dbshow 宏包[*] ⟹ English Version

李昌锴 <lichangkai225qq.com>

2022 年 1 月 3 日

Contents

[*] https://github.com/ZhiyuanLck/dbshow

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。

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` ⟨*token list*⟩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。

2 接口文档

2.1 \dbNewDatabase

`\dbNewDatabase` `\dbNewDatabase*`

```
\dbNewDatabase [⟨base database⟩] {⟨database⟩} {
  ⟨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
}
```

2.2 \dbNewStyle 和样式选项

\dbNewStyle

\dbNewStyle [⟨*base styles*⟩] {⟨*style*⟩} {⟨*database*⟩} {⟨*opts*⟩}

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

filter = ⟨*filter*⟩ (initially -none-)

为当前样式设置由 \dbCombineFilters 所定义的过滤器

sort = { ⟨*attr spec1*⟩, ⟨*attr spec2*⟩, … } (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。⟨*attr*⟩ 表示增序，⟨*attr*⟩* 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 }
}
```

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。

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*⟩/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*⟩/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*⟩。

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。

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。

\dbNewConditional
\dbNewConditional*

```
\dbNewConditional  {⟨name⟩}        {⟨attr⟩} {⟨cond spec⟩}
\dbNewConditional* {⟨name⟩}        {⟨attr⟩} {⟨cond spec⟩}
\dbNewConditional  {⟨name⟩} {⟨int/fp attr⟩} {⟨relation⟩}
\dbNewConditional* {⟨name⟩} {⟨int/fp attr⟩} {⟨relation⟩}
\dbNewConditional  {⟨name⟩} {⟨str/tl attr⟩} {⟨regex expr⟩}
\dbNewConditional* {⟨name⟩} {⟨str/tl attr⟩} {⟨regex expr⟩}
\dbNewConditional  {⟨name⟩}  {⟨clist attr⟩} {⟨val list⟩}
\dbNewConditional* {⟨name⟩}  {⟨clist attr⟩} {⟨val list⟩}
\dbNewConditional  {⟨name⟩}   {⟨date attr⟩} {⟨review points⟩|⟨date⟩}
\dbNewConditional* {⟨name⟩}   {⟨date attr⟩} {⟨relation⟩}
```

\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 <，>，=，==，!=，>=，<=。

\dbCombineConditionals

```
\dbCombineConditionals {⟨name⟩} {⟨cond combination⟩} [⟨info⟩]
```

\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⟩}`
\dbuse `\dbuse {⟨attr⟩}`

\dbsave 用来存储数据,只能在 item 环境中使用。\dbuse 用来使用数据,只能在 item-code 选项中使用。

2.7 使用 \dbshow 展示数据库

\dbshow `\dbshow {⟨style⟩} {⟨database⟩}`

使用 ⟨*style*⟩ 样式来展示 ⟨*database*⟩。

2.8 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}}
}
```

\dbItemIfEmptyT `\dbItemIfEmptyTF {⟨true code⟩} {⟨false code⟩}`
\dbItemIfEmptyF `\dbItemIfEmptyT {⟨true code⟩}`
\dbItemIfEmptyTF `\dbItemIfEmptyF {⟨false code⟩}`

该判别式用来判断当前元素是否为空。下面的示例展示了如何在展示 database-test 数据库中 text 属性的元素时在元素非空的时候前后都加上 *。

```
\dbNewStyle{style-cond2}{database-test}{
  text/before-code = {\dbItemIfEmptyF{*}},
  text/after-code = {\dbItemIfEmptyF{*}},
}
```

\dbClistItemIfEmptyT `\dbClistItemIfEmptyTF {⟨true code⟩} {⟨false code⟩}`
\dbClistItemIfEmptyF `\dbClistItemIfEmptyT {⟨true code⟩}`
\dbClistItemIfEmptyTF `\dbClistItemIfEmptyF {⟨false code⟩}`

