbval > 3 && \dbval < 10.2`。支持的操作符有 `<`, `>`, `=`, `==`, `!=`, `>=`, `<=`, `!`。

对于类型为 `str` 和 `tl` 的属性, `<regex>` 为正则表达式, `\dbNewConditional` 表示部分匹配, `\dbNewConditional*` 表示整体匹配。

```
\dbNewConditional{cond1}{str-attr}{abc} % 匹配 abc, abcd, 1abc, =abc= 等
\dbNewConditional*{cond2}{str-attr}{abc} % 只匹配 abc
```

对于类型为 `clist` 的属性, 使用 `\dbNewConditional` 定义的条件只要 `<val list>` 中的任意一个元素在属性值 (列表) 中则条件成立; 使用 `\dbNewConditional*` 定义的条件只有 `<val list>` 中每一个值都在属性值 (列表) 中条件才成立。

```
\dbNewConditional{cond1}{clist-attr}{a, b, c} % a, b, d 满足条件
\dbNewConditional*{cond2}{clist-attr}{a, b, c} % a, b, d 不满足条件
```

对于类型为 `date` 的属性, `\dbNewConditional` 使用复习点来定义过滤条件, `<review points>` 是 `\dbNewReviewPoints` 定义的复习点, `<date>` 是用来比较的日期; `\dbNewConditional*` 使用单个关系式来定义过滤条件。支持的操作符有 `<`, `>`, `=`, `==`, `!=`, `>=`, `<=`。

<code>\dbCombineConditionals</code>	<code>\dbCombineConditionals</code>	<code>{<name>}</code>	<code>{<cond combination>}</code>	<code>[<info>]</code>
-------------------------------------	-------------------------------------	-----------------------------	---	-----------------------------

`\dbCombineConditionals` 定义名为 `{<name>}` 的过滤器, 并将 `\dbNewConditional` 定义的条件组合起来, 比如 `\dbCombineConditionals{filter}{(cond1 && cond2) || !cond3}`。`<cond combination>` 中可以使用的关系操作符为 `&&`, `||`, `!`。可以将 `filter` 选项设置为 `<name>` 来应用过滤器。`<info>` 为过滤器的相关信息, 在展示数据库的时候可以用 `\dbFilterInfo` 指代。

2.5 使用 dbitem 环境存储数据

```
dbitem \begin{dbitem}{<database>}[
    <attr1> = <val1>,
    <attr2> = <val2>,
    ...
]
<code>
\end{dbitem}
```

`dbitem` 环境用来存储数据。有两种存储数据的方法, 较短的数据可以在选项列表中通过键值对设置值, 较长的数据可以在 `<code>` 中使用 `\dbsave` 存储。`\dbsave` 会覆盖选项中设置的值。没有设置的值将会被设置为全局默认值, 下面给出一个存储示例。

```

\begin{dbitem}[date=2022-01-01, info= 测试]
  \dbsave{question}{这是一个测试问题}
  \dbsave{answer}{这是一个测试答案}
\end{dbitem}

```

2.6 \dbsave 和 \dbuse

```

\dbsave \dbsave {\attr} {\data}

```

\dbsave 用来存储数据，只能在 item 环境中使用。

```

\dbuse * \dbuse {\attr}

```

\dbuse 用来使用数据，只能在 item-code 选项中使用。 \dbuse 是可展开的。

2.7 条件判别式

```

\dbIfEmptyT * \dbIfEmptyTF {\true code} {\false code}
\dbIfEmptyF * \dbIfEmptyT {\true code}
\dbIfEmptyTF * \dbIfEmptyF {\false code}

```

该判别式用来判断当前数据库是否为空。下面的示例展示了如何预防空的列表环境。

```

\dbNewStyle{style-cond1}{database-test}{
  before-code = {\dbIfEmptyF{\begin{enumerate}}},
  after-code = {\dbIfEmptyF{\end{enumerate}}},
  item-code = {\item \dbuse{attr-test}}
}

