dbshow 宏包 v1.4*　⟹ English Version

李昌锴 <lichangkai225@qq.com>

2022 年 1 月 13 日

Contents

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

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。

dbshow 依赖版本日期至少为 2022-11-07 的 l3kernel。

- 名字后带有 $\star$ 的命令是可以完全展开的（fully-expandable）；
- 名字后带有 ☆ 的命令可以有限制地展开（restricted-expandable）；
- 名字后不带有特殊字符的命令是不可展开的（non-expandable）；
- 名字后带有 $\star$ 的选项不影响相关的代码的是否可展开；
- 名字后带有 ☆ 的选项是否影响相关代码的可展开性取决于选项的设置；
- 名字后不带有特殊字符的选项会使与之相关的代码变得不可展开。

1.1 数据类型

宏包基于 expl3 的基础类型构建了 6 种类型：

`date` 日期类型，以 `yyyy/mm/dd` 形式存储，支持大小比较，排序（转换成字符串）。默认值为 `\dbtoday`。

`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。

除了日期类型，所有类型都是 expl3 的内置类型。dbshow 构建了一个简单的 `date` 类型，支持转换成整数以及带样式的打印。

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
\dbNewDatabase*
New: 2022-01-05
Updated: 2022-01-10

```
\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
}
```

\dbshow
New: 2022-01-05

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

使用 ⟨*style*⟩ 样式来展示 ⟨*database*⟩。

\dbclear
New: 2022-01-07

```
\dbclear {⟨database⟩}
```

清空 ⟨*database*⟩ 里的所有内容。

2.2 \dbNewStyle 和样式选项

\dbNewStyle
New: 2022-01-05

```
\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}{}`。

2.2.1 通用选项

filter

New: 2022-01-05

filter = ⟨*filter*⟩

为当前样式设置由 \dbCombineFilters 所定义的过滤器

raw-filter

New: 2022-01-06

raw-filter = ⟨*conditional expression*⟩

使用条件表达式设置匿名过滤器，这里的条件指通过 \dbNewConditional 定义的条件。下面代码中两个示例的过滤器具有相同的功能。

```
% 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

New: 2022-01-05

sort = { ⟨*attr spec1*⟩, ⟨*attr spec2*⟩, … }

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 }
}
```

item-code ☆

New: 2022-01-05

item-code = ⟨*code*⟩

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 ☆

New: 2022-01-05

Updated: 2022-01-08

⟨*attr*⟩/sep = ⟨*separator*⟩

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

⟨*attr*⟩/sep = {
  ⟨*separator before year*⟩,
  ⟨*separator between year and month*⟩,
  ⟨*separator between month and day*⟩,
  ⟨*separator after day*⟩
}

该选项只适用于类型为 clist 或 date 的属性，用来设置列表间元素的间隔。参数为一个 ⟨*separator*⟩ 时，所有元素间的分隔符被设置为 ⟨*separator*⟩。⟨*separator before year*⟩ 和 ⟨*separator after day*⟩ 被设置为空。

参数为 3 个元素的逗号分隔的列表时，此选项用来设置列表元素的分隔符，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*⟩。对于类型为 clist 的属性，设置此选项时如果参数列表数量不是 1 或者 3 会触发报错。

```
% clist-attr is an attribute of database db
% suppose the val of clist-attr is { 1, 2, 3 }
\dbNewStyle{clist-sep}{db}{
  clist-attr/sep = { ,~ },                % print 1, 2, 3
  clist-attr/sep = { {,~}, {,~}, {and~} } % print 1, 2 and 3
}
```

参数为 4 个元素的逗号分隔的列表时，此选项用来设置日期的分隔符，分别用来设置 ⟨*year*⟩ 之前的分隔符 ⟨*separator before year*⟩，⟨*year*⟩ 和 ⟨*month*⟩ 之间的分隔符 ⟨*separator between year and month*⟩，⟨*month*⟩ 和 ⟨*day*⟩ 之间的分隔符，以及 ⟨*day*⟩ 之后的分隔符。对于类型为 date 的属性，设置此选项时如果参数列表数量不是 1 或者 4 会触发报错。

```
% date-attr is an attribute of database db
% suppose the val of date-attr is 2022/01/01
\dbNewStyle{date-sep}{db}{
  date-attr/sep = -,              % print 2022-01-01
  date-attr/sep = { |, -, -, | } % print |2022-01-01|
}
```

<attr>/zfill ⋆

New: 2022-01-08

⟨*attr*⟩/zfill = ⟨**true**|*false*⟩

该选项只适用于类型为 date 的属性。控制输出月份和天时是否补零。

\dbdatesep ⋆

New: 2022-01-13

\dbdatesep {⟨*separator*⟩}

设置内部解析日期时的间隔符，默认为 /，即存储数据的格式为 yyyy/mm/dd。

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 = { = },
}
```

<attr>/wrapper ☆ ⟨*attr*⟩/wrapper = ⟨*control sequence*⟩

New: 2022-01-08

该选项只适用于类型为 date 的属性。⟨*control sequence*⟩ 只接收一个参数即日期, 如果设置了此选项, 则最后输出的日期为 ⟨*control sequence*⟩{⟨*date*⟩}。

before-code ☆ before-code = ⟨*code*⟩

New: 2022-01-05

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 = ⟨*code*⟩

New: 2022-01-05

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 ☆ record-before-code = ⟨*code*⟩

New: 2022-01-05

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 ☆ record-after-code = ⟨*code*⟩

New: 2022-01-05

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 ☆ ⟨*attr*⟩/before-code = ⟨*code*⟩

New: 2022-01-05

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 ☆ ⟨*attr*⟩/after-code = ⟨*code*⟩

New: 2022-01-05

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 ☆ ⟨*attr*⟩/item-before-code = ⟨*code*⟩

New: 2022-01-05

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 ☆ ⟨*attr*⟩/item-after-code = ⟨*code*⟩

New: 2022-01-05

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*⟩}

New: 2022-01-05

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}

New: 2022-01-05

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*

New: 2022-01-05

Updated: 2022-01-08

```
\dbNewConditional  {⟨name⟩}        {⟨attr⟩} {⟨cond spec⟩}
\dbNewConditional* {⟨name⟩}        {⟨attr⟩} {⟨cond spec⟩}
\dbNewConditional  {⟨name⟩} {⟨int/fp attr⟩} {⟨expr⟩}
\dbNewConditional* {⟨name⟩} {⟨int/fp attr⟩} {⟨expr⟩}
\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⟩} {⟨expr⟩}
\dbNewConditional* {⟨name⟩}   {⟨date attr⟩} {⟨review points⟩|⟨date⟩}
```

\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, ⟨*expr*⟩ 传递给 \int_compare:nTF 或 \fp_compare:nTF 处理。

NOTE: / 为四舍五入除法，截断除法请用 \dbIntDivTruncate。

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 定义的条件后续处理中会将 ⟨*expr*⟩ 中的所有日期转换成相对 *1971* 年 *1* 月 *1* 日的一个整数值，然后将处理后的表达式传递给 \int_compare:nTF 做进一步处理; \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。

\dbCombineConditionals

New: 2022-01-05

```
\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

New: 2022-01-05

Updated: 2022-01-13

```
\begin{dbitem} {⟨database⟩} [⟨attr-val list⟩]
  ⟨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。⟨*attr*⟩ = ⟨*val*⟩ 等同于 \dbsave{⟨*attr*⟩}{⟨*val*⟩}，⟨*attr*⟩* = ⟨*val*⟩ 等同于 \dbsave*{⟨*attr*⟩}{⟨*val*⟩}，数据在 e 或者 x 类型的参数中不可展开。\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}
```

\dbitemkv `\dbitemkv {⟨database⟩} [⟨attr-val list⟩]`

New: 2022-01-13

只使用 ⟨*attr-val list*⟩ 来存储数据。

2.6 \dbsave 和 \dbuse

\dbsave `\dbsave {⟨attr⟩} {⟨data⟩}`

\dbsave* `\dbsave* {⟨attr⟩} {⟨data⟩}`

New: 2022-01-05

Updated: 2022-01-08

\dbsave 用来存储数据, 只能在 item 环境中使用。使用 \dbsave* 存储的数据会被 \exp_not:n 包裹。

\dbuse ⋆ `\dbuse {⟨attr⟩}`

New: 2022-01-05

Updated: 2022-01-08

\dbuse 用来展示数据，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⟩}`

New: 2022-01-05

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 表达式函数

\dbIntAbs	⋆
\dbIntSign	⋆
\dbIntDivRound	⋆
\dbIntDivTruncate	⋆
\dbIntMax	⋆
\dbIntMin	⋆
\dbIntMod	⋆
\dbFpSign	⋆

New: 2022-01-10

\dbIntAbs {⟨*intexpr*⟩}
\dbIntSign {⟨*intexpr*⟩}
\dbIntDivRound {⟨$intexpr_1$⟩} {⟨$intexpr_2$⟩}
\dbIntDivTruncate {⟨$intexpr_1$⟩} {⟨$intexpr_2$⟩}
\dbIntMax {⟨$intexpr_1$⟩} {⟨$intexpr_2$⟩}
\dbIntMin {⟨$intexpr_1$⟩} {⟨$intexpr_2$⟩}
\dbIntMod {⟨$intexpr_1$⟩} {⟨$intexpr_2$⟩}
\dbFpSign {⟨*fpexpr*⟩}

\dbIntAbs 等同于 \int_abs:n
\dbIntSign 等同于 \int_sign:n
\dbIntDivRound 等同于 \int_div_round:nn
\dbIntDivTruncate 等同于 \int_div_truncate:nn
\dbIntMax 等同于 \int_max:nn
\dbIntMin 等同于 \int_min:nn
\dbIntMod 等同于 \int_mod:nn
\dbFpSign 等同于 \fp_sign:n
详细的文档见 interface3

2.9 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	⋆
\dbDatabase	⋆
\dbFilterName	⋆
\dbFilterInfo	⋆
\dbIndex	⋆
\dbarabic	⋆
\dbalph	⋆
\dbAlph	⋆
\dbroman	⋆
\dbRoman	⋆

New: 2022-01-05

\dbval	当前属性的值
\dbDatabase	数据库名称
\dbFilterName	当前样式过滤器的名称
\dbFilterInfo	当前样式过滤器的相关信息
\dbIndex	数据索引，等同于 \dbuseid
\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	用大写字母表示的查询集数据计数
\dbroman	用小写罗马字母表示的查询集数据计数
\dbroman	用大写罗马字母表示的查询集数据计数

3 错题本示例

见第 6 节。

Package dbshow v1.4*

⟹ 中文版本

Changkai Li <lichangkai225@qq.com>

2022/01/13

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.

dbshow depends on l3kernel with version date after 2022-11-07.

- Macros with a ⋆ are fully-expandable;
- Macros with a ☆ are restricted-expandable;
- Macros without appending a special symbol are nonexpandable;
- Options with a ⋆ *do not* affect the expandability of related macros;
- Options with a ☆ affect the expandability of related macros according to its value;
- Options without appending a special symbol make the related macros nonexpandable.

4.1 Data Types

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

`date` date saved in `yyyy/mm/dd` format, supports comparison, sorting (converting to string), default `\dbtoday`.

`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.

All types are internal types of expl3 except `date` type, which provides by dbshow itself and supports converting to integer and printing with style.

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

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.

5 Interfaces

5.1 Create, Display and Clear Database

\dbNewDatabase
\dbNewDatabase*

New: 2022-01-05
Updated: 2022-01-10

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

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
}
```

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

New: 2022-01-05

Show the ⟨*database*⟩ with ⟨*style*⟩.

\dbclear `\dbclear {⟨database⟩}`

New: 2022-01-07

Clear the content of ⟨*database*⟩.

5.2 \dbNewStyle and Style Options

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

New: 2022-01-05

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 `filter = ⟨filter⟩`

New: 2022-01-05

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

raw-filter `raw-filter = ⟨conditional expression⟩`

New: 2022-01-06

Set anonymous with conditionals defined by `\dbNewConditional`. Two filters shows in the code below have the same meaning.

```
% 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 `sort = { ⟨attr spec1⟩, ⟨attr spec2⟩, … }`

New: 2022-01-05

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 }
}
```

item-code ☆

New: 2022-01-05

`item-code` = ⟨*code*⟩

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

<attr>/sep ☆

New: 2022-01-05

Updated: 2022-01-08

⟨*attr*⟩`/sep` = ⟨*separator*⟩

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

⟨*attr*⟩`/sep = {`
  ⟨*separator before year*⟩`,`
  ⟨*separator between year and month*⟩`,`
  ⟨*separator between month and day*⟩`,`
  ⟨*separator after day*⟩
`}`

Only for attributes of type `clist` or `date`. Set the separator between items. If the argument is an one-item comma list, all separators are set to ⟨*separator*⟩ but ⟨*separator before year*⟩ and ⟨*separator after day*⟩ is set empty.