该判别式用来判断列表属性中的元素是否为空。下面的示例展示了如何在展示 database-test 数据库中 labels 属性（标签列表）的元素时在标签非空的时候前后都加上 *。

```
\dbNewDatabase{database-test}{labels=clist}
\dbNewStyle{style-cond3}{database-test}{
  labels/item-before-code = {\dbClistItemIfEmptyF{*}},
  labels/item-after-code = {\dbClistItemIfEmptyF{*}},
}
```

2.9 常量

\dbDatabase
\dbFilterName
\dbFilterInfo
\dbIndex

\dbDatabase 指代数据库名，\dbFilterName 指代过滤器名称，\dbFilterInfo 指代过滤器额外信息，\dbIndex 指代当前索引。

3 错题本示例

见第 6 节。

Package dbshow [*]

⟹ 中文版本

Li Changkai <lichangkai225qq.com>

2022/01/03

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.

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.

5 Interfaces

5.1 \dbNewDatabase

`\dbNewDatabase` `\dbNewDatabase*`

```
\dbNewDatabase [⟨base database⟩] {⟨database⟩} {
  ⟨attr1⟩ = ⟨type spec1⟩,
  ⟨attr2⟩ = ⟨type spec2⟩,
  …
}
\dbNewDatabase* {⟨database⟩} {
  ⟨attr1⟩ = ⟨type spec1⟩,
  ⟨attr2⟩ = ⟨type spec2⟩,
  …
}
```

[*] https://github.com/ZhiyuanLck/dbshow

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
}
```

5.2 \dbNewStyle and Style Options

\dbNewStyle

`\dbNewStyle [⟨base styles⟩] {⟨style⟩} {⟨database⟩} {⟨opts⟩}`

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}{}`.

`filter` = ⟨*filter*⟩ (initially -none-)

Set the ⟨*filter*⟩ defined by `\dbCombineFilters`.

`sort` = { ⟨*attr spec1*⟩, ⟨*attr spec2*⟩, … } (no value)

Set sorting rules. Attributes of type `str, date, int, fp` is supported to sort. Multi-level sort is allowed. ⟨*attr*⟩ represents for ascending order, and ⟨*attr*⟩* 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 }
}
```

`before-code` = ⟨*before code*⟩ (no value)

Set the ⟨*before code*⟩ that is executed before displaying the database.

`after-code` = ⟨*after code*⟩ (no value)

Set the ⟨*after code*⟩ that is executed after displaying the database.

`item-code` = ⟨*item code*⟩ (no value)

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

⟨*attr*⟩`/before-code` = ⟨*before code*⟩ (no value)

Set the ⟨*before code*⟩ that is executed by `\dbuse` before displaying certain attribute.

⟨*attr*⟩`/after-code` = ⟨*after code*⟩ (no value)

Set the ⟨*after code*⟩ that is executed by `\dbuse` after displaying certain attribute.

⟨*attr*⟩`/sep` = ⟨*sep spec*⟩ (initially ,~)

⟨*attr*⟩`/separator` = ⟨*separator*⟩

```
⟨attr⟩/separator = {
  ⟨separator between two⟩,
  ⟨separator between more than two⟩,
  ⟨separator between final two⟩
}
```

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

> If the comma list has more than two items, the ⟨*separator between more than two*⟩ is placed between each pair of items except the last, for which the ⟨*separator between final two*⟩ is used. If the comma list has exactly two items, then they are placed in the input stream separated by the ⟨*separator between two*⟩. 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.

`item-before-code` = ⟨*before code*⟩ (no value)

Only for attributes of type `clist`. Set the ⟨*after code*⟩ that is excuted before displaying the item of the comma list.

`item-after-code` = ⟨*after code*⟩ (no value)

Only for attributes of type `clist`. Set the ⟨*after code*⟩ that is excuted after displaying the item of the comma list.

5.3 Use `\dbNewReviewPoints` to Define Review Points

`\dbNewReviewPoints` `\dbNewReviewPoints {⟨name⟩} {⟨points⟩}`

Define the new ⟨*points*⟩ that is specially designed for reviewing something. ⟨*points*⟩ 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}{⟨database⟩}
  ⟨code⟩
\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.