```

2.8 特殊命令

dbshow 定义了一些特殊的命令，会根据语境展开为不同的内容。

\dbval	当前属性的值，只能在 \dbNewConditional 中使用
\dbDatabase	数据库名称
\dbFilterName	过滤器名称
\dbFilterInfo	过滤器信息
\dbIndex	原始数据索引，等同于 \dbuse{id} 但是 \dbuse 是不可展开的
\dbarabic	用数字表示的查询集数据计数
\dbalph	用小写字母表示的查询集数据计数
\dbAlph	用小写字母表示的查询集数据计数
\dbroman	用小写罗马数字表示的查询集数据计数
\dbRoman	用大写罗马数字表示的查询集数据计数

3 错题本示例

见第 6 节。

4 Introduction

The initial motivation to write this package is that I want to write a template, which can collect questions you gave the wrong answer and can display those questions you would like to review by some conditionals, such as questions with certain label, questions you have answered uncorrectly for certain times or questions having not been reviewed for certain days. So this package provides a database to do such thing.

The package provides four core functions: data storage and display, data filtering, data sorting and data display. All data is saved once and then you can display these data with custom filters, orders and styles.

4.1 Data Types

The package constructs 6 types based on the internal typed of `expl3`:

- date** date based on `datetime2` in iso format, supports comparison, sorting (converting to `str`), default `\Today`.
- str** string, supports regex match and sorting, default empty.
- tl** token list, supports regex match, default empty.
- int** integer, supports comparison and sorting, default 0.
- fp** floating point, supports comparison and sorting, default 0.
- clist** comma list, default empty.

4.2 Comparison to **datatool**

dbshow and **datatool** implement the same core functions. But **dbshow** is based on `expl3` and it supports string regex and multi-level sorting. **dbshow** tries to divide style from the contents (data in database): all styles are predefined and can be reused conveniently so that there can be only codes to save data and one-line code to show the database inside the `document` environment. You can hide the details in the preamble and focus on the data you want to display. **dbshow** provides a simple temporary runtime database, which means it can not input and output data from/to extern files (they should be responsible by some professional programming languages). When you need to delete or revise a record, just go to where it is recorded in the source code rather than use a macro to manipulate data after they are saved. In a sense, `TeX` file is also a kind of data persistence.

*Repository: <https://github.com/ZhiyuanLck/dbshow>, Telegram Group: https://t.me/latex_dbshow

5 Interfaces

5.1 Create, Display and Clear Database

<code>\dbNewDatabase</code>	<code>\dbNewDatabase [<i><base database></i>] {<i><database></i>} {</code>
<code>\dbNewDatabase*</code>	<code> <i><attr1></i> = <i><type spec1></i>,</code>
	<code> <i><attr2></i> = <i><type spec2></i>,</code>
	<code> ...</code>
	<code>}</code>
	<code>\dbNewDatabase* {<i><database></i>} {</code>
	<code> <i><attr1></i> = <i><type spec1></i>,</code>
	<code> <i><attr2></i> = <i><type spec2></i>,</code>
	<code> ...</code>
	<code>}</code>

Create a new database named *<database>*, unstarred form provides the optional *<base database>* from which current database inherit the attributes settings. The unstarred form always replace the old definition, while starred form appends the new options.

<attr> = *<type>*
<attr> = *<type>*|*<default>*

The first form defines the *<attr>* as *<type>*, and the second also sets the default value.

NOTE: Every database has a default attribute *id* to store the index of the item.

The example below define a database named **ques**.

```
\dbNewDatabase{ques}{
  question = tl, % store question
  answer = tl,  % store corresponding answer
  date = date,  % store the date when you were wrong
  info = tl,    % store extra info
  labels = clist % store question labels
}
```

<code>\dbshow</code>	<code>\dbshow {<i><style></i>} {<i><database></i>}</code>
	Show the <i><database></i> with <i><style></i> .

<code>\dbclean</code>	<code>\dbclean {<i><database></i>}</code>
	Clear the content of <i><database></i> .