If the argument is a comma list of 3 items, it is used to set the separator between items of the comma list. 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.

For attributes of type `clist`, incorrect number (numbers exclude 1 and 3) of items of the argument will raise an error.

```
% clist-attr is an attribute of database db
% suppose the val of clist-attr is { 1, 2, 3 }
\dbNewStyle{clist-sep}{db}{
  clist-attr/sep = { ,~ },                % print 1, 2, 3
  clist-attr/sep = { {,~}, {,~}, {and~} } % print 1, 2 and 3
}
```

If the argument is a comma list of 4 items, it is used to set the separators of the date. For attributes of type `date`, incorrect number (numbers exclude 1 and 4) will raise an error.

```
% date-attr is an attribute of database db
% suppose the val of date-attr is 2022/01/01
\dbNewStyle{date-sep}{db}{
  date-attr/sep = -,             % print 2022-01-01
  date-attr/sep = { |, -, -, | } % print |2022-01-01|
}
```

<attr>/zfill ⋆ ⟨*attr*⟩/zfill = ⟨true|*false*⟩

New: 2022-01-08

Only for attributes of type `date`. Control whether to fill zero on the left of the month or day.

\dbdatesep ⋆ \dbdatesep {⟨*separator*⟩}

New: 2022-01-13

Set the separator for internal date parsing. The default value is `/`, i.e. the date must be store in the format of `yyyy/mm/dd`.

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, ⟨*style1*⟩ and ⟨*style2*⟩ is the same style, which wrap ⟨*attr1*⟩ with *. When you want another style which wrap ⟨*attr1*⟩ with =, if you choose the way of ⟨*style1*⟩, ⟨*item code*⟩ are repeated, otherwise if you choose the way of ⟨*style2*⟩, ⟨*item code*⟩ 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 = { = },
}
```

<attr>/wrapper ☆ ⟨*attr*⟩/wrapper = ⟨*control sequence*⟩

New: 2022-01-08

Only for attributes of type `date`. Output of `\dbuse{`⟨*date attr*⟩`}` will be ⟨*control sequence*⟩`{`⟨*date*⟩`}`.

before-code ☆ before-code = ⟨*code*⟩

New: 2022-01-05

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

after-code ☆ after-code = ⟨*code*⟩

New: 2022-01-05

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

record-before-code ☆ record-before-code = ⟨*code*⟩

New: 2022-01-05

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

record-after-code ☆ record-after-code = ⟨*code*⟩

New: 2022-01-05

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

<attr>/before-code ☆ ⟨*attr*⟩/before-code = ⟨*code*⟩

New: 2022-01-05

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

<attr>/after-code ☆ ⟨*attr*⟩/after-code = ⟨*code*⟩

New: 2022-01-05

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

<attr>/item-before-code ☆ ⟨*attr*⟩/item-before-code = ⟨*code*⟩

New: 2022-01-05

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

<attr>/item-after-code ☆ ⟨*attr*⟩/item-after-code = ⟨*code*⟩

New: 2022-01-05

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

5.3 Use `\dbNewReviewPoints` to Define Review Points

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

New: 2022-01-05

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

New: 2022-01-05

```
\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*

New: 2022-01-05
Updated: 2022-01-08

```
\dbNewConditional  {⟨name⟩}        {⟨attr⟩} {⟨cond spec⟩}
\dbNewConditional* {⟨name⟩}        {⟨attr⟩} {⟨cond spec⟩}
\dbNewConditional  {⟨name⟩} {⟨int/fp attr⟩} {⟨expr⟩}
\dbNewConditional* {⟨name⟩} {⟨int/fp attr⟩} {⟨expr⟩}
\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⟩} {⟨expr⟩}
\dbNewConditional* {⟨name⟩}   {⟨date attr⟩} {⟨review points⟩|⟨date⟩}
```

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`, ⟨*expr*⟩ is passed to `\int_compare:nTF` or `\fp_compare:nTF`.

NOTE: Division using `/` rounds to the closest integer. Use `\dbIntDivTruncate` to rounds the result toward 0.

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 replace each date with a integer representing for the days between ⟨*date*⟩ and *1971/01/01*, and the result is passed to `\int_compare:nTF`. Starred form defines the conditional with review points defined by `\dbNewRdbNewReviewPoints` and ⟨*date*⟩ is the date to be compared.

\dbCombineConditionals

New: 2022-01-05

`\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

New: 2022-01-05

Updated: 2022-01-13

```
\begin{dbitem} {⟨database⟩} [⟨attr-val list⟩]
  ⟨code⟩
\end{dbitem}
```

The data are stored with `dbitem` environment in two ways. Short data can be stored in ⟨*attr-val list*⟩ and long data can be stored by `\dbsave`, which will suppress the value set by the option list. ⟨*attr*⟩ = ⟨*val*⟩ is equal to `\dbsave{`⟨*attr*⟩`}{`⟨*val*⟩`}`, and ⟨*attr*⟩ = ⟨*val*⟩ is equal to `\dbsave*{`⟨*attr*⟩`}{`⟨*val*⟩`}`, in which case, data will not be expanded in an `e` or `x`-type argument. 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 a test answer.}
\end{dbitem}
```

\dbitemkv

New: 2022-01-13

`\dbitemkv {⟨database⟩} [⟨attr-val list⟩]`

Store data with ⟨*attr-val list*⟩.

5.6 `\dbsave` and `\dbuse`

\dbsave
\dbsave*

New: 2022-01-05

Updated: 2022-01-08

```
\dbsave  {⟨attr⟩} {⟨data⟩}
\dbsave* {⟨attr⟩} {⟨data⟩}
```

`\dbsave` save the ⟨*data*⟩ to ⟨*attr*⟩ of current record. `\dbsave` can be used only inside the `dbitem` environment. ⟨*data*⟩ stored by `\dbsave*` is wrapped with `\exp_not:n` while ⟨*data*⟩ stored by `\dbsave` keeps the same.

\dbuse ⋆

New: 2022-01-05

Updated: 2022-01-08

`\dbuse {⟨attr⟩}`

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

5.7 Conditionals

\dbIfEmptyT ⋆
\dbIfEmptyF ⋆
\dbIfEmptyTF ⋆

New: 2022-01-05

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

5.8 Expression Functions

\dbIntAbs ⋆	\dbIntAbs {⟨*intexpr*⟩}
\dbIntSign ⋆	\dbIntSign {⟨*intexpr*⟩}
\dbIntDivRound ⋆	\dbIntDivRound {⟨*intexpr*$_1$⟩} {⟨*intexpr*$_2$⟩}
\dbIntDivTruncate ⋆	\dbIntDivTruncate {⟨*intexpr*$_1$⟩} {⟨*intexpr*$_2$⟩}
\dbIntMax ⋆	\dbIntMax {⟨*intexpr*$_1$⟩} {⟨*intexpr*$_2$⟩}
\dbIntMin ⋆	\dbIntMin {⟨*intexpr*$_1$⟩} {⟨*intexpr*$_2$⟩}
\dbIntMod ⋆	\dbIntMod {⟨*intexpr*$_1$⟩} {⟨*intexpr*$_2$⟩}
\dbFpSign ⋆	\dbFpSign {⟨*fpexpr*⟩}

New: 2022-01-10

\dbIntAbs is identical to \int_abs:n

\dbIntSign is identical to \int_sign:n

\dbIntDivRound is identical to \int_div_round:nn

\dbIntDivTruncate is identical to \int_div_truncate:nn

\dbIntMax is identical to \int_max:nn

\dbIntMin is identical to \int_min:nn

\dbIntMod is identical to \int_mod:nn

\dbFpSign is identical to \fp_sign:n

Detailed documentation see interface3

5.9 Special Macros

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

\dbval ⋆	\dbval
\dbDatabase ⋆	\dbDatabase
\dbFilterName ⋆	\dbFilterName
\dbFilterInfo ⋆	\dbFilterInfo
\dbIndex ⋆	\dbIndex
\dbarabic ⋆	\dbarabic
\dbalph ⋆	\dbalph
\dbAlph ⋆	\dbAlph
\dbroman ⋆	\dbroman
\dbRoman ⋆	\dbroman

New: 2022-01-05

\dbval	Attribute value, only according in \dbNewConditional.
\dbDatabase	Database name.
\dbFilterName	Filter name.
\dbFilterInfo	Filter information.
\dbIndex	Record index, identical to \dbuseid
\dbarabic	Show the counter of query set as digits.
\dbalph	Show the counter of query set as lowercase letters.
\dbAlph	Show the counter of query set as uppercase letters.
\dbroman	Show the counter of query set as lowercase roman numerals.
\dbroman	Show the counter of query set as uppercase 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}
  \dbNewConditional{after}{date}{\dbval > 2022/01/02}
\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
\dbNewStyle{hard}{ques-book}{
```

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

% show all hard questions that have been answered incorrectly for more than
% one time with hyperlink to answers
\dbNewStyle[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
\dbNewStyle[hard]{review}{ques-book}{
  before-code = {\section{Questions to be Reviewed}},
  raw-filter = {review},
}
% show all questions that is record after 2022/01/02 with hyperlink to answers
\dbNewStyle[hard]{after}{ques-book}{
  before-code = {\section{Questions after 2022/01/02}},
  raw-filter = {after},
}

\AtEndDocument{
  \dbshow{review}{ques-book}
  \dbshow{hard}{ques-book}
  \dbshow{bad}{ques-book}
  \dbshow{after}{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: $\int 2x \dd x$.
  }
  \dbsave{answer}{%
    $\int 2x \dd x = x^2$
  }
\end{dbitem}

\end{document}
```

7 Implementation

```
⟨*package⟩
⟨@@=dbshow⟩
```

Check version of l3kernel.

```
% prop_concat, prop_gset_from_keyval
\__kernel_dependency_version_check:nn { 2021-11-07 } { l3prop }
% str_compare
\__kernel_dependency_version_check:nn { 2021-05-17 } { l3str }
% clist_map_tokens, clist_use
\__kernel_dependency_version_check:nn { 2021-05-10 } { l3clist }
```

7.1 Variants and Variables

```
\cs_generate_variant:Nn \msg_warning:nnnn      { nnnx }
\cs_generate_variant:Nn \keys_set:nn           { nv }
\cs_generate_variant:Nn \clist_use:nn          { xx }
\cs_generate_variant:Nn \clist_use:nnnn        { xxxx }
\cs_generate_variant:Nn \clist_map_inline:nn   { Vn }
\cs_generate_variant:Nn \prop_get:NnN          { cVN }
\cs_generate_variant:Nn \regex_extract_all:nnN { nVN }
\cs_generate_variant:Nn \regex_split:nnN       { nVN }
\prg_generate_conditional_variant:Nnn \str_compare:nNn { VNV } { TF }
\prg_generate_conditional_variant:Nnn \int_compare:nNn { VNV } { TF }
\prg_generate_conditional_variant:Nnn \fp_compare:nNn  { VNV } { TF }
\prg_generate_conditional_variant:Nnn \regex_match:nn  { VV }  { TF }
\prg_generate_conditional_variant:Nnn \clist_if_in:Nn  { Nx }  { TF }
```

\c__dbshow_default_value_prop Default default value of different types.

```
\prop_const_from_keyval:Nn \c__dbshow_default_value_prop {
  date  = \dbtoday,
  str   = ,
  tl    = ,
  clist = ,
  int   = 0,
  fp    = 0
}
```

(*End definition for* `\c__dbshow_default_value_prop`.)

__dbshow_type_clist Supported types by dbshow.

```
\clist_const:Nn \__dbshow_type_clist { date, str, tl, clist, int, fp }
```

(*End definition for* `\__dbshow_type_clist`.)

\g__dbshow_raw_filter_int Counter of anonymous filter.

