dbshow 宏包 v1.5* ⟹ English Version

李昌锴 <lichangkai225@qq.com>

2022 年 1 月 17 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。

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

- 名字后带有 ⋆ 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 ⋆ 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。

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。

\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
Updated: 2022-01-15

```
\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` — `filter = ⟨filter⟩` (initially `-none-`, no default)

New: 2022-01-05

为当前样式设置由 `\dbCombineFilters` 所定义的过滤器。示例 1 演示了如何定义条件，将条件组合成过滤器以及使用过滤器。

示例 1：使用过滤器筛选数据

```
\dbNewDatabase{filter-db}{name=str, count=int}
\begin{dbFilters}{filter-db}
  \dbNewCond {cond1}{count}{\dbval > 3}
  \dbNewCond*{cond2}{name} {\d+}
  \dbCombCond{filter-and}{cond1 && cond2}
  \dbCombCond{filter-or} {cond1 || cond2}
\end{dbFilters}
\dbitemkv{filter-db}{name=123, count=4}
\dbitemkv{filter-db}{name=ab3, count=2}
\dbitemkv{filter-db}{name=bag, count=5}
\dbNewStyle{filter-and-style}{filter-db}{
  filter      = filter-and,
  before-code = \par Filter And\par,
  item-code   = {\dbuse{name}: \dbuse{count}\quad},
}
\dbNewStyle{filter-or-style}{filter-db}{
  filter      = filter-or,
  before-code = \par Filter Or\par,
  item-code   = {\dbuse{name}: \dbuse{count}\quad},
}
\dbshow{filter-and-style}{filter-db}
\dbshow{filter-or-style} {filter-db}
```

Filter And
123: 4
Filter Or
123: 4 bag: 5

`raw-filter` — `raw-filter = ⟨conditional expression⟩` (initially unset, no default)

New: 2022-01-06

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。示例 2 使用 `raw-filter` 选项，直接组合筛选条件，达到与示例 1 相同的效果。

示例 2：使用匿名过滤器筛选数据

```
\dbNewDatabase{filter-db}{name=str, count=int}
\begin{dbFilters}{filter-db}
  \dbNewCond {cond1}{count}{\dbval > 3}
  \dbNewCond*{cond2}{name} {\d+}
\end{dbFilters}
\dbitemkv{filter-db}{name=123, count=4}
\dbitemkv{filter-db}{name=ab3, count=2}
\dbitemkv{filter-db}{name=bag, count=5}
\dbNewStyle{filter-and-style}{filter-db}{
```

```
  raw-filter  = {cond1 && cond2},
  before-code = \par Filter And\par,
  item-code   = {\dbuse{name}: \dbuse{count}\quad},
}
\dbNewStyle{filter-or-style}{filter-db}{
  raw-filter  = {cond1 || cond2},
  before-code = \par Filter Or\par,
  item-code   = {\dbuse{name}: \dbuse{count}\quad},
}
\dbshow{filter-and-style}{filter-db}
\dbshow{filter-or-style} {filter-db}
```

Filter And
123: 4
Filter Or
123: 4 bag: 5

sort

New: 2022-01-05

`sort = { ⟨attr spec1⟩, ⟨attr spec2⟩, … }` (initially unset, no default)

为当前样式设置排序规则。支持对 `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。示例 3 的排序规则为先按 `count` 降序排序，`count` 相同的再按 `name` 增序排序。

示例 3：多级排序

```
\dbNewDatabase{sort-db}{name=str, count=int}
\dbNewStyle{sort-style}{sort-db}{
  sort = {count*, name},
  item-code = {\dbuse{name}: \dbuse{count}\quad}
}
\dbitemkv{sort-db}{name=bag, count=1}
\dbitemkv{sort-db}{name=box, count=1}
\dbitemkv{sort-db}{name=tag, count=2}
\dbitemkv{sort-db}{name=pen, count=3}
\dbshow{sort-style}{sort-db}
```

pen: 3 tag: 2 bag: 1 box: 1

before-code ☆

New: 2022-01-05

`before-code = ⟨before code⟩` (initially empty, no default)

该选项用来设置在展示整个数据库之**前**需要执行的代码（见示例 4）。

after-code ☆

New: 2022-01-05

`after-code = ⟨after code⟩` (initially empty, no default)

该选项用来设置在展示整个数据库之**后**需要执行的代码（见示例 4）。

示例 4：设置展示数据库前后的代码