5.2 \dbNewStyle and Style Options

<code>\dbNewStyle</code>	<code>\dbNewStyle [<i><base styles></i>] {<i><style></i>} {<i><database></i>} {<i><opts></i>}</code>
--------------------------	--

Define a new *<style>* that binds to *<database>*. The style can inherit from a list of *<base styles>* such as `\dbNewStyle[base1, base2]{new-style}{ques}{}`.

5.2.1 General Options

filter = $\langle filter \rangle$ (initially -none-)

Set the $\langle filter \rangle$ defined by `\dbCombineFilters`.

raw-filter = $\langle cond\ combination \rangle$ (no value)

Set anonymous with defined conditionals. Two examples below perform the same function.

```
% method 1
\begin{dbFilters}{db}
  \dbNewConditional{cond1}{int-attr}{\rval > 1}
  \dbNewConditional*{cond2}{str-attr}{\d+}
\end{dbFilters}
\dbNewStyle{style}{db}{raw-filter={cond1 && cond2}}

% method 2
\begin{dbFilters}{db}
  \dbNewConditional{cond1}{int-attr}{\rval > 1}
  \dbNewConditional*{cond2}{str-attr}{\d+}
  \dbCombineFilters{filter}{cond1 && cond2}
\end{dbFilters}
\dbNewStyle{style}{db}{filter=filter}
```

sort = { $\langle attr\ spec1 \rangle$, $\langle attr\ spec2 \rangle$, ... } (no value)

Set sorting rules. Attributes of type `str`, `date`, `int`, `fp` is supported to sort. Multi-level sort is allowed. $\langle attr \rangle$ represents for ascending order, and $\langle attr \rangle^*$ represents for descending order. The example below use four fields to determine the order of the records. It sorts on `level` in descending order first and if two `levels` are same then sorts on `birth` in ascending order and so on.

```
\dbNewDatabase{sort-example}{
  name = str,
  birth = date,
  level = int,
  weight = fp,
}
\dbNewStyle{sort-style}{sort-example}{
  sort = { level*, birth, name, weight }
}
```

item-code = $\langle item\ code \rangle$ (no value)

Set the code that show a single record. You can use `\dbuse` to display certian attribute.

$\langle attr \rangle / sep$ = $\langle sep\ spec \rangle$ (initially ,~)
 $\langle attr \rangle / separator$ = $\langle separator \rangle$
 $\langle attr \rangle / separator$ = {
 $\langle separator\ between\ two \rangle$,
 $\langle separator\ between\ more\ than\ two \rangle$,
 $\langle separator\ between\ final\ two \rangle$
}

Only for attributes of type `clist`. Set the separator between clist items. The first form accept one arguments and set the separator as $\langle sep \rangle$. The second form is more complicated, the following documentation is quoted from `interface3`:

If the comma list has more than two items, the $\langle separator\ between\ more\ than\ two \rangle$ is placed between each pair of items except the last, for which the $\langle separator\ between\ final\ two \rangle$ is used. If the comma list has exactly two items, then they are placed in the input stream separated by the $\langle separator\ between\ two \rangle$. If the comma list has a single item, it is placed in the input stream, and a comma list with no items produces no output.

5.2.2 Decorators

The options below serves as decorators. In some cases, decorator can also be encoded directly into `item-code` or some other places, which is convenient sometimes. The benefit of defining decorators with options is that styles step further to be divided with contents. In the examples below, $\langle style1 \rangle$ and $\langle style2 \rangle$ is the same style, which wrap $\langle attr1 \rangle$ with `*`. When you want another style which wrap $\langle attr1 \rangle$ with `=`, if you choose the way of $\langle style1 \rangle$, $\langle item\ code \rangle$ are repeated, otherwise if you choose the way of $\langle style2 \rangle$, $\langle item\ code \rangle$ is inherited and you only need define the decorators.

```
\dbNewStyle{style1}{db}{
  item-code = {%
    *\rvuse{attr1}*\rvuse{attr2}
    % more code
  }
}
\dbNewStyle{base-style}{db}{
  item-code = {%
    \rvuse{attr1}\rvuse{attr2}
    % more code
  }
}
\dbNewStyle[base-style]{style2}{db}{
  attr1/before-code = { * },
  attr1/after-code = { * },
}
\dbNewStyle[base-style]{style3}{db}{
  attr1/before-code = { = },
  attr1/after-code = { = },
}
```

before-code = $\langle before\ code \rangle$ (no value)

Set the $\langle before\ code \rangle$ that is executed before displaying the database.

after-code = $\langle after\ code \rangle$ (no value)

Set the $\langle after\ code \rangle$ that is executed after displaying the database.

record-before-code = $\langle before\ code \rangle$ (no value)

Set the $\langle \textit{before code} \rangle$ that is executed before displaying the record.