```
\int_gzero_new:N \g__dbshow_raw_filter_int
```

(*End definition for* `\g__dbshow_raw_filter_int`.)

7.2 Messages

#1 : ⟨*database*⟩

```
\msg_new:nnn { dbshow } { non-existent-database } {
  Database~'#1'~does~not~exist~\msg_line_context:.
}
```

__dbshow_check_database:n Check if the database is valid.

#1 : ⟨*database*⟩

```
\cs_new:Nn \__dbshow_check_database:n {
  \prop_if_exist:cF { g__dbshow_attr_type_prop_#1 }
    { \msg_fatal:nnn { dbshow } { non-existent-database } {#1} }
}
```

(*End definition for* __dbshow_check_database:n.)

#1 : ⟨*database*⟩
#2 : ⟨*attr*⟩

```
\msg_new:nnn { dbshow } { non-existent-attr } {
  Attribute~'#2'~of~database~'#1'~does~not~exist~\msg_line_context:.
}
```

__dbshow_check_attr:nn Check if the attribute is valid.
__dbshow_check_attr:nV

#1 : ⟨*database*⟩
#2 : ⟨*attr*⟩

```
\cs_new:Nn \__dbshow_check_attr:nn {
  \prop_if_in:cnF { g__dbshow_attr_type_prop_#1 } {#2}
    { \msg_fatal:nnnn { dbshow } { non-existent-attr } {#1} {#2} }
}
\cs_generate_variant:Nn \__dbshow_check_attr:nn { nV }
```

(*End definition for* __dbshow_check_attr:nn.)

#1 : ⟨*style*⟩
#2 : ⟨*database*⟩

```
\msg_new:nnn { dbshow } { non-existent-style } {
  Style~'#1'~of~database~'#2'~does~not~exist~\msg_line_context:.
}
```

__dbshow_check_style:nn Check if the style is valid.

#1 : ⟨*style*⟩
#2 : ⟨*database*⟩

```
\cs_new:Nn \__dbshow_check_style:nn {
  \tl_if_exist:cF { g__dbshow_style_opts_tl_#1_#2 }
    { \msg_warning:nnnn { dbshow } { non-existent-style } {#1} {#2} }
}
```

(*End definition for* __dbshow_check_style:nn.)

#1 : ⟨*database*⟩
#2 : ⟨*cond*⟩

```
\msg_new:nnn { dbshow } { non-existent-cond } {
  Conditional~'#2'~of~database~'#1'~does~not~exist~\msg_line_context:.
}
```

`\__dbshow_check_cond:nnn` Check if the conditional is valid.

#1 : ⟨*database*⟩
#2 : ⟨*cond*⟩

```
\cs_new:Nn \__dbshow_check_cond:nnn {
  \tl_if_exist:cF { g__dbshow_filter_attr_#1_#2 }
    { \msg_fatal:nnnn { dbshow } { non-existent-cond } {#1} {#2} }
}
```

(*End definition for* `\__dbshow_check_cond:nnn`.)

#1 : ⟨*database*⟩
#2 : ⟨*filter*⟩

```
\msg_new:nnn { dbshow } { non-existent-filter } {
  Filter~'#2'~of~database~'#1'~does~not~exist~and~is~ignored~\msg_line_context:.
}
```

`\__dbshow_check_filter:nn` Check if the filter is valid. Ignore the default filter `-none-`.
`\__dbshow_check_filter:nv`

#1 : ⟨*database*⟩
#2 : ⟨*filter*⟩

```
\cs_new:Nn \__dbshow_check_filter:nn {
  \seq_if_exist:cF { g__dbshow_filter_run_seq_#1_#2 } {
    \str_if_eq:eeF {#2} { -none- } {
      \msg_warning:nnnx { dbshow } { non-existent-filter } {#1} {#2}
    }
  }
}
\cs_generate_variant:Nn \__dbshow_check_filter:nn { nv }
```

(*End definition for* `\__dbshow_check_filter:nn`.)

#1 : ⟨*type*⟩

```
\msg_new:nnn { dbshow } { non-existent-type } {
  Type~'#1'~does~not~exist,~the~type~of~attribute~should~be~one~of~
  \{date,~str,~tl,~clist,~int,~fp\}~\msg_line_context:.
}
```

`\__dbshow_check_type:n` Check if the type is valid.

#1 : ⟨*type*⟩

```
\cs_new:Nn \__dbshow_check_type:n {
  \clist_if_in:NnF \__dbshow_type_clist {#1}
    { \msg_fatal:nnn { dbshow } { non-existent-type } {#1} }
}
```

(*End definition for* `\__dbshow_check_type:n`.)

`#1` : ⟨*valid count*⟩
`#2` : ⟨*real count*⟩
`#3` : ⟨*value*⟩

```
\msg_new:nnn { dbshow } { wrong-seperator } {
  option~'sep'~should~contain~#1~items~but~only~#2~items~was~given,~
  sep~=~\{#3\}~\msg_line_context:.
}
```

`\__dbshow_sep_error:nnn` Check the value of `<attr>/sep` is valid.
`\__dbshow_sep_error:xxx`

`#1` : ⟨*valid count*⟩
`#2` : ⟨*real count*⟩
`#3` : ⟨*value*⟩

```
\cs_new:Nn \__dbshow_sep_error:nnn {
  \msg_error:nnnnn { dbshow } { wrong-seperator } {#1} {#2} {#3}
}
\cs_generate_variant:Nn \__dbshow_sep_error:nnn { xxx }
```

(*End definition for* `\__dbshow_sep_error:nnn`.)

`#1` : ⟨*type*⟩

```
\msg_new:nnn { dbshow } { unsupported-sort-type } {
  unsupported~sort~type:~'#1'~\msg_line_context:.~The~type~should~be~one~of~
  \{str,~date,~int,~fp\}.
}
```

7.3 Create Database

`\__dbshow_process_default_value:w` Create map from ⟨*attr*⟩ to ⟨*type*⟩ and map from ⟨*attr*⟩ to ⟨*default value*⟩.

`#1` : ⟨*database*⟩
`#2` : ⟨*attr*⟩
`#3` : ⟨*type*⟩
`#4` : ⟨*default value*⟩

```
\cs_new:Npn \__dbshow_process_default_value:w
#1\__dbshow_sep#2\__dbshow_sep#3|#4\__dbshow_stop {
  \__dbshow_check_type:n {#3}
  \prop_gput:cxx { g__dbshow_attr_type_prop_#1 } {#2} {#3}
  \prop_gput:cxx { g__dbshow_default_map_#1 } {#2} {#4}
}
```

(*End definition for* `\__dbshow_process_default_value:w`.)

`\__dbshow_process_attr_type_prop:n` Parse default value from the value of the type map. If ⟨*type spec*⟩ is ⟨*type*|*default value*⟩ then set the default value to ⟨*default value*⟩, otherwise if ⟨*type spec*⟩ is ⟨*type*⟩ then set the default value to the default value of the type.

`#1` : ⟨*database*⟩

```
\cs_new_protected:Nn \__dbshow_process_attr_type_prop:n {
  \prop_gclear_new:c { g__dbshow_default_map_#1 }
```

##1 : ⟨*attr*⟩
##2 : ⟨*type spec*⟩

```
  \prop_map_inline:cn { g__dbshow_attr_type_prop_#1 } {
    \str_if_in:nnTF {##2} { | } {
      \__dbshow_process_default_value:w
        #1\__dbshow_sep##1\__dbshow_sep##2\__dbshow_stop
    } {
      \prop_get:NnN \c__dbshow_default_value_prop {##2} \l__dbshow_tmp_default
      \__dbshow_process_default_value:w
        #1\__dbshow_sep##1\__dbshow_sep##2|\l__dbshow_tmp_default\__dbshow_stop
    }
  }
}
```

(*End definition for* `\__dbshow_process_attr_type_prop:n`.)

`\__dbshow_database_new:nn` Create a new ⟨*database*⟩ with ⟨*database spec*⟩, which is a list of ⟨*attr*⟩ = ⟨*type spec*⟩. This function initialize the index counter and save ⟨*attr*⟩ = ⟨*type spec*⟩ pairs to the type map. If ⟨*database*⟩ has existed, the old definition is discarded.

#1 : ⟨*database*⟩
#2 : ⟨*database spec*⟩

```
\cs_new_protected:Nn \__dbshow_database_new:nn {
  \int_gzero_new:c { g__dbshow_counter_#1 }
  \prop_gset_from_keyval:cn { g__dbshow_attr_type_prop_#1 } {#2}
}
```

(*End definition for* `\__dbshow_database_new:nn`.)

`\__dbshow_database_new_append:nn` Create a new ⟨*database*⟩ with ⟨*database spec*⟩, which is a list of ⟨*attr*⟩ = ⟨*type spec*⟩. This function initialize the index counter and save ⟨*attr*⟩ = ⟨*type spec*⟩ pairs to the type map. If ⟨*database*⟩ has existed, the old definition is merged into the new definition that has a higher priority.

#1 : ⟨*database*⟩
#2 : ⟨*database spec*⟩

```
\cs_new_protected:Nn \__dbshow_database_new_append:nn {
  \int_gzero_new:c { g__dbshow_counter_#1 }
  \prop_if_exist:cF { g__dbshow_attr_type_prop_#1 }
    { \prop_new:c { g__dbshow_attr_type_prop_#1 } }
  \prop_gset_from_keyval:Nn \l_tmpa_prop {#2}
  \prop_concat:ccc { g__dbshow_attr_type_prop_#1 }
    { g__dbshow_attr_type_prop_#1 } { l_tmpa_prop }
}
```

(*End definition for* `\__dbshow_database_new_append:nn`.)

__dbshow_database_new_inherit:nnn

Create a new ⟨*database*⟩ with ⟨*database spec*⟩, which is a list of ⟨*attr*⟩ = ⟨*type spec*⟩. The new ⟨*database*⟩ is based on ⟨*base database*⟩. If ⟨*database*⟩ is equal to ⟨*base database*⟩, `\__dbshow_database_new_append:nn` is called, otherwise the index counter is initialized and the definition is merged with the definition of ⟨*base dadatabase*⟩.

`#1` : ⟨*database*⟩
`#2` : ⟨*base database*⟩
`#3` : ⟨*database spec*⟩

```
\cs_new_protected:Nn \__dbshow_database_new_inherit:nnn {
  \__dbshow_check_database:n {#2}
  \str_if_eq:nnTF {#1} {#2} {
    \__dbshow_database_new_append:nn {#1} {#3}
  } {
    \int_gzero_new:c { g__dbshow_counter_#1 }
    \prop_gset_from_keyval:cn { g__dbshow_attr_type_prop_#1 } {#3}
    \prop_concat:ccc { g__dbshow_attr_type_prop_#1 }
      { g__dbshow_attr_type_prop_#2 } { g__dbshow_attr_type_prop_#1 }
  }
}
```

(*End definition for* `\__dbshow_database_new_inherit:nnn`.)

\dbNewDatabase

User interface to create a new ⟨*database*⟩. Internal attribute `id` is added to ⟨*database*⟩. After attributes are settle down, default values are parsed by `\__dbshow_process_attr_type_prop:n`, and keys serving to save data of ⟨*database*⟩ are defined. At last, we define the `default` style as its name implies.

`#1` : ⟨*star*⟩ that indicates append or not
`#2` : ⟨*base database*⟩
`#3` : ⟨*database*⟩
`#4` : ⟨*database spec*⟩

```
\NewDocumentCommand { \dbNewDatabase } { s o m m } {
  \IfNoValueTF {#2} {
    \IfBooleanTF {#1}
      { \__dbshow_database_new_append:nn {#3} {#4} }
      { \__dbshow_database_new:nn {#3} {#4} }
  } { \__dbshow_database_new_inherit:nnn {#3} {#2} {#4} }
  \__dbshow_database_new_append:nn {#3} { id=int }
  \__dbshow_process_attr_type_prop:n {#3}
  \__dbshow_set_database_keys:n {#3}
  \dbNewStyle{default}{#3}{}
}
```

(*End definition for* `\dbNewDatabase`. *This function is documented on page* 13.)

__dbshow_set_database_keys:n

Set keys of ⟨*database*⟩ to make it able to save data with key-value pairs.

`#1` : ⟨*database*⟩

```
\cs_new_protected:Nn \__dbshow_set_database_keys:n {
```

`##1` : ⟨*type*⟩

```
    \prop_map_inline:cn { g__dbshow_attr_type_prop_#1 } {
      \keys_define:nn { dbshow/database/#1 } {
```

`####1` : ⟨*data*⟩

```
        ##1 .code:n = \__dbshow_save_data:nnn {#1} {##1} {####1},
        ##1* .code:n = {
          \__dbshow_save_data:nnn {#1} {##1} { \exp_not:n {####1} },
        },
      }
    }
}
```

(*End definition for* `\__dbshow_set_database_keys:n`.)

`\__dbshow_get_type:nn` `\__dbshow_get_type:nV`

Convenient function to get the ⟨*type*⟩ of ⟨*attr*⟩ of ⟨*database*⟩.

`#1` : ⟨*database*⟩
`#2` : ⟨*attr*⟩