`\dbNewConditional`
`\dbNewConditional*`

```
\dbNewConditional  {⟨name⟩}        {⟨attr⟩} {⟨cond spec⟩}
\dbNewConditional* {⟨name⟩}        {⟨attr⟩} {⟨cond spec⟩}
\dbNewConditional  {⟨name⟩} {⟨int/fp attr⟩} {⟨relation⟩}
\dbNewConditional* {⟨name⟩} {⟨int/fp attr⟩} {⟨relation⟩}
\dbNewConditional  {⟨name⟩} {⟨str/tl attr⟩} {⟨regex expr⟩}
\dbNewConditional* {⟨name⟩} {⟨str/tl attr⟩} {⟨regex expr⟩}
\dbNewConditional  {⟨name⟩}  {⟨clist attr⟩} {⟨val list⟩}
\dbNewConditional* {⟨name⟩}  {⟨clist attr⟩} {⟨val list⟩}
\dbNewConditional  {⟨name⟩}   {⟨date attr⟩} {⟨review points⟩|⟨date⟩}
\dbNewConditional* {⟨name⟩}   {⟨date attr⟩} {⟨relation⟩}
```

Define the conditional named ⟨*name*⟩ that binds to ⟨*attr*⟩. `\dbval` is replaced with the real value of the attribute inside the ⟨*cond spec*⟩.

For attributes of type `int` and `fp`, two forms have the same definition. ⟨*relation*⟩ 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 ⟨*regex expr*⟩.

```
\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 ⟨*val list*⟩ is in the comma list. While the conditional defined by starred form is true only if every item of ⟨*val list*⟩ 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 ⟨*review points*⟩ to define the conditional and ⟨*date*⟩ is the date to be compared. The starred form define the conditional with **single** relation formula. Supported operators are `<, >, =, ==, !=, >=, <=`.

`\dbCombineConditionals`

```
\dbCombineConditionals {⟨name⟩} {⟨cond combination⟩} [⟨info⟩]
```

Define the filter ⟨*name*⟩, which combine the conditionals and store the extra ⟨*info*⟩ into `\dbFilterInfo`. So you can write something as
`\dbCombineConditionals{filter}{(cond1 && cond2) || !cond3}`.
Supported operators are `&&, ||, !`. You can set the option `filter` to ⟨*name*⟩ 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`

\dbsave `\dbsave {⟨attr⟩} {⟨data⟩}`

\dbuse `\dbuse {⟨attr⟩}`

Date is stored with `\dbsave` and is displayed with `\dbuse`. `\dbsave` can be used only inside the `dbitem` environment and `\dbuse` can be only used inside the option `item-code`.

5.7 Use `\dbshow` to Display the Database

\dbshow `\dbshow {⟨style⟩} {⟨database⟩}`

Show the ⟨*database*⟩ with ⟨*style*⟩.

5.8 Conditionals

\dbIfEmptyT `\dbIfEmptyTF {⟨true code⟩} {⟨false code⟩}`

\dbIfEmptyF `\dbIfEmptyT {⟨true code⟩}`

\dbIfEmptyTF `\dbIfEmptyF {⟨false 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}}
}
```

\dbItemIfEmptyT
\dbItemIfEmptyF
\dbItemIfEmptyTF

\dbItemIfEmptyTF {⟨*true code*⟩} {⟨*false code*⟩}
\dbItemIfEmptyT {⟨*true code*⟩}
\dbItemIfEmptyF {⟨*false code*⟩}

Test if the value of the attribute is empty. The example belows shows how to surround the non-empty `text` attribute with the symbol *.

```
\dbNewStyle{style-cond2}{database-test}{
  text/before-code = {\dbItemIfEmptyF{*}},
  text/after-code = {\dbItemIfEmptyF{*}},
}
```

\dbClistItemIfEmptyT
\dbClistItemIfEmptyF
\dbClistItemIfEmptyTF

\dbClistItemIfEmptyTF {⟨*true code*⟩} {⟨*false code*⟩}
\dbClistItemIfEmptyT {⟨*true code*⟩}
\dbClistItemIfEmptyF {⟨*false code*⟩}