`record-after-code = $\langle \textit{after code} \rangle$` (no value)

Set the $\langle \textit{after code} \rangle$ that is executed after displaying the record.

`$\langle \textit{attr} \rangle$ /before-code = $\langle \textit{before code} \rangle$` (no value)

Set the $\langle \textit{before code} \rangle$ that is executed by `\dbuse` before displaying certain attribute.

`$\langle \textit{attr} \rangle$ /after-code = $\langle \textit{after code} \rangle$` (no value)

Set the $\langle \textit{after code} \rangle$ that is executed by `\dbuse` after displaying certain attribute.

`$\langle \textit{attr} \rangle$ /item-before-code = $\langle \textit{before code} \rangle$` (no value)

Only for attributes of type `clist`. Set the $\langle \textit{after code} \rangle$ that is excuted before displaying the item of the comma list.

`$\langle \textit{attr} \rangle$ /item-after-code = $\langle \textit{after code} \rangle$` (no value)

Only for attributes of type `clist`. Set the $\langle \textit{after code} \rangle$ that is excuted after displaying the item of the comma list.

5.3 Use `\dbNewReviewPoints` to Define Review Points

`\dbNewReviewPoints` `\dbNewReviewPoints $\{\langle \textit{name} \rangle\} \{\langle \textit{points} \rangle\}$`

Define the new $\langle \textit{points} \rangle$ that is specially designed for reviewing something. $\langle \textit{points} \rangle$ is a list of integers. Suppose you record the date when you did not answer correctly and you plan to review every 2, 5 and 15 days. The following code give what you want.

```
\dbNewReviewPoints{review-point}{2, 5, 15}           % define points
\begin{dbFilters}
  \dbNewConditional{cond1}{date}{review-point|\Today} % define conditional
  \dbCombineConditionals{filter1}{cond1}              % define filter
\end{dbFilters}
\dbNewStyle{review-style}{ques}{filter=filter1}      % define style
```

5.4 Define Filters inside `dbFilters` Environment

`dbFilters` `\begin{dbFilters} $\{\langle \textit{database} \rangle\}$`
 $\langle \textit{code} \rangle$
 `\end{dbFilters}`

Filters are defined inside `dbFilters` environment, inside which, `\dbNewConditional` is defined to declare conditionals and `\dbCombineConditionals` is defined to combine conditionals. Filters are independent in different databases, which means the same name of filters is allowed in different databases.

<code>\dbNewConditional</code>	<code>\dbNewConditional {<name>}{<attr>}{<cond spec>}</code>
<code>\dbNewConditional*</code>	<code>\dbNewConditional* {<name>}{<attr>}{<cond spec>}</code>

<code>\dbNewConditional</code>	<code>{<name>}{<int/fp attr>}{<relation>}</code>
<code>\dbNewConditional*</code>	<code>{<name>}{<int/fp attr>}{<relation>}</code>
<code>\dbNewConditional</code>	<code>{<name>}{<str/tl attr>}{<regex expr>}</code>
<code>\dbNewConditional*</code>	<code>{<name>}{<str/tl attr>}{<regex expr>}</code>
<code>\dbNewConditional</code>	<code>{<name>}{<clist attr>}{<val list>}</code>
<code>\dbNewConditional*</code>	<code>{<name>}{<clist attr>}{<val list>}</code>
<code>\dbNewConditional</code>	<code>{<name>}{<date attr>}{<review points> <date>}</code>
<code>\dbNewConditional*</code>	<code>{<name>}{<date attr>}{<relation>}</code>

Define the conditional named $\langle name \rangle$ that binds to $\langle attr \rangle$. `\dbval` is replaced with the real value of the attribute inside the $\langle cond\ spec \rangle$.