```
\cs_new:Nn \__dbshow_get_type:nn {
  \prop_item:cn { g__dbshow_attr_type_prop_#1 } {#2}
}
\cs_generate_variant:Nn \__dbshow_get_type:nn { nV }
```

(*End definition for* `\__dbshow_get_type:nn`.)

`\__dbshow_get_counter:n` Convenient function to get the value of the index counter.

`#1` : ⟨*database*⟩

```
\cs_new:Nn \__dbshow_get_counter:n {
  \int_use:c { g__dbshow_counter_#1 }
}
```

(*End definition for* `\__dbshow_get_counter:n`.)

`\__dbshow_step_counter:n` Convenient function to step the index counter.

`#1` : ⟨*database*⟩

```
\cs_new:Nn \__dbshow_step_counter:n {
  \int_gincr:c { g__dbshow_counter_#1 }
}
```

(*End definition for* `\__dbshow_step_counter:n`.)

`\dbclear` User interface to clear the ⟨*database*⟩. The index counter is set to zero and data is not really wiped.

`#1` : ⟨*database*⟩

```
\NewDocumentCommand { \dbclear } { m } {
  \int_gzero:c { g__dbshow_counter_#1 }
}
```

(*End definition for* `\dbclear`. *This function is documented on page* 14.)

7.4 Store Data

__dbshow_save_data:nnn
__dbshow_save_data:nnx

Internal function to store data into \g_@@_data_⟨*database*⟩_⟨*attr*⟩_⟨*index*⟩.

#1 : ⟨*database*⟩
#2 : ⟨*attr*⟩
#3 : ⟨*data*⟩

```
\cs_new:Nn \__dbshow_save_data:nnn {
  \__dbshow_check_attr:nn {#1} {#2}
  \str_case_e:nn { \__dbshow_get_type:nn {#1} {#2} } {
    { str }   { \str_clear_new:c }
    { tl }    { \tl_gclear_new:c }
    { clist } { \clist_gclear_new:c }
    { int }   { \int_gzero_new:c }
    { fp }    { \fp_gzero_new:c }
    { date }  { \__dbdate_gclear_new:x }
  } { g__dbshow_data_#1_#2_\__dbshow_get_counter:n {#1} }
  \str_case_e:nn { \__dbshow_get_type:nn {#1} {#2} } {
    { str }   { \str_gset:cn }
    { tl }    { \tl_gset:cn }
    { clist } { \clist_gset:cn }
    { int }   { \int_gset:cn }
    { fp }    { \fp_gset:cn }
    { date }  { \__dbdate_gset:xx }
  } { g__dbshow_data_#1_#2_\__dbshow_get_counter:n {#1} } {#3}
}
\cs_generate_variant:Nn \__dbshow_save_data:nnn { nnx }
```

(*End definition for* __dbshow_save_data:nnn.)

dbitem
__dbshow_set_default:nn
\dbsave
\dbsave*

Environment dbitem is the user interface to save data with ⟨*attr*⟩ = ⟨*data*⟩ pairs or using \dbsave. First we step the index counter and set the default value for each attribute. Then we set the value by ⟨*attr-value list*⟩ and the index is also stored to default attribute id.

#1 : ⟨*database*⟩
#2 : ⟨*attr-value list*⟩

```
\NewDocumentEnvironment { dbitem } { m +O{} } {
  \__dbshow_check_database:n {#1}
  \__dbshow_step_counter:n {#1}
```

This function is used to set the default values of each attribute.

#1 : ⟨*attr*⟩

```
  \cs_set:Nn \__dbshow_set_default:nn {
    \__dbshow_save_data:nnx {#1} {##1} {
      \prop_item:cn { g__dbshow_default_map_#1 } {##1}
    }
  }
  \prop_map_function:cN { g__dbshow_attr_type_prop_#1 } \__dbshow_set_default:nn
  \__dbshow_save_data:nnx {#1} { id } { \__dbshow_get_counter:n {#1} }
  \keys_set:nn { dbshow/database/#1 } {#2}
```

#1 : ⟨*star*⟩ that indicates whether to use `\exp_not:n`
#2 : ⟨*attr*⟩
#3 : ⟨*data*⟩

```
\NewDocumentCommand { \dbsave } { s m +m } {
  \IfBooleanTF {##1} {
    \__dbshow_save_data:nnn {#1} {##2} { \exp_not:n {##3} }
  } {
    \__dbshow_save_data:nnn {#1} {##2} {##3}
  }
}
} {}
```

(*End definition for* `dbitem` *and others. These functions are documented on page 19.*)

`\dbitemkv` Store data with ⟨*attr-value list*⟩

#1 : ⟨*database*⟩
#2 : ⟨*attr-value list*⟩

```
\NewDocumentCommand { \dbitemkv } { m +m } {
  \begin{dbitem}{#1}[#2]
  \end{dbitem}
}
```

(*End definition for* `\dbitemkv`*. This function is documented on page 19.*)

7.5 Filter

Following functions have the same argument specifications:

#1 : ⟨*star boolean*⟩
#2 : ⟨*expr*⟩ specified by `\dbNewConditional`
#3 : ⟨*current value*⟩, i.e. `\dbval`
#4 : ⟨*true code*⟩ to set filter boolean to true
#5 : ⟨*false code*⟩ to set filter boolean to false

These functions aim to filter ⟨*attr*⟩ of certain type with ⟨*expr*⟩ in every epoch of iteration. The basic idea is to save the filter result into a boolean variable corresponding to each conditional.

`\__dbshow_filter_int:NNNnn` Filter integers.
`\__dbshow_filter_int:cccnn`

#1 : ⟨*star boolean*⟩
#2 : ⟨*expr*⟩ specified by `\dbNewConditional`
#3 : ⟨*current value*⟩, i.e. `\dbval`
#4 : ⟨*true code*⟩ to set filter boolean to true
#5 : ⟨*false code*⟩ to set filter boolean to false

```
\cs_new:Nn \__dbshow_filter_int:NNNnn {
  \int_compare:nTF {#2} {#4} {#5}
}
\cs_generate_variant:Nn \__dbshow_filter_int:NNNnn { cccnn }
```

(*End definition for* `\__dbshow_filter_int:NNNnn`.)

`\__dbshow_filter_fp:NNNnn`
`\__dbshow_filter_fp:cccnn`

Filter floating point values.

`#1` : ⟨*star boolean*⟩
`#2` : ⟨*expr*⟩ specified by `\dbNewConditional`
`#3` : ⟨*current value*⟩, i.e. `\dbval`
`#4` : ⟨*true code*⟩ to set filter boolean to true
`#5` : ⟨*false code*⟩ to set filter boolean to false

```
\cs_new:Nn \__dbshow_filter_fp:NNNnn {
  \int_compare:nTF {#2} {#4} {#5}
}
\cs_generate_variant:Nn \__dbshow_filter_fp:NNNnn { cccnn }
```

(*End definition for* `\__dbshow_filter_fp:NNNnn`.)

`\__dbshow_filter_clist:NNNnn`
`\__dbshow_filter_clist:cccnn`

Filter comma lists. If ⟨*star bool*⟩ is true, all values specified by ⟨*conditional*⟩ should be contained in current list. Otherwise, condition is met if any value specified by ⟨*conditional*⟩ appears in current list.

`#1` : ⟨*star boolean*⟩
`#2` : ⟨*expr*⟩ specified by `\dbNewConditional`
`#3` : ⟨*current value*⟩, i.e. `\dbval`
`#4` : ⟨*true code*⟩ to set filter boolean to true
`#5` : ⟨*false code*⟩ to set filter boolean to false

```
\cs_new_protected:Nn \__dbshow_filter_clist:NNNnn {
```

Initial filter boolean of starred conditional is true. And then we check every value in ⟨*conditional*⟩. If there is some value that does not appear in current list, the result is set to false and the loop is broken.

```
  \bool_if:NTF #1 {
    #4 \clist_map_inline:Vn #2
    { \clist_if_in:NnF #3 {##1} { #5 \clist_map_break: } }
  } {
```

Initial filter boolean of unstarred conditional is false. And then we check every value in ⟨*conditional*⟩. If there is some value that does appear in current list, the result is set to true and the loop is broken.

```
    #5 \clist_map_inline:Vn #2
    { \clist_if_in:NnT #3 {##1} { #4 \clist_map_break: } }
  }
}
\cs_generate_variant:Nn \__dbshow_filter_clist:NNNnn { cccnn }
```

(*End definition for* `\__dbshow_filter_clist:NNNnn`.)

`\__dbshow_filter_str:NNNnn`
`\__dbshow_filter_str:cccnn`

Filter strings with regex expression. Starred filter matches the whole string while unstarred matches part of the string.

`#1` : ⟨*star boolean*⟩
`#2` : ⟨*expr*⟩ specified by `\dbNewConditional`

#3 : ⟨*current value*⟩, i.e. `\dbval`
#4 : ⟨*true code*⟩ to set filter boolean to true
#5 : ⟨*false code*⟩ to set filter boolean to false

```
\cs_new_protected:Nn \__dbshow_filter_str:NNNnn {
  \bool_if:NT #1 {
    \tl_put_left:Nn #2 { \A }
    \tl_put_right:Nn #2 { \Z }
  }
  \regex_match:VVTF #2 #3 {#4} {#5}
}
\cs_generate_variant:Nn \__dbshow_filter_str:NNNnn { cccnn }
```

(*End definition for* `\__dbshow_filter_str:NNNnn`.)

`\__dbshow_filter_tl:NNNnn` `\__dbshow_filter_tl:cccnn` Filter token list with regex expression. It is the same with string.

```
\cs_set_eq:NN \__dbshow_filter_tl:NNNnn \__dbshow_filter_str:NNNnn
\cs_generate_variant:Nn \__dbshow_filter_tl:NNNnn { cccnn }
```

(*End definition for* `\__dbshow_filter_tl:NNNnn`.)

`\__dbshow_parse_date_cond:NNw` Help function to parse ⟨*review points*⟩ and ⟨*date*⟩ and store them in ⟨*clist var*⟩ and ⟨*tl var*⟩.

#1 : ⟨*clist var*⟩ to store ⟨*review points*⟩
#2 : ⟨*tl var*⟩ to store ⟨*date*⟩

```
\cs_new_protected:Npn \__dbshow_parse_date_cond:NNw #1#2#3|#4\__dbshow_stop {
  \clist_set_eq:Nc #1 { g__review_points_#3 }
  \tl_set:Nn #2 {#4}
}
```

(*End definition for* `\__dbshow_parse_date_cond:NNw`.)

`\__dbshow_filter_date:NNNnn` `\__dbshow_filter_date:cccnn` Filter dates by ⟨*review points*⟩ or ⟨*expr*⟩.

#1 : ⟨*star boolean*⟩
#2 : ⟨*expr*⟩ specified by `\dbNewConditional`
#3 : ⟨*current value*⟩, i.e. `\dbval`
#4 : ⟨*true code*⟩ to set filter boolean to true
#5 : ⟨*false code*⟩ to set filter boolean to false

```
\cs_new_protected:Nn \__dbshow_filter_date:NNNnn {
```

For starred ⟨*conditional*⟩, calculate the days between current day, i.e. `\dbval` and the date to be compared, i.e. `\l__dbshow_filter_tmp_tl` to see if the result appears in the ⟨*review points*⟩.

```
  \bool_if:NTF #1 {
    \int_zero_new:N \l__dbshow_filter_diff_int
    \exp_last_unbraced:NNNV \__dbshow_parse_date_cond:NNw
      \l__dbshow_filter_tmp_clist \l__dbshow_filter_tmp_tl {#2} \__dbshow_stop
    \__dbdate_clear_new:n { tmp_day1 }
    \__dbdate_clear_new:n { tmp_day2 }
```

```
    \__dbdate_set:xx { tmp_day1 } { \l__dbshow_filter_tmp_tl }
    \__dbdate_set:xx { tmp_day2 } {#3}
    \__dbdate_sub:nnN { tmp_day1 } { tmp_day2 } \l__dbshow_filter_diff_int
    \clist_if_in:NxTF \l__dbshow_filter_tmp_clist
      { \int_use:N \l__dbshow_filter_diff_int } {#4} {#5}
  } {
```

For unstarred ⟨*conditional*⟩ which parses ⟨*expr*⟩. We first replace each date to an absolute integer. We did not use `\regex_replace_all:nnN` because `\_dbdate_to_int:nN` is nonexpandable. So the code seems a little complicated and unsightly, but it worked. Note that ⟨*expr*⟩ should be updated locally.