Test if the item of comma list is empty. The example belows shows how to surround the non-empty label with the symbol *.

```
\dbNewDatabase{database-test}{labels=clist}
\dbNewStyle{style-cond3}{database-test}{
  labels/item-before-code = {\dbClistItemIfEmptyF{*}},
  labels/item-after-code = {\dbClistItemIfEmptyF{*}},
}
```

5.9 Constants

\dbDatabase
\dbFilterName
\dbFilterInfo
\dbIndex

`\dbDatabase` represents for the database name, `\dbFilterName` represents for the filter name, `\dbFilterInfo` represents for the extra info of the filter and `\dbIndex` represents for the item index.

6 Example of Flaw Sweeper Template

```
\documentclass{article}
\usepackage{dbshow}
\usepackage{hyperref}

\NewDocumentCommand { \hyperlinktarget } { m m m } {%
  \hyperlink{#1}{#3}\hypertarget{#2}{}
}

\dbNewDatabase{ques}{
  question=tl,
  answer=tl,
  date=date,
  count=int,
```

```
  labels=clist
}
\dbNewReviewPoints{review}{1, 3, 7, 15, 30, 60}

\begin{dbFilters}{ques}
  % which need reviewed today?
  \dbNewConditional{date}{date}{review|2021-12-25}
  \dbNewConditional{easy}{labels}{easy}
  \dbNewConditional{not-easy}{labels}{mid, hard}
  % questions you haven't answered correctly for more than 3 times!
  \dbNewConditional{bad}{count}{\dbval > 3}
  \dbCombineConditionals{date}{date}[to be review today]
  \dbCombineConditionals{easy}{easy}[easy questions]
  \dbCombineConditionals{not-easy}{not-easy}[hard questions]
  \dbCombineConditionals{bad}{bad}[bad questions]
  \dbCombineConditionals{bad-easy}{bad && easy}[bad but easy questions]
\end{dbFilters}

\newcounter{ques}

\dbNewStyle{test}{ques} {
  before-code = {\setcounter{ques}{0}\section{Test}},
  item-code = {
    \refstepcounter{ques}
    \par\noindent\arabic{ques}. \dbuse{labels} \dbuse{date}\hfill \dbuse{count}
    \par\noindent ques: \dbuse{question}
    \par\noindent\hyperlinktarget
      {answer_\dbIndex}{ques_\dbIndex}{go to answer}
  },
}
\dbNewStyle{check}{ques} {
  before-code = {\setcounter{ques}{0}\section{Answer}},
  item-code = {
    \refstepcounter{ques}
    \par\noindent\arabic{ques}. \dbuse{labels} \dbuse{date}\hfill \dbuse{count}
    \par\noindent ques: \dbuse{question}
    \par\noindent\hyperlinktarget
      {ques_\dbIndex}{answer_\dbIndex}{back to question}
    \par\noindent answer: \dbuse{answer}
  },
}
\dbNewStyle[test]{test-bad}{ques}{
  filter=bad,
  before-code = {\setcounter{ques}{0}\section{Bad Question}}
}

\AtEndDocument{
  \dbshow{test}{ques}
  \dbshow{test-bad}{ques}
```

```
  \dbshow{check}{ques}
}

\begin{document}

\begin{dbitem}{ques}[date=2021-12-21, count=1, labels=easy]
  \dbsave{question}{This is test question 1.}
  \dbsave{answer}{This is test question 1.}
\end{dbitem}

\begin{dbitem}{ques}[date=2021-12-22, count=4, labels=easy]
  \dbsave{question}{This is test question 2.}
  \dbsave{answer}{This is test question 2.}
\end{dbitem}

\begin{dbitem}{ques}[date=2021-12-23, count=3, labels=hard]
  \dbsave{question}{This is test question 3.}
  \dbsave{answer}{This is test question 3.}
\end{dbitem}

\end{document}
```

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.