```
\dbNewDatabase{wrap-db}{text=tl}
\dbNewStyle{wrap-style}{wrap-db}{
  before-code = \textit{before code}\quad,
  after-code  = \textit{after code},
```

```
  item-code    = \dbarabic.~\dbuse{text}\quad
}
\dbitemkv{wrap-db}{text=text1}
\dbitemkv{wrap-db}{text=text2}
\dbitemkv{wrap-db}{text=text3}
\dbshow{wrap-style}{wrap-db}
```

before code 1. text1 2. text2 3. text3 *after code*

item-code ☆

New: 2022-01-05

item-code = ⟨*item code*⟩ (initially unset, no default)

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{⟨*attr*⟩} 来指代属性 ⟨*attr*⟩ 的值。示例 5 演示了如何展示一个首字母缩写词表。

示例 5：展示数据库条目

```
\dbNewDatabase{item-db}{acronym=str, desc=tl}
\dbNewStyle{item-style}{item-db}{
  before-code = {\dbIfEmptyF{\begin{description}}},
  after-code  = {\dbIfEmptyF{\end{description}}},
  item-code   = {\item[\dbuse{acronym}] \dbuse{desc}},
  sort        = acronym,
}
\dbitemkv{item-db}{acronym=PM,  desc={Prime Minister}}
\dbitemkv{item-db}{acronym=CBD, desc={Central Business District}}
\dbitemkv{item-db}{acronym=DL,  desc={Deep Learning}}
\dbshow{item-style}{item-db}
```

CBD Central Business District

DL Deep Learning

PM Prime Minister

item-code* ☆

New: 2022-01-17

item-code* = ⟨*item code*⟩ (initially unset, no default)

使用该选项设置的代码在被插入到最终执行代码序列之前会先通过 \protected@edef 完全展开。示例 6 展示了如何通过该选项使用表格来展示数据。

示例 6：用表格展示数据

```
\dbNewDatabase{tab-db}{name=str, count=int}
\dbNewStyle{tab-style}{tab-db}{
  before-code = {%
    \begin{tabular}{ll}
      name & count \\
  },
  after-code  = \end{tabular},
  item-code*  = {%
    \textcolor{red}{\dbuse{name}} & \dbuse{count} \\
  },
}
\dbitemkv{tab-db}{name=bag, count=100}
\dbitemkv{tab-db}{name=pig, count=20}
```

```
\dbshow{tab-style}{tab-db}
```

name	count
bag	100
pig	20

item-before-code ☆

New: 2022-01-08

Updated: 2022-01-14

item-before-code = ⟨*before code*⟩ (initially empty, no default)

在 ⟨*item code*⟩ 之**前**执行的代码（见示例 7）。

item-after-code ☆

New: 2022-01-08

Updated: 2022-01-14

item-after-code = ⟨*after code*⟩ (initially empty, no default)

在 ⟨*item code*⟩ 之**后**执行的代码（见示例 7）。

示例 7：设置展示条目之前和之后的代码

```
\dbNewDatabase{item-wrap-db}{text=tl, hint=tl}
\dbNewStyle{item-wrap-style}{item-wrap-db}{
  item-before-code = \begingroup\ttfamily<,
  item-after-code  = >\endgroup,
  item-code        = \dbuse{text}~(\dbuse{hint}),
}
\dbitemkv{item-wrap-db}{text=example, hint={this is an example}}
\dbshow{item-wrap-style}{item-wrap-db}
```

```
<example (this is an example)>
```

2.2.2 属性选项

<attr>/code ☆

New: 2022-01-14

⟨*attr*⟩/code = ⟨*code*⟩ (initially `#1`, no default)

设置 ⟨*attr*⟩ 的样式代码。在 ⟨*code*⟩ 中用 `#1` 指代属性的值。示例 8 将数量超过 10 个的物品打印为红色，少于 10 个的则打印为青色。

示例 8：设置单个属性样式