```
    \int_zero_new:N \l__dbshow_filter_tmpa_int
    \int_zero_new:N \l__dbshow_filter_tmpb_int
    \tl_set:Nx \l__dbshow_expr_tl {#2}
    \regex_extract_all:nVN { \d{4}/\d+/\d+ }
      \l__dbshow_expr_tl \l__dbshow_filter_date_seq
    \regex_split:nVN { \d{4}/\d+/\d+ }
      \l__dbshow_expr_tl \l__dbshow_filter_other_seq
    \tl_clear:N \l__dbshow_expr_tl
    \int_set:Nn \l__dbshow_filter_tmpa_int
      { \seq_count:N \l__dbshow_filter_date_seq }
    \int_step_inline:nn { \l__dbshow_filter_tmpa_int } {
      \tl_put_right:Nx \l__dbshow_expr_tl
        { \seq_item:Nn \l__dbshow_filter_other_seq {##1} }
      \__dbdate_clear_new:n { date-tmp }
      \__dbdate_set:xx { date-tmp }
        { \seq_item:Nn \l__dbshow_filter_date_seq {##1} }
      \__dbdate_to_int:nN { date-tmp } \l__dbshow_filter_tmpb_int
      \tl_put_right:Nx \l__dbshow_expr_tl
        { \int_use:N \l__dbshow_filter_tmpb_int }
    }
    \tl_put_right:Nx \l__dbshow_expr_tl {
      \seq_item:Nn \l__dbshow_filter_other_seq
        { \l__dbshow_filter_tmpa_int + 1 }
    }
    \int_compare:nTF { \l__dbshow_expr_tl } {#4} {#5}
  }
}
\cs_generate_variant:Nn \__dbshow_filter_date:NNNnn { cccnn }
```

(*End definition for* `\__dbshow_filter_date:NNNnn`.)

`\__dbshow_filter:nnn` Filter records with ⟨*conditional*⟩

`#1` : ⟨*database*⟩
`#2` : ⟨*conditional*⟩
`#3` : ⟨*index*⟩

```
\cs_set:Nn \__dbshow_filter:nnn {
  \tl_set_eq:Nc \l__dbshow_attr_tl { g__dbshow_filter_attr_tl_#1_#2 }
  \cs_set_eq:Nc \dbval { g__dbshow_data_#1_\l__dbshow_attr_tl _#3 }
  \use:c
    { __dbshow_filter_\__dbshow_get_type:nV {#1} \l__dbshow_attr_tl :cccnn }
```

```
    { g__dbshow_cond_star_bool_#1_#2 }
    { g__dbshow_filter_expr_tl_#1_#2 }
    { dbval }
    { \bool_gset_true:c  { g__dbshow_filter_bool_#1_#2 } }
    { \bool_gset_false:c { g__dbshow_filter_bool_#1_#2 } }
}
```

(*End definition for* `\__dbshow_filter:nnn`.)

`\__dbshow_new_conditional:nnnnn` For a ⟨*conditional*⟩ of ⟨*attr*⟩, map ⟨*conditional*⟩ to ⟨*attr*⟩ and map ⟨*conditional*⟩ to expr. The ⟨*boolean var*⟩ is created to store the filter result. An hook function is defined and the ⟨*conditional*⟩ is recorded in the sequence.

`#1` : ⟨*database*⟩
`#2` : ⟨*conditional*⟩
`#3` : ⟨*attr*⟩
`#4` : ⟨*expr*⟩
`#5` : ⟨*star*⟩

```
\cs_new_protected:Nn \__dbshow_new_conditional:nnnnn {
  \tl_gset:cn { g__dbshow_filter_attr_tl_#1_#2 } {#3}
  \tl_gset:cn { g__dbshow_filter_expr_tl_#1_#2 } {#4}
  \bool_if_exist:cF { g__dbshow_filter_bool_#1_#2 }
    { \bool_new:c { g__dbshow_filter_bool_#1_#2 } }
  \bool_if_exist:cF { g__dbshow_cond_star_bool_#1_#2 }
    { \bool_new:c { g__dbshow_cond_star_bool_#1_#2 } }
  \IfBooleanTF {#5}
    { \bool_gset_true:c { g__dbshow_cond_star_bool_#1_#2 } }
    { \bool_gset_false:c { g__dbshow_cond_star_bool_#1_#2 } }
```

Filter hook function.

`##1` : ⟨*index*⟩

```
  \cs_gset:cn { g__dbshow_filter_hook_#1_#2:n } {
    \__dbshow_filter:nnn {#1} {#2} {##1}
  }
  \seq_gput_right:cn { g__dbshow_cond_seq_#1 } {#2}
}
```

(*End definition for* `\__dbshow_new_conditional:nnnnn`.)

`\__dbshow_combine_conditional:nnn` `\__dbshow_combine_conditional:nVn` First extract all the ⟨*conditional*⟩ in ⟨*conditional expr*⟩ and for every ⟨*conditional*⟩ in the record sequence, if it is in ⟨*conditional expr*⟩, then add the corresponding hook function to running sequence. Then replace all the ⟨*conditional*⟩ in ⟨*conditional expr*⟩ with corresponding boolean variable and save the result in the filter boolean variable.

`#1` : ⟨*database*⟩
`#2` : ⟨*filter*⟩
`#3` : ⟨*conditional expr*⟩

```
\cs_new_protected:Nn \__dbshow_combine_conditional:nnn {
  \tl_gset_eq:cN { g__dbshow_filter_bool_tl_#1_#2 } \c_true_bool
  \seq_gclear_new:c { g__dbshow_filter_run_seq_#1_#2 }
  \regex_extract_all:nnN { \w+ } {#3} \l__dbshow_cond_seq
```

##1 : ⟨*conditional*⟩

```
  \seq_map_inline:Nn \l__dbshow_cond_seq {
    \seq_if_in:cnT { g__dbshow_cond_seq_#1 } {##1} {
      \seq_gput_right:cn { g__dbshow_filter_run_seq_#1_#2 }
        { g__dbshow_filter_hook_#1_##1:n }
    }
  }
  \tl_set:Nn \l__dbshow_cond_expr_tl {#3}
  \regex_replace_all:nnN
    { \w+ } { \c{ g__dbshow_filter_bool_#1_\0 } }
    \l__dbshow_cond_expr_tl
  \tl_gset_eq:cN
    { g__dbshow_filter_bool_tl_#1_#2 } \l__dbshow_cond_expr_tl
}
\cs_generate_variant:Nn \__dbshow_combine_conditional:nnn { nVn }
```

(*End definition for* `\__dbshow_combine_conditional:nnn`.)

dbFilters
\dbNewConditional
\dbCombineConditionals

Environment to define conditionals and filters.

#1 : ⟨*database*⟩

```
\NewDocumentEnvironment { dbFilters } { m } {
  \seq_gclear_new:c { g__dbshow_cond_seq_#1 }
```

##1 : ⟨*star*⟩
##2 : ⟨*conditional*⟩
##3 : ⟨*attr*⟩
##4 : ⟨*expr*⟩

```
  \DeclareDocumentCommand { \dbNewConditional } { s m m m } {
    \__dbshow_new_conditional:nnnnn
      {#1} {##2} {##3} {##4} {##1}
  }
```

##1 : ⟨*filter*⟩
##2 : ⟨*conditional expr*⟩
##3 : ⟨*filter info*⟩

```
  \DeclareDocumentCommand { \dbCombineConditionals } { m m O{} } {
    \tl_gset:cn { g__dbshow_filter_info_tl_#1_##1 } {##3}
    \__dbshow_combine_conditional:nnn {#1} {##1} {##2}
  }
} {}
```

(*End definition for* `dbFilters`, `\dbNewConditional`, *and* `\dbCombineConditionals`. *These functions are documented on page* 18.)

\dbNewReviewPoints

User interface to define ⟨*review points*⟩.

#1 : ⟨*name*⟩
#2 : ⟨*points*⟩

```
\NewDocumentCommand { \dbNewReviewPoints } { m m } {
  \clist_set:cn { g__review_points_#1 } {#2}
}
```

(*End definition for* `\dbNewReviewPoints`. *This function is documented on page 17.*)

`\dbIntAbs` Function aliases to support more operations of integer and floating points.

`\dbIntSign`
`\dbIntDivRound`
`\dbIntDivTruncate`
`\dbIntMax`
`\dbIntMin`
`\dbIntMod`
`\dbFpSign`

```
\cs_set_eq:NN \dbIntAbs         \int_abs:n
\cs_set_eq:NN \dbIntSign        \int_sign:n
\cs_set_eq:NN \dbIntDivRound    \int_div_round:nn
\cs_set_eq:NN \dbIntDivTruncate \int_div_truncate:nn
\cs_set_eq:NN \dbIntMax         \int_max:nn
\cs_set_eq:NN \dbIntMin         \int_min:nn
\cs_set_eq:NN \dbIntMod         \int_mod:nn
\cs_set_eq:NN \dbFpSign         \fp_sign:n
```

(*End definition for* `\dbIntAbs` *and others. These functions are documented on page 20.*)

7.6 Style and Options

`\__dbshow_identity:n` Does nothing and keep things unchanged.

```
\cs_new:Nn \__dbshow_identity:n {#1}
```

(*End definition for* `\__dbshow_identity:n`.)

`\__dbshow_new_attr_style:nnn` Define style keys for each attribute.

`#1` : ⟨*style*⟩
`#2` : ⟨*database*⟩
`#3` : ⟨*attr*⟩

```
\cs_new_protected:Nn \__dbshow_new_attr_style:nnn {
  \__dbshow_check_attr:nn {#2} {#3}
  \keys_define:nn { dbshow/style/#1/#3 } {
    before-code      .tl_gset:c    = {
      g__dbshow_style_attr_before_tl_#1_#2_#3
    },
    before-code      .initial:n    = ,
    after-code       .tl_gset:c    = {
      g__dbshow_style_attr_after_tl_#1_#2_#3
    },
    after-code       .initial:n    = ,
```

For comma list and date.

```
    sep              .clist_gset:c = {
      g__dbshow_style_attr_sep_#1_#2_#3
    },
```

Only for comma list.

```
    item-before-code .tl_gset:c    = {
      g__dbshow_style_attr_item_before_tl_#1_#2_#3
    },
    item-before-code .initial:n    = ,
    item-after-code  .tl_gset:c    = {
      g__dbshow_style_attr_item_after_tl_#1_#2_#3
    },
    item-after-code  .initial:n    = ,
```

Only for date.

```
    zfill            .bool_gset:c  = {
      g__dbshow_style_attr_zfill_bool_#1_#2_#3
    },
    zfill            .initial:n    = true,
    zfill            .default:n    = true,
    wrapper          .tl_gset:c    = {
      g__dbshow_style_attr_wrapper_#1_#2_#3
    },
    wrapper          .initial:n    = { \__dbshow_identity:n },
  }
  \str_case_e:nn { \__dbshow_get_type:nn {#2} {#3} } {
    { clist }
    { \keys_set:nn { dbshow/style/#1/#3 } { sep = { { ,~ } } } }
    { date }
    { \keys_set:nn { dbshow/style/#1/#3 } { sep = { { /  } } } }
  }
}
```

(*End definition for* `\__dbshow_new_attr_style:nnn`.)

`\__dbshow_new_database_style:nn` Define style keys.

`#1` : ⟨*style*⟩
`#2` : ⟨*database*⟩

```
\cs_new_protected:Nn \__dbshow_new_database_style:nn {
  \__dbshow_check_database:n {#2}
  \keys_define:nn { dbshow/style/#1 } {
    raw-filter         .code:n       = {
      \int_gincr:N  \g__dbshow_raw_filter_int
      \str_set:Nx \l__dbshow_raw_filter_str
        { -raw\int_use:N \g__dbshow_raw_filter_int - }
      \tl_gset:cV { g__dbshow_filter_#1_#2 } \l__dbshow_raw_filter_str
      \__dbshow_combine_conditional:nnn {#2}  \l__dbshow_raw_filter_str {##1}
    },
    filter             .tl_gset:c    = { g__dbshow_filter_#1_#2 },
    filter             .initial:n    = -none-,
    sort               .clist_gset:c = { g__dbshow_sort_clist_#1_#2 },
    before-code        .tl_gset:c    = { g__dbshow_style_before_tl_#1_#2 },
    before-code        .initial:n    = ,
    item-code          .tl_gset:c    = { g__dbshow_style_database_item_tl_#1_#2 },
    item-code          .initial:n    = ,
    after-code         .tl_gset:c    = { g__dbshow_style_after_tl_#1_#2 },
    after-code         .initial:n    = ,
    record-before-code .tl_gset:c    = { g__dbshow_style_record_before_tl_#1_#2 },
    record-before-code .initial:n    = ,
    record-after-code  .tl_gset:c    = { g__dbshow_style_record_after_tl_#1_#2 },
    record-after-code  .initial:n    = ,
  }
  \prop_map_inline:cn { g__dbshow_attr_type_prop_#2 }
    { \__dbshow_new_attr_style:nnn {#1} {#2} {##1} }
}
```

(*End definition for* `\__dbshow_new_database_style:nn`.)

`\dbNewStyle` Set style options based on ⟨*base style*⟩.

`#1` : ⟨*base style clist*⟩
`#2` : ⟨*style*⟩
`#3` : ⟨*database*⟩
`#4` : ⟨*options*⟩