For attributes of type `int` and `fp`, two forms have the same definition. $\langle relation \rangle$ can be a single relation formula, such as `\dbval > 3`, or the combination of several relation formulas, such as `\dbval > 3 && \dbval < 10.2`. Supported operators are `<`, `>`, `=`, `==`, `!=`, `>=`, `<=`, `!`.

For attribute of type `str` and `tl`, unstarred form matches any part while starred form matches the whole part with the $\langle regex\ expr \rangle$.

```
\dbNewConditional{cond1}{str-attr}{abc} % match abc, abcd, 1abc, =abc=, etc
\dbNewConditional*{cond2}{str-attr}{abc} % only match abc
```

For attributes of type `clist`, the conditional defined by unstarred form is true if any item of $\langle val\ list \rangle$ is in the comma list. While the conditional defined by starred form is true only if every item of $\langle val\ list \rangle$ is in the comma list. As is showed below, for `cond1`, `a` is in `{a, b, d}` so `cond1` is true. While `c` is not in `{a, b, d}` so `cond2` is false.

```
\dbNewConditional{cond1}{clist-attr}{a, b, c} % {a, b, d} -> true
\dbNewConditional*{cond2}{clist-attr}{a, b, c} % {a, b, d} -> false
```

For attributes of type `date`, unstarred form uses $\langle review\ points \rangle$ to define the conditional and $\langle date \rangle$ is the date to be compared. The starred form define the conditional with **single** relation formula. Supported operators are `<`, `>`, `=`, `==`, `!=`, `>=`, `<=`.

<code>\dbCombineConditionals</code>	<code>\dbCombineConditionals {<name>}{<cond combination>}[<info>]</code>
-------------------------------------	--

Define the filter $\langle name \rangle$, which combine the conditionals and store the extra $\langle info \rangle$ into `\dbFilterInfo`. So you can write something as

```
\dbCombineConditionals{filter}{(cond1 && cond2) || !cond3}.
```

Supported operators are `&&`, `||`, `!`. You can set the option `filter` to $\langle name \rangle$ to apply the filter when you display the database.

5.5 Store Data with dbitem Environment

```
dbitem \begin{dbitem}{<database>}[
    <attr1> = <val1>,
    <attr2> = <val2>,
    ...
]
    <code>
\end{dbitem}
```

The data are stored with `dbitem` environment in two ways. Smaller data can be stored in the option list and the bigger data can be stored by `\dbsave`, which will suppress the value set by the option list. An example code is showned below.

```
\begin{dbitem}[date=2022-01-01, info=test]
  \dbsave{question}{This is a test question.}
  \dbsave{answer}{This is the correct answer of the question.}
\end{dbitem}
```

5.6 `\dbsave` and `\dbuse`

<code>\dbsave</code>	<code>\dbsave {<attr>} {<data>}</code>
----------------------	--

`\dbsave` save the `<data>` to `<attr>` of current record. `\dbsave` can be used only inside the `dbitem` environment.

<code>\dbuse *</code>	<code>\dbuse {<attr>}</code>
-----------------------	------------------------------------

Display the value of `<attr>` of current record. `\dbuse` is **expandable** and can be only used inside the option `item-code`.

5.7 Conditionals

<code>\dbIfEmptyT *</code>	<code>\dbIfEmptyTF {<true code>} {<false code>}</code>
<code>\dbIfEmptyF *</code>	<code>\dbIfEmptyT {<true code>}</code>
<code>\dbIfEmptyTF *</code>	<code>\dbIfEmptyF {<false code>}</code>

Test if the database is empty. The example below shows how to avoid an empty list environment.

```
\dbNewStyle{style-cond1}{database-test}{
  before-code = {\dbIfEmptyF{\begin{enumerate}}},
  after-code = {\dbIfEmptyF{\end{enumerate}}},
  item-code = {\item \dbuse{attr-test}}
}
```

5.8 Special Macros

Some special macros are defined to expand to different contents according to context.