```
\NewDocumentCommand { \dbNewStyle } { o m m +m } {
  \tl_gset:cn { g__dbshow_style_opts_tl_#2_#3 } { #4, }
  \IfValueT {#1} {
    \tl_clear_new:N \l__dbshow_style_tmp_tl
```

`##1` : ⟨*base style*⟩

```
    \clist_map_inline:nn {#1} {
      \__dbshow_check_style:nn {##1} {#3}
      \tl_if_exist:cT { g__dbshow_style_opts_tl_##1_#3 } {
        \tl_concat:ccc { l__dbshow_style_tmp_tl }
          { l__dbshow_style_tmp_tl } { g__dbshow_style_opts_tl_##1_#3 }
      }
    }
    \tl_gconcat:ccc { g__dbshow_style_opts_tl_#2_#3 }
      { l__dbshow_style_tmp_tl } { g__dbshow_style_opts_tl_#2_#3 }
  }
  \__dbshow_new_database_style:nn {#2} {#3}
  \keys_set:nv { dbshow/style/#2 } { g__dbshow_style_opts_tl_#2_#3 }
}
```

(*End definition for* `\dbNewStyle`. *This function is documented on page* 14.)

7.7 Sort

`\__dbshow_sort_parse_star:NNNw` Parse descending sorting rule.

`#1` : ⟨*tl var*⟩ representing for relation to keep order the same
`#2` : ⟨*tl var*⟩ representing for relation to swap the order
`#3` : ⟨*tl var*⟩ to store the ⟨*attr*⟩

```
\cs_new_protected:Npn \__dbshow_sort_parse_star:NNNw #1#2#3#4* {
  \tl_set:Nn #1 { > }
  \tl_set:Nn #2 { < }
  \tl_set:Nn #3 {#4}
}
```

(*End definition for* `\__dbshow_sort_parse_star:NNNw`.)

`\__dbshow_sort:nNn` Sort the records.

`#1` : ⟨*database*⟩
`#2` : ⟨*index clist*⟩
`#3` : ⟨*style*⟩

```
\cs_new_protected:Nn \__dbshow_sort:nNn {
  \int_zero_new:N \l__dbshow_sort_len_int
  \int_zero_new:N \l__dbshow_sort_tmp_int
  \int_set:Nn \l__dbshow_sort_len_int
    { \clist_count:c { g__dbshow_sort_clist_#3_#1 } }
```

Sort recursively.

```
  \clist_sort:Nn #2 {
    \int_zero:N \l__dbshow_sort_tmp_int
```

Sort single attribute.

```
    \cs_set:Nn \__dbshow_sort_single: {
      \int_incr:N \l__dbshow_sort_tmp_int
      \str_set:Nx \l__dbshow_sort_attr_str {
        \clist_item:cn
          { g__dbshow_sort_clist_#3_#1 }
          { \l__dbshow_sort_tmp_int }
      }
```

Parse ⟨*attr*⟩ and corresponding ⟨*type*⟩ and set compare operators.

```
      \str_if_in:NnTF \l__dbshow_sort_attr_str { * } {
        \exp_after:wN \__dbshow_sort_parse_star:NNNw
          \exp_after:wN \l__dbshow_sort_same_op_tl
          \exp_after:wN \l__dbshow_sort_swap_op_tl
          \exp_after:wN \l__dbshow_sort_attr_str
          \l__dbshow_sort_attr_str
      } {
        \tl_set:Nn \l__dbshow_sort_same_op_tl { < }
        \tl_set:Nn \l__dbshow_sort_swap_op_tl { > }
      }
```

Check if type is valid and transform date to string.

```
      \__dbshow_check_attr:nV {#1} \l__dbshow_sort_attr_str
      \tl_set:Nx \l__dbshow_sort_type_tl
        { \__dbshow_get_type:nV {#1} \l__dbshow_sort_attr_str }
      \clist_if_in:nVF
      { str, int, date, fp } { \l__dbshow_sort_type_tl } {
        \msg_error:nnx { dbshow } { unsupported-sort-type }
          { \l__dbshow_sort_type_tl }
      }
      \str_if_eq:eeT { \l__dbshow_sort_type_tl } { date }
        { \tl_set:Nn \l__dbshow_sort_type_tl { str } }
```

Set operands and compare function.

```
      \cs_set_eq:Nc \l__dbshow_sort_tmpa_tl
        { g__dbshow_data_#1_\l__dbshow_sort_attr_str _##1 }
      \cs_set_eq:Nc \l__dbshow_sort_tmpb_tl
        { g__dbshow_data_#1_\l__dbshow_sort_attr_str _##2 }
      \cs_set_eq:Nc \__dbshow_compare
        { \l__dbshow_sort_type_tl _compare:VNVTF }
```

Compare two operands and if they are equal, compare the next attribute.

```
      \__dbshow_compare \l__dbshow_sort_tmpa_tl
        \l__dbshow_sort_same_op_tl \l__dbshow_sort_tmpb_tl
        { \sort_return_same: }
      {
        \__dbshow_compare \l__dbshow_sort_tmpa_tl
          \l__dbshow_sort_swap_op_tl \l__dbshow_sort_tmpb_tl
          { \sort_return_swapped: }
        {
          \int_compare:nTF
            { \l__dbshow_sort_len_int = \l__dbshow_sort_tmp_int }
            { \sort_return_same: }
            { \__dbshow_sort_single: }
        }
      }
    }
    \__dbshow_sort_single:
  }
}
```

(*End definition for* `\__dbshow_sort:nNn`.)

7.8 Display Data

`\__dbshow_clist_wrapper:NNn` Wrap the clist item with ⟨*before code*⟩ and ⟨*after code*⟩.

`#1` : ⟨*before code tl*⟩
`#2` : ⟨*after code tl*⟩
`#3` : ⟨*item*⟩

```
\cs_new:Nn \__dbshow_clist_wrapper:NNn {
  \exp_not:n { { #1#3#2 }, }
}
```

(*End definition for* `\__dbshow_clist_wrapper:NNn`.)

`\__dbshow_clist_use:NNNN` Display a comma list.
`\__dbshow_clist_use:cccc`

`#1` : ⟨*data clist*⟩
`#2` : ⟨*separator clist*⟩
`#3` : ⟨*before code tl*⟩
`#4` : ⟨*after code tl*⟩

```
\cs_new:Nn \__dbshow_clist_use:NNNN {
  \int_case:nnF { \clist_count:N #2 } {
    { 1 } {
      \clist_use:xx
        { \clist_map_tokens:Nn #1 { \__dbshow_clist_wrapper:NNn #3 #4 } }
        { \clist_item:Nn #2 { 1 } }
    }
    { 3 } {
      \clist_use:xxxx
        { \clist_map_tokens:Nn #1 { \__dbshow_clist_wrapper:NNn #3 #4 } }
```

```
        { \clist_item:Nn #2 { 1 } }
        { \clist_item:Nn #2 { 2 } }
        { \clist_item:Nn #2 { 3 } }
    }
  } {
    \__dbshow_sep_error:xxx
      { 1~or~3 }
      { \clist_count:N #2 }
      { \clist_use:Nn #2 { ,~ } }
  }
}
\cs_generate_variant:Nn \__dbshow_clist_use:NNNN { cccc }
```

(*End definition for* `\__dbshow_clist_use:NNNN`.)

`\__dbshow_date_use:nNN` Display date.

`\__dbshow_date_use:ncc`

#1 : ⟨*data*⟩
#2 : ⟨*separator clist*⟩
#3 : ⟨*zfill boolean*⟩

```
\cs_new:Nn \__dbshow_date_use:nNN {
  \int_case:nnF { \clist_count:N #2 } {
    { 1 } {
      \bool_if:NTF {#3}
        { \__dbdate_use_zfill:nf }
        { \__dbdate_use:nf }
        {#1}
        { \clist_item:Nn #2 { 1 } }
    }
    { 4 } {
      \bool_if:NTF {#3}
        { \__dbdate_use_zfill:nffff }
        { \__dbdate_use:nffff }
        {#1}
        { \clist_item:Nn #2 { 1 } }
        { \clist_item:Nn #2 { 2 } }
        { \clist_item:Nn #2 { 3 } }
        { \clist_item:Nn #2 { 4 } }
    }
  } {
    \__dbshow_sep_error:xxx
      { 1~or~4 }
      { \clist_count:N #2 }
      { \clist_use:Nn #2 { ,~ } }
  }
}
\cs_generate_variant:Nn \__dbshow_date_use:nNN { ncc }
```

(*End definition for* `\__dbshow_date_use:nNN`.)

`\__dbshow_use_data:nnnn` Display Data.

#1 : ⟨*database*⟩

#2 : ⟨*attr*⟩
#3 : ⟨*index*⟩
#4 : ⟨*style*⟩

```
% #1 database #2 attr #3 index #4 style
\cs_new:Nn \__dbshow_use_data:nnnn {
  \str_case_e:nn
  { \prop_item:cn { g__dbshow_attr_type_prop_#1 } {#2} } {
    { str }   { \str_use:c { g__dbshow_data_#1_#2_#3 } }
    { tl }    { \tl_use:c { g__dbshow_data_#1_#2_#3 } }
    { int }   { \int_use:c { g__dbshow_data_#1_#2_#3 } }
    { fp }    { \fp_use:c { g__dbshow_data_#1_#2_#3 } }
    { clist } {
      \__dbshow_clist_use:cccc { g__dbshow_data_#1_#2_#3 }
        { g__dbshow_style_attr_sep_#4_#1_#2 }
        { g__dbshow_style_attr_item_before_tl_#4_#1_#2 }
        { g__dbshow_style_attr_item_after_tl_#4_#1_#2 }
    }
    { date }  {
      \exp_args:Nnx
      \tl_use:c { g__dbshow_style_attr_wrapper_#4_#1_#2 } {
        \__dbshow_date_use:ncc { g__dbshow_data_#1_#2_#3 }
          { g__dbshow_style_attr_sep_#4_#1_#2 }
          { g__dbshow_style_attr_zfill_bool_#4_#1_#2 }
      }
    }
  }
}
```

(*End definition for* `\__dbshow_use_data:nnnn`.)

`\dbDatabase` Define context macros.
`\dbFilterName`
`\dbFilterInfo`

#1 : ⟨*database*⟩
#2 : ⟨*filter*⟩

```
\cs_new_protected:Nn \__dbshow_show_set_macro:nn {
  \tl_set:Nn    \dbDatabase   {#1}
  \tl_set:Nn    \dbFilterName {#2}
  \tl_set_eq:Nc \dbFilterInfo { g__dbshow_filter_info_tl_#1_#2 }
}
```

(*End definition for* `\dbDatabase`, `\dbFilterName`, *and* `\dbFilterInfo`. *These functions are documented on page* 20.)

`\__dbshow_show_filter:nn` Filter records by executing the hook function in the running sequence and then testing the result boolean.

#1 : ⟨*database*⟩
#2 : ⟨*filter*⟩
#3 : ⟨*index clist*⟩

```
\cs_new_protected:Nn \__dbshow_show_filter:nnN {
```

##1 : ⟨*index*⟩

```
    \int_step_inline:nn { \__dbshow_get_counter:n {#1} } {
      \seq_if_exist:cTF { g__dbshow_filter_run_seq_#1_#2 } {
```

####1 : ⟨*hook*⟩

```
        \seq_map_inline:cn { g__dbshow_filter_run_seq_#1_#2 } {
          \use:c {####1} {##1}
        }
        \exp_args:Nv
        \bool_if:nT { g__dbshow_filter_bool_tl_#1_#2 }
          { \clist_put_right:Nn #3 {##1} }
      } { \clist_put_right:Nn #3 {##1} }
    }
  }
```

(*End definition for* `\__dbshow_show_filter:nn`.)

`\__dbshow_show_set_counter:N` Define macros to display counter.

`\dbalph` `\dbAlph` `\dbarabic` `\dbroman` `\dbRoman`

#1 : ⟨*int var*⟩