<code>\dbval</code>	Attribute value, only accessible in <code>\dbNewConditional</code> .
<code>\dbDatabase</code>	Database name.
<code>\dbFilterName</code>	Filter name.
<code>\dbFilterInfo</code>	Filter information.
<code>\dbIndex</code>	Index of original records, which is equal to <code>\rvuse{id}</code> but <code>\rvuse</code> is nonexpandable.
<code>\dbarabic</code>	Show the counter of query set as digits.
<code>\dbalph</code>	Show the counter of query set as lowercase letters.

<code>\dbAlph</code>	Show the counter of query set as uppercase letters.
<code>\dbroman</code>	Show the counter of query set as roman numerals.
<code>\dbRoman</code>	Show the counter of query set as Roman numerals.

6 Example of Flaw Sweeper Template

```

\documentclass{article}
\usepackage{amsmath, physics}
\usepackage{geometry}
\usepackage{dbshow}
\usepackage{tikz}
\usepackage{tcolorbox}
\tcbuselibrary{skins}
\usetikzlibrary{shadings}
\usepackage[hidelinks]{hyperref}

\geometry{
  margin=2cm
}

% #1 link node #2 target node #3 text to show
\NewDocumentCommand \linktarget { m m m } {%
  \hyperlink{#1}{#3}%
  \raisebox{1em}{\hypertarget{#2}{}}%
}

% question box
\tcbset{
  base/.style={
    empty,
    frame engine=path,
    colframe=yellow!10,
    coltitle=red!70,
    fonttitle=\bfseries\sffamily,
    sharp corners,
    left=4pt,
    right=4pt,
    drop fuzzy shadow,
    drop fuzzy shadow,
    borderline west={3pt}{-3pt}{red!80},
  }
}

\newtcolorbox{mybox}[1]{%
  base, title = {#1}
}

\dbNewReviewPoints{review}{1, 3, 7, 15, 30, 60}

```



```

\dbNewDatabase{ques-book}{
  ques = tl,
  answer = tl,
  count = int|1,
  labels = clist,
  date = date,
}

\begin{dbFilters}{ques-book}
  \dbNewConditional{hard}{labels}{hard}
  \dbNewConditional{bad}{count}{\dbval > 1}
  \dbNewConditional{review}{date}{review|2022-01-07}
\end{dbFilters}

% show all questions with hyperlink to answers
\dbNewStyle{ques}{ques-book}{
  before-code = {\section{Questions}},
  item-code = {
    \begin{mybox}{%
      \linktarget{answer_\dbIndex}{ques_\dbIndex}{%
        Question \dbarabic%
        \hspace{2em}\dbuse{date}%
        \hspace{2em}\dbuse{labels}%
        \hfill\dbuse{count}%
      }%
    }
    \dbuse{ques}%
  }
  \end{mybox}
},
  labels/sep = /,
}

% show all questions and answers with hyperlink to questions
\dbNewStyle{answer}{ques-book}{
  before-code = {\section{Questions and Answers}},
  item-code = {
    \begin{mybox}{%
      \linktarget{ques_\dbIndex}{answer_\dbIndex}{%
        Question \dbarabic%
        \hspace{2em}\dbuse{date}%
        \hspace{2em}\dbuse{labels}%
        \hfill\dbuse{count}%
      }%
    }
    \dbuse{ques}\tcbsubtitle{Answer}\dbuse{answer}%
  }
  \end{mybox}
},
  labels/sep = /,
}

% show all hard questions with hyperlink to answers