```
\cs_new_protected:Nn \__dbshow_show_set_counter:N {
  \tl_set:Nx \dbalph   { \int_to_alph:n {#1} }
  \tl_set:Nx \dbAlph   { \int_to_Alph:n {#1} }
  \tl_set:Nx \dbarabic { \int_to_arabic:n {#1} }
  \tl_set:Nx \dbRoman  { \int_to_Roman:n {#1} }
  \tl_set:Nx \dbroman  { \int_to_roman:n {#1} }
}
```

(*End definition for* `\__dbshow_show_set_counter:N` *and others. These functions are documented on page 20.*)

`\__dbshow_show_item:nn` Display single record.

`\dbIndex` `\dbuse`

#1 : ⟨*style*⟩
#2 : ⟨*database*⟩
#3 : ⟨*index clist*⟩

```
\cs_new_protected:Nn \__dbshow_show_item:nnN {
  \int_zero_new:N \l__dbshow_show_int
```

##1 : ⟨*index*⟩

```
  \clist_map_inline:Nn #3 {
    \int_incr:N \l__dbshow_show_int
    \__dbshow_show_set_counter:N \l__dbshow_show_int
    \tl_set:Nn \dbIndex {##1}
```

####1 : ⟨*attr*⟩

```
    \cs_set:Npn \dbuse ####1 {
      \__dbshow_check_attr:nn {#2} {####1}
      \tl_use:c { g__dbshow_style_attr_before_tl_#1_#2_####1 }
      \__dbshow_use_data:nnnn {#2} {####1} {##1} {#1}
      \tl_use:c { g__dbshow_style_attr_after_tl_#1_#2_####1 }
    }
    \tl_use:c { g__dbshow_style_record_before_tl_#1_#2 }
```

```
    \tl_use:c { g__dbshow_style_database_item_tl_#1_#2 }
    \tl_use:c { g__dbshow_style_record_after_tl_#1_#2 }
  }
}
```

(*End definition for* `\__dbshow_show_item:nn`, `\dbIndex`, *and* `\dbuse`. *These functions are documented on page* 20.)

`\__dbshow_show_set_cond:N` Define conditional to test if the number of records to show is zero.

`\dbIfEmptyT`
`\dbIfEmptyF`
`\dbIfEmptyTF`

`#1` : ⟨*index clist*⟩

```
\cs_new_protected:Nn \__dbshow_show_set_cond:N {
  \prg_set_conditional:Nnn \__dbshow_if_empty: { T, F, TF } {
    \clist_if_empty:NTF #1
      { \prg_return_true: }
      { \prg_return_false: }
  }
  \cs_set_eq:NN \dbIfEmptyT  \__dbshow_if_empty:T
  \cs_set_eq:NN \dbIfEmptyF  \__dbshow_if_empty:F
  \cs_set_eq:NN \dbIfEmptyTF \__dbshow_if_empty:TF
}
```

(*End definition for* `\__dbshow_show_set_cond:N` *and others. These functions are documented on page* 19.)

`\__dbshow_show:nnn` First filter records and sort them if needed and display at last.

`\__dbshow_show:nnv`

```
\cs_new_protected:Nn \__dbshow_show:nnn {
  \__dbshow_show_set_macro:nn {#2} {#3}
  \clist_clear_new:N \l__dbshow_show_index_clist
  \__dbshow_show_filter:nnN {#2} {#3} \l__dbshow_show_index_clist
  \clist_if_empty:cF { g__dbshow_sort_clist_#1_#2 }
    { \__dbshow_sort:nNn {#2} \l__dbshow_show_index_clist {#1} }
  \tl_use:c { g__dbshow_style_before_tl_#1_#2 }
  \__dbshow_show_item:nnN {#1} {#2} \l__dbshow_show_index_clist
  \tl_use:c { g__dbshow_style_after_tl_#1_#2 }
}
\cs_generate_variant:Nn \__dbshow_show:nnn { nnv }
```

(*End definition for* `\__dbshow_show:nnn`.)

`\dbshow` User insterface to display the ⟨*database*⟩ with ⟨*style*⟩.

`#1` : ⟨*style*⟩
`#2` : ⟨*database*⟩

```
\NewDocumentCommand { \dbshow } { m m } {
  \__dbshow_check_database:n {#2}
  \__dbshow_check_filter:nv {#2} { g__dbshow_filter_#1_#2 }
  \__dbshow_show:nnv {#1} {#2} { g__dbshow_filter_#1_#2 }
}
```

(*End definition for* `\dbshow`. *This function is documented on page* 14.)

7.9 Date Type

```
⟨@@=dbdate⟩
```

`\__dbdate_if_leap_p:n` Check if the year is leap.
`\__dbdate_if_leap:nTF`

`#1` : ⟨*year*⟩

```
\prg_new_conditional:Nnn \__dbdate_if_leap:n { T, F, TF, p } {
  \bool_if:nTF {
    \int_compare_p:nNn { \int_mod:nn {#1} { 400 } } = { 0 } ||
    (!\int_compare_p:nNn { \int_mod:nn {#1} { 100 } } = { 0 } &&
      \int_compare_p:nNn { \int_mod:nn {#1} { 4 } } = { 0 })
  } { \prg_return_true: } { \prg_return_false: }
}
```

(*End definition for* `\__dbdate_if_leap:nTF`.)

`\c__dbdate_month_clist` Number of days of every month.

```
\clist_const:Nn \c__dbdate_month_clist
  { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
```

(*End definition for* `\c__dbdate_month_clist`.)

`\__dbdate_to_int:nnnN` Transform date to integer relative to `1971-01-01`.

`#1` : ⟨*year*⟩
`#2` : ⟨*month*⟩
`#3` : ⟨*day*⟩
`#4` : ⟨*int var*⟩ to store the result

```
\cs_new_protected:Nn \__dbdate_to_int:nnnN {
  \int_zero_new:N \l__dbdate_ans_int
  \int_zero_new:N \l__dbdate_tmpa_int
  \int_zero_new:N \l__dbdate_tmpb_int
  \int_set:Nn \l__dbdate_ans_int { #3 - 1 }
  \int_step_inline:nn { #2 - 1 } {
    \int_add:Nn \l__dbdate_ans_int {
      \clist_item:Nn \c__dbdate_month_clist {##1}
    }
    \bool_if:nT {
      \int_compare_p:nNn {##1} = { 2 } &&
      \__dbdate_if_leap_p:n {#1}
    } { \int_incr:N \l__dbdate_ans_int }
  }
  \int_add:Nn \l__dbdate_ans_int { 365 * (#1 - 1971) }
  \int_add:Nn \l__dbdate_ans_int {
    \int_div_truncate:nn { #1 - 1 } { 4 } -
    \int_div_truncate:nn { 1971 } { 4 }
  }
  \int_sub:Nn \l__dbdate_ans_int {
    \int_div_truncate:nn { #1 - 1 } { 100 } -
    \int_div_truncate:nn { 1971 } { 100 }
  }
```

```
    \int_add:Nn \l__dbdate_ans_int {
      \int_div_truncate:nn { #1 - 1 } { 400 } -
      \int_div_truncate:nn { 1971 } { 400 }
    }
    \int_set_eq:NN #4 \l__dbdate_ans_int
}
```

(*End definition for* `\__dbdate_to_int:nnnN`.)

`\__dbdate_to_int:NNNN` Transform date to integer relative to `1971-01-01`.

`\__dbdate_to_int:cccN`

`#1` : ⟨*year int var*⟩
`#2` : ⟨*month int var*⟩
`#3` : ⟨*day int var*⟩
`#4` : ⟨*int var*⟩ to store the result

```
\cs_new_protected:Nn \__dbdate_to_int:NNNN {
  \__dbdate_to_int:nnnN {#1} {#2} {#3} #4
}
\cs_generate_variant:Nn \__dbdate_to_int:NNNN { cccN }
```

(*End definition for* `\__dbdate_to_int:NNNN`.)

`\__dbdate_to_int:nN` Transform date to integer relative to `1971-01-01`.

`#1` : ⟨*date var*⟩
`#2` : ⟨*int var*⟩ to store the result

```
\cs_new:Nn \__dbdate_to_int:nN {
  \__dbdate_to_int:cccN
    { __dbdate_year_#1 }
    { __dbdate_month_#1 }
    { __dbdate_day_#1 }
    #2
}
```

(*End definition for* `\__dbdate_to_int:nN`.)

`\__dbdate_set_val:n` Set the value of ⟨*data var*⟩ to `yyyy/mm/dd`.

`\__dbdate_gset_val:n`

`#1` : ⟨*date var*⟩

```
\cs_new_protected:Nn \__dbdate_set_val:n {
  \tl_set:cx {#1} { \__dbdate_use_zfill:nn {#1} { \g__dbdate_sep_tl } }
}
\cs_new_protected:Nn \__dbdate_gset_val:n {
  \tl_gset:cx {#1} { \__dbdate_use_zfill:nn {#1} { \g__dbdate_sep_tl } }
}
```

(*End definition for* `\__dbdate_set_val:n` *and* `\__dbdate_gset_val:n`.)

`\__dbdate_init:n` Initialize ⟨*date var*⟩.

`\__dbdate_ginit:n`

`#1` : ⟨*date var*⟩

```
\cs_new_protected:Nn \__dbdate_init:n {
  \__dbdate_set:nnnn {#1} { 1971 } { 1 } { 1 }
  \__dbdate_set_val:n {#1}
}
\cs_new_protected:Nn \__dbdate_ginit:n {
  \__dbdate_gset:nnnn {#1} { 1971 } { 1 } { 1 }
  \__dbdate_gset_val:n {#1}
}
```

(*End definition for* `\__dbdate_init:n` *and* `\__dbdate_ginit:n`.)

`\__dbdate_new:n` Create a new date variable.

`\__dbdate_new:x`

`#1` : ⟨*date var*⟩

```
\cs_new_protected:Nn \__dbdate_new:n {
  \int_new:c { __dbdate_year_#1 }
  \int_new:c { __dbdate_month_#1 }
  \int_new:c { __dbdate_day_#1 }
  \__dbdate_ginit:n {#1}
}
\cs_generate_variant:Nn \__dbdate_new:n { x }
```

(*End definition for* `\__dbdate_new:n`.)

`\__dbdate_clear_new:n` Clear or create a new date variable.

`\__dbdate_clear_new:x`

`\__dbdate_gclear_new:n`

`\__dbdate_gclear_new:x`

`#1` : ⟨*date var*⟩

```
\cs_new_protected:Nn \__dbdate_clear_new:n {
  \int_zero_new:c { __dbdate_year_#1 }
  \int_zero_new:c { __dbdate_month_#1 }
  \int_zero_new:c { __dbdate_day_#1 }
  \__dbdate_init:n {#1}
}
\cs_generate_variant:Nn \__dbdate_clear_new:n { x }
\cs_new_protected:Nn \__dbdate_gclear_new:n {
  \int_gzero_new:c { __dbdate_year_#1 }
  \int_gzero_new:c { __dbdate_month_#1 }
  \int_gzero_new:c { __dbdate_day_#1 }
  \__dbdate_ginit:n {#1}
}
\cs_generate_variant:Nn \__dbdate_gclear_new:n { x }
```

(*End definition for* `\__dbdate_clear_new:n` *and* `\__dbdate_gclear_new:n`.)

`\__dbdate_set:nnnn` Set the value of ⟨*date var*⟩.

`\__dbdate_gset:nnnn`

`#1` : ⟨*date var*⟩

`#2` : ⟨*year*⟩

`#3` : ⟨*month*⟩

`#4` : ⟨*day*⟩

```
\cs_new_protected:Nn \__dbdate_set:nnnn {
  \int_set:cn { __dbdate_year_#1 }  {#2}
  \int_set:cn { __dbdate_month_#1 } {#3}
  \int_set:cn { __dbdate_day_#1 }   {#4}
  \__dbdate_set_val:n {#1}
}
\cs_new_protected:Nn \__dbdate_gset:nnnn {
  \int_gset:cn { __dbdate_year_#1 }  {#2}
  \int_gset:cn { __dbdate_month_#1 } {#3}
  \int_gset:cn { __dbdate_day_#1 }   {#4}
  \__dbdate_gset_val:n {#1}
}
```

(*End definition for* `\__dbdate_set:nnnn` *and* `\__dbdate_gset:nnnn`.)

`\__dbdate_set_sep:n` `\__dbdate_set:w` `\__dbdate_gset:w` `\dbdatesep`

Set internal date separator. Default is `/`.

`#1` : ⟨*separator*⟩

```
\cs_new_protected:Nn \__dbdate_set_sep:n {
  \tl_gset:Nn \g__dbdate_sep_tl { #1 }
}
```

Set the value of ⟨*date var*⟩.

`##1` : ⟨*date var*⟩
`##2` : ⟨*year*⟩
`##3` : ⟨*month*⟩
`##4` : ⟨*day*⟩

```
  \cs_gset_protected:Npn \__dbdate_set:w ##1\__dbdate_sep##2#1##3#1##4\__dbdate_stop {
    \__dbdate_clear_new:n {##1}
    \__dbdate_set:nnnn {##1} {##2} {##3} {##4}
  }
  \cs_gset_protected:Npn \__dbdate_gset:w ##1\__dbdate_sep##2#1##3#1##4\__dbdate_stop {
    \__dbdate_gclear_new:n {##1}
    \__dbdate_gset:nnnn {##1} {##2} {##3} {##4}
  }
}
\cs_gset_eq:NN \dbdatesep \__dbdate_set_sep:n
\dbdatesep{/}
```

(*End definition for* `\__dbdate_set_sep:n` *and others. This function is documented on page* 16.)

`\__dbdate_set:nn` `\__dbdate_set:xx` `\__dbdate_gset:nn` `\__dbdate_gset:xx`

Set the value of ⟨*date var*⟩.

`#1` : ⟨*date var*⟩
`#2` : ⟨*date*⟩

```
\cs_new_protected:Nn \__dbdate_set:nn {
  \__dbdate_set:w #1\__dbdate_sep#2\__dbdate_stop
}
\cs_generate_variant:Nn \__dbdate_set:nn { xx }
\cs_new_protected:Nn \__dbdate_gset:nn {
  \__dbdate_gset:w #1\__dbdate_sep#2\__dbdate_stop
}
\cs_generate_variant:Nn \__dbdate_gset:nn { xx }
```

(*End definition for* `\__dbdate_set:nn` *and* `\__dbdate_gset:nn`.)

`\__dbdate_sub:nnN` Calculate the number of days between ⟨*date var2*⟩ and ⟨*date var1*⟩.

`#1` : ⟨*date var1*⟩
`#2` : ⟨*date var2*⟩
`#3` : ⟨*int var*⟩ to store the result

```
\cs_new_protected:Nn \__dbdate_sub:nnN {
  \int_zero_new:N \l__dbdate_sub_tmpa_int
  \int_zero_new:N \l__dbdate_sub_tmpb_int
  \__dbdate_to_int:nN {#1} \l__dbdate_sub_tmpa_int
  \__dbdate_to_int:nN {#2} \l__dbdate_sub_tmpb_int
  \int_set:Nn #3 { \l__dbdate_sub_tmpa_int - \l__dbdate_sub_tmpb_int }
}
```

(*End definition for* `\__dbdate_sub:nnN`.)

`\__dbdate_show_two:N` `\__dbdate_show_two:c` Prepend 0 to single digit.

```
\cs_new:Nn \__dbdate_show_two:N {
  \int_compare:nNnTF {#1} > { 9 }
    { \int_use:N #1 } { 0\int_use:N #1 }
}
\cs_generate_variant:Nn \__dbdate_show_two:N { c }
```

(*End definition for* `\__dbdate_show_two:N`.)

`\__dbdate_use:nnnnn` `\__dbdate_use:nffff` `\__dbdate_use_zfill:nnnnn` `\__dbdate_use_zfill:nffff` Display date.

`#1` : ⟨*date var*⟩
`#2` : ⟨*separator 1*⟩
`#3` : ⟨*separator 2*⟩
`#4` : ⟨*separator 3*⟩
`#5` : ⟨*separator 4*⟩

```
\cs_new:Nn \__dbdate_use:nnnnn {
  #2\int_use:c { __dbdate_year_#1 }
  #3\int_use:c { __dbdate_month_#1 }
  #4\int_use:c { __dbdate_day_#1 }#5
}
\cs_generate_variant:Nn \__dbdate_use:nnnnn { nffff }
\cs_new:Nn \__dbdate_use_zfill:nnnnn {
  #2\int_use:c { __dbdate_year_#1 }
  #3\__dbdate_show_two:c { __dbdate_month_#1 }
  #4\__dbdate_show_two:c { __dbdate_day_#1 }#5
}
\cs_generate_variant:Nn \__dbdate_use_zfill:nnnnn { nffff }
```

(*End definition for* `\__dbdate_use:nnnnn` *and* `\__dbdate_use_zfill:nnnnn`.)

`\__dbdate_use:nn` `\__dbdate_use:nf` `\__dbdate_use_zfill:nn` `\__dbdate_use_zfill:nf` Display date with the same separator.

```
\cs_new:Nn \__dbdate_use:nn {
  \__dbdate_use:nnnnn {#1} {} {#2} {#2} {}
```

```
}
\cs_generate_variant:Nn \__dbdate_use:nn { nf }
\cs_new:Nn \__dbdate_use_zfill:nn {
  \__dbdate_use_zfill:nnnnn {#1} {} {#2} {#2} {}
}
\cs_generate_variant:Nn \__dbdate_use_zfill:nn { nf }
```

(*End definition for* `\__dbdate_use:nn` *and* `\__dbdate_use_zfill:nn`.)

`\__dbdate_show:n` Show date in terminal.

`#1` : ⟨*date var*⟩

```
\cs_new_protected:Nn \__dbdate_show:n {
  \exp_args:Nx \tl_show:n { >#1~=~\__dbdate_use:nn {#1} { - } }
}
```

(*End definition for* `\__dbdate_show:n`.)

`\dbtoday` Display date of today in `yyyy/mm/dd`.

```
\tl_set:Nn \dbtoday {
  \int_use:N \c_sys_year_int  \g__dbdate_sep_tl
  \int_use:N \c_sys_month_int \g__dbdate_sep_tl
  \int_use:N \c_sys_day_int
}
```

(*End definition for* `\dbtoday`. *This variable is documented on page* **??**.)

```
\endinput
```
⟨/package⟩

Change History

1.1

2022-01-05 **Add macro**: `\dbarabic`, `\dbalph`, `\dbAlph`, `\dbroman`, `\dbRoman` 11, 20
2022-01-06 **Add option**: `raw-filter` 5, 14
2022-01-06 **Fix bug**: `\dbIndex` not defined 11, 20
2022-01-07 **Update doc**: improve example 11, 21

1.2

2022-01-07 **Add doc**: add comparison to datatool 3, 13
2022-01-07 **Add macro**: `\dbclear` 4, 14
2022-01-08 **Add options**: `record-before-code`, `record-after-code` 7, 17
2022-01-08 **Fix bug**: string sorting bug 5, 14
2022-01-08 **Remove macros**: `\dbItemIfEmpty(TF)`, `\dbClistItemIfEmpty(TF)` 10, 20
2022-01-08 **Update macro**: make `\dbuse` fully-expandable 10, 19

1.3

2022-01-08 **Add macro**: `\dbsave*` 10, 19
2022-01-08 **Add option**: `<attr>/wrapper` 7, 17
2022-01-08 **Add option**: `<attr>/zfill` 6, 16
2022-01-08 **Remove dependency**: datatime2 . . . . . 3, 12
2022-01-08 **Update logic**: swap definition of starred and unstarred conditionals of date 9, 18
2022-01-09 **Add doc**: descripton for expansion 2, 12
2022-01-09 **Update option**: `<attr>/sep` 5, 15
2022-01-10 **Add macros**: expression function aliases 11, 20
2022-01-10 **Update doc**: truncated division 9, 18

1.4

2022-01-10 **Add check**: version of l3kernel 2, 12
2022-01-10 **Fix doc code**: wrong changes history sorting 54
2022-01-10 **Update macro**: change | to `\dbshow_sep` in weird argument of
`\dbshow_process_default_value:w` 4, 13
2022-01-13 **Add macro**: `\dbdatesep` 6, 16
2022-01-13 **Add macro**: `\dbitemkv` 10, 19
2022-01-13 **Update code**: merge code and doc into dbshow.dtx 25
2022-01-13 **Update env**: dbitem 9, 19
2022-01-14 **Fix bug**: data cannot contain `\par` 32

Add

v1.1 2022-01-05 **Add macro**: `\dbarabic`, `\dbalph`, `\dbAlph`, `\dbroman`, `\dbRoman` 11, 20
v1.1 2022-01-06 **Add option**: `raw-filter` 5, 14
v1.2 2022-01-07 **Add doc**: add comparison to datatool 3, 13
v1.2 2022-01-07 **Add macro**: `\dbclear` 4, 14
v1.2 2022-01-08 **Add options**: `record-before-code`, `record-after-code` 7, 17
v1.3 2022-01-08 **Add macro**: `\dbsave*` 10, 19
v1.3 2022-01-08 **Add option**: `<attr>/wrapper` 7, 17
v1.3 2022-01-08 **Add option**: `<attr>/zfill` 6, 16
v1.3 2022-01-09 **Add doc**: descripton for expansion 2, 12
v1.3 2022-01-10 **Add macros**: expression function aliases 11, 20
v1.4 2022-01-10 **Add check**: version of l3kernel 2, 12
v1.4 2022-01-13 **Add macro**: `\dbdatesep` 6, 16
v1.4 2022-01-13 **Add macro**: `\dbitemkv` 10, 19

Bug

v1.1 2022-01-06 **Fix bug**: `\dbIndex` not defined 11, 20
v1.2 2022-01-08 **Fix bug**: string sorting bug 5, 14
v1.4 2022-01-14 **Fix bug**: data cannot contain `\par` 32

Documentation

v1.1 2022-01-07 **Update doc**: improve example . . . 11, 21
v1.2 2022-01-07 **Add doc**: add comparison to datatool . . . 3, 13
v1.3 2022-01-09 **Add doc**: descripton for expansion . . . 2, 12
v1.3 2022-01-10 **Update doc**: truncated division . . . 9, 18
v1.4 2022-01-10 **Fix doc code**: wrong changes history sorting . . . 54

Fix

v1.1 2022-01-06 **Fix bug**: `\dbIndex` not defined . . . 11, 20
v1.2 2022-01-08 **Fix bug**: string sorting bug . . . 5, 14
v1.4 2022-01-10 **Fix doc code**: wrong changes history sorting . . . 54
v1.4 2022-01-14 **Fix bug**: data cannot contain `\par` . . . 32

Macro

v1.1 2022-01-05 **Add macro**: `\dbarabic`, `\dbalph`, `\dbAlph`, `\dbroman`, `\dbRoman` . . . 11, 20
v1.2 2022-01-07 **Add macro**: `\dbclear` . . . 4, 14
v1.2 2022-01-08 **Remove macros**: `\dbItemIfEmpty(TF)`, `\dbClistItemIfEmpty(TF)` . . . 10, 20
v1.2 2022-01-08 **Update macro**: make `\dbuse` fully-expandable . . . 10, 19
v1.3 2022-01-08 **Add macro**: `\dbsave*` . . . 10, 19
v1.3 2022-01-10 **Add macros**: expression function aliases . . . 11, 20
v1.4 2022-01-10 **Update macro**: change | to `\dbshow_sep` in weird argument of `\dbshow_process_default_value:w` . . . 4, 13
v1.4 2022-01-13 **Add macro**: `\dbdatesep` . . . 6, 16
v1.4 2022-01-13 **Add macro**: `\dbitemkv` . . . 10, 19

Option

v1.1 2022-01-06 **Add option**: `raw-filter` . . . 5, 14
v1.2 2022-01-08 **Add options**: `record-before-code`, `record-after-code` . . . 7, 17
v1.3 2022-01-08 **Add option**: `<attr>/wrapper` . . . 7, 17
v1.3 2022-01-08 **Add option**: `<attr>/zfill` . . . 6, 16
v1.3 2022-01-09 **Update option**: `<attr>/sep` . . . 5, 15

Remove

v1.2 2022-01-08 **Remove macros**: `\dbItemIfEmpty(TF)`, `\dbClistItemIfEmpty(TF)` . . . 10, 20
v1.3 2022-01-08 **Remove dependency**: datatime2 . . . 3, 12

Update

v1.1 2022-01-07 **Update doc**: improve example . . . 11, 21
v1.2 2022-01-08 **Update macro**: make `\dbuse` fully-expandable . . . 10, 19
v1.3 2022-01-08 **Update logic**: swap definition of starred and unstarred conditionals of date . . . 9, 18
v1.3 2022-01-09 **Update option**: `<attr>/sep` . . . 5, 15
v1.3 2022-01-10 **Update doc**: truncated division . . . 9, 18
v1.4 2022-01-10 **Update macro**: change | to `\dbshow_sep` in weird argument of `\dbshow_process_default_value:w` . . . 4, 13
v1.4 2022-01-13 **Update code**: merge code and doc into dbshow.dtx . . . 25
v1.4 2022-01-13 **Update env**: dbitem . . . 9, 19

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

Numbers

A

B

C

D

E

F

I

K

U

W

Z