```

```

\dbNameStyle{hard}{ques-book}{
  before-code = {\section{Hard Questions}},
  item-code = {
    \begin{mybox}{%
      \hyperlink{answer_\dbName}{%
        Question \dbName%
        \hspace{2em}\dbName{date}%
        \hspace{2em}\dbName{labels}%
        \hfill\dbName{count}%
      }%
    }
    \dbName{ques}%
  }
  \end{mybox}
},
raw-filter = hard,
labels/sep = /,
}

% show all hard questions that have answered incorrectly for more than one
% time with hyperlink to answers
\dbNameStyle[hard]{bad}{ques-book}{
  before-code = {\section{Bad Questions}},
  raw-filter = {bad && hard},
}

% show all questions that plan to be reviewed on 2022-01-07 with hyperlink to
% answers
\dbNameStyle[hard]{review}{ques-book}{
  before-code = {\section{Questions to be Reviewed}},
  raw-filter = {review},
}

\AtEndDocument{
  \dbshow{review}{ques-book}
  \dbshow{hard}{ques-book}
  \dbshow{bad}{ques-book}
  \dbshow{ques}{ques-book}
  \dbshow{answer}{ques-book}
}

\begin{document}

\begin{dbitem}{ques-book}[
  date=2022-01-01,
  labels={math, equation, easy},
  count=2
]
\dbsave{ques}{%
  Solve the linear equation:  $x + 16 = 31$ .
}
\dbsave{answer}{%
   $x = 31 - 16 = 15$ 
}

```

```

}
\end{dbitem}

\begin{dbitem}{ques-book}[
  date=2022-01-01,
  labels={math, equation, hard},
  count=3
]
\dbsave{ques}{%
  Solve the linear equation:  $2y = 16$ .
}
\dbsave{answer}{%
   $y = 16 / 2 = 8$ 
}
\end{dbitem}

\begin{dbitem}{ques-book}[
  date=2022-01-04,
  labels={math, integral, hard},
  count=1
]
\dbsave{ques}{%
  Find the indefinite integral:  $\int 2x \, dx$ .
}
\dbsave{answer}{%
   $\int 2x \, dx = x^2$ 
}
\end{dbitem}

\end{document}

```

1 Questions to be Reviewed

Question 1	2022-01-04	math/integral/hard	1
Find the indefinite integral: $\int 2x \, dx$.			

2 Hard Questions

Question 1	2022-01-01	math/equation/hard	3
Solve the linear equation: $2y = 16$.			
Question 2	2022-01-04	math/integral/hard	1
Find the indefinite integral: $\int 2x \, dx$.			

3 Bad Questions

Question 1	2022-01-01	math/equation/hard	3
Solve the linear equation: $2y = 16$.			

4 Questions

Question 1	2022-01-01	math/equation/easy	2
Solve the linear equation: $x + 16 = 31$.			
Question 2	2022-01-01	math/equation/hard	3
Solve the linear equation: $2y = 16$.			
Question 3	2022-01-04	math/integral/hard	1
Find the indefinite integral: $\int 2x \, dx$.			

5 Questions and Answers

Question 1	2022-01-01	math/equation/easy	2
Solve the linear equation: $x + 16 = 31$.			
Answer $x = 31 - 16 = 15$			

1

Question 2	2022-01-01	math/equation/hard	3
Solve the linear equation: $2y = 16$.			
Answer $y = 16/2 = 8$			

Question 3	2022-01-04	math/integral/hard	1
Find the indefinite integral: $\int 2x \, dx$.			
Answer $\int 2x \, dx = x^2$			

2

Change History

Date	Ver.	Describe
2022-01-05	1.0	First version Add macro: <code>\dbarabic</code> , <code>\dbalph</code> , <code>\dbAlph</code> , <code>\dbroman</code> , <code>\dbRoman</code>
2022-01-06		Fix bug: <code>\dbIndex</code> undefined Add style option: <code>raw-filter</code>
2022-01-07	1.1	Improve example
2022-01-07	1.1.1	Add macro: <code>\dbclear</code>
2022-01-07	1.1.2	Add documentation: comparison to <code>datatool</code>
2022-01-08	1.2	Fix str sorting bug Remove macros: <code>\dbItemIfEmpty(TF)</code> , <code>\dbClistItemIfEmpty(TF)</code> Make <code>\dbuse</code> expandable Add option: <code>record-before-code</code> , <code>record-after-code</code>

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
<code><attr>/after-code</code> (option)	<i>6, 13</i>
<code><attr>/before-code</code> (option)	<i>6, 13</i>
<code><attr>/item-after-code</code> (option)	<i>6, 13</i>
<code><attr>/item-before-code</code> (option)	<i>6, 13</i>
<code><attr>/sep</code> (option)	<i>5, 11</i>
A	
after-code (option)	<i>5, 12</i>
B	
before-code (option)	<i>5, 12</i>
D	
<code>\dbAlpha</code>	<i>8, 16</i>
<code>\dbAlph</code>	<i>20</i>
<code>\dbalph</code>	<i>8, 15</i>
<code>\dbalph</code>	<i>20</i>
<code>\dbarabic</code>	<i>8, 15</i>
<code>\dbarabic</code>	<i>20</i>
<code>\dbcClear</code>	<i>4, 10, 20</i>
<code>\dbClistItemIfEmpty(TF)</code>	<i>20</i>
<code>\dbCombineConditionals</code>	<i>6, 7, 13, 14</i>
<code>\dbCombineFilters</code>	<i>4, 11</i>
<code>\dbDatabase</code>	<i>8, 15</i>
<code>\dbFilterInfo</code>	<i>8, 15</i>
<code>\dbFilterInfo</code>	<i>7, 14</i>
<code>\dbFilterName</code>	<i>8, 15</i>
<code>dbFilters</code> (environment)	<i>6, 13</i>
<code>\dbIfEmptyF</code>	<i>8, 15</i>
<code>\dbIfEmptyT</code>	<i>8, 15</i>
<code>\dbIfEmptyTF</code>	<i>8, 15</i>
<code>\dbIndex</code>	<i>8, 15</i>
<code>\dbIndex</code>	<i>20</i>
<code>dbitem</code> (environment)	<i>7, 14</i>
<code>\dbItemIfEmpty(TF)</code>	<i>20</i>
<code>\dbNewConditional</code>	<i>6–8, 13–15</i>
<code>\dbNewConditional*</code>	<i>7, 14</i>
<code>\dbNewDatabase</code>	<i>3, 10</i>
<code>\dbNewDatabase*</code>	<i>3, 10</i>
<code>\dbNewReviewPoints</code>	<i>6, 7, 13</i>
<code>\dbNewStyle</code>	<i>4, 10</i>
<code>\dbRoman</code>	<i>8, 16</i>
<code>\dbRoman</code>	<i>20</i>
<code>\dbroman</code>	<i>8, 16</i>
<code>\dbroman</code>	<i>20</i>
<code>\dbsave</code>	<i>7, 8, 15</i>
<code>\dbshow</code>	<i>3, 10</i>
<code>\dbuse</code>	<i>5, 6, 8, 11, 13, 15, 20</i>
<code>\dbval</code>	<i>8, 15</i>
<code>\dbval</code>	<i>7, 14</i>
E	
environments:	
<code>dbFilters</code>	<i>6, 13</i>
<code>dbitem</code>	<i>7, 14</i>
F	
filter (option)	<i>4, 11</i>
filter	<i>7, 14</i>
I	
item-code (option)	<i>5, 11</i>
item-code	<i>8, 15</i>
O	
options:	
<code><attr>/after-code</code>	<i>6, 13</i>
<code><attr>/before-code</code>	<i>6, 13</i>
<code><attr>/item-after-code</code>	<i>6, 13</i>
<code><attr>/item-before-code</code>	<i>6, 13</i>
<code><attr>/sep</code>	<i>5, 11</i>
after-code	<i>5, 12</i>
before-code	<i>5, 12</i>
filter	<i>4, 11</i>
item-code	<i>5, 11</i>
raw-filter	<i>4, 11</i>
record-after-code	<i>6, 13</i>
record-before-code	<i>6, 12</i>
sort	<i>4, 11</i>
R	
raw-filter (option)	<i>4, 11</i>
raw-filter	<i>20</i>
record-after-code (option)	<i>6, 13</i>
record-after-code	<i>20</i>
record-before-code (option)	<i>6, 12</i>
record-before-code	<i>20</i>
<code>\rvuse</code>	<i>15</i>
S	
sort (option)	<i>4, 11</i>
T	
<code>\Today</code>	<i>2, 9</i>