```
\dbNewDatabase{attr-code-db}{name=str, count=int}
\begin{dbFilters}{attr-code-db}
  \dbNewCond{large}{count}{\dbval >= 10}
\end{dbFilters}
\dbNewStyle{base-style}{attr-code-db}{
  item-code  = \dbuse{name}:~\dbuse{count}\quad,
}
\dbNewStyle[base-style]{large-style}{attr-code-db}{
  raw-filter = large,
  name/code  = \textcolor{red}{#1},
}
\dbNewStyle[base-style]{small-style}{attr-code-db}{
```

```
  raw-filter = !large,
  name/code  = \textcolor{teal}{#1},
}
\dbitemkv{attr-code-db}{name=bag, count=1}
\dbitemkv{attr-code-db}{name=pen, count=12}
\dbitemkv{attr-code-db}{name=pig, count=5}
\dbitemkv{attr-code-db}{name=egg, count=50}
\dbshow{large-style}{attr-code-db}
\dbshow{small-style}{attr-code-db}
```

pen: 12 egg: 50 bag: 1 pig: 5

<attr>/code* ☆ New: 2022-01-14

⟨attr⟩/code* = ⟨code⟩ (initially unset, no default)

设置 ⟨attr⟩ 的样式代码。在 ⟨code⟩ 中用 #1 指代**展开**的属性的值。这对某些需要以特定格式解析参数的命令比较有用。

示例 9 中 zhnumber 宏包的 \zhdate 命令接收 yyyy/mm/dd 格式的时期并输出中文日期，本宏包默认的日期输出格式是 yyyy/mm/dd，因此可以通过 date/code* 选项，将日期完全展开后然后传递给 \zhdate 命令，如果不展开，\zhdate 接收到的是若干个用来展示日期的控制序列，而不是 yyyy/mm/dd 格式的时期，进而触发报错。

示例 9：中文日期

```
% \usepackage{zhnumber}
\dbNewDatabase{exp-db}{date=date, event=tl}
\dbNewStyle{exp-style}{exp-db}{
  item-code  = \par\makebox[4cm][l]{\dbuse{date}}\dbuse{event},
  date/code* = \zhdate{#1},
}
\dbitemkv{exp-db}{date=2020/12/31, event=eat}
\dbitemkv{exp-db}{date=2021/01/01, event=sleep}
\dbshow{exp-style}{exp-db}
```

2020 年 12 月 31 日 eat
2021 年 1 月 1 日 sleep

<attr>/before-code ☆ New: 2022-01-05

⟨attr⟩/before-code = ⟨before code⟩ (initially empty, no default)

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 ☆ New: 2022-01-05

⟨attr⟩/after-code = ⟨after code⟩ (initially empty, no default)

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。

属性样式代码的执行顺序为：

1. <attr>/before-code
2. <attr>/code or <attr>/code*
3. <attr>/after-code

<attr>/item-code ☆

⟨attr⟩/item-code = ⟨item code⟩ (initially #1, no default)

New: 2022-01-16

设置列表元素的样式代码。在 ⟨item code⟩ 中用 #1 指代列表元素的值。示例 10 演示了如何为列表元素设置额外的样式。

示例 10：设置列表元素样式代码

```
\dbNewDatabase{clist-db}{name=str, label=clist}
\dbNewStyle{clist-style}{clist-db}{
  item-code       = \par\dbuse{name}:~\dbuse{label},
  label/item-code = (\textcolor{red}{\textit{#1}}),
}
\dbitemkv{clist-db}{name=pig,  label={animal, meat}}
\dbitemkv{clist-db}{name=Alex, label={person, male}}
\dbshow{clist-style}{clist-db}
```

pig: (*animal*), (*meat*)
Alex: (*person*), (*male*)

<attr>/item-code* ☆

⟨attr⟩/item-code* = ⟨item code⟩ (initially unset, no default)

New: 2022-01-16

设置列表元素的样式代码。在 ⟨item code⟩ 中用 #1 指代**展开**的列表元素的值。

<attr>/item-before-code ☆

⟨attr⟩/item-before-code = ⟨code⟩ (initially empty, no default)

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⟩ (initially empty, no default)

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。

列表元素样式代码的执行顺序为：

1. <attr>/item-before-code
2. <attr>/item-code or <attr>/item-code*
3. <attr>/item-after-code

<attr>/sep ☆

⟨attr⟩/sep = ⟨separator⟩ (initially {,~} for clist and / for date, no default)

⟨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⟩
}

New: 2022-01-05

Updated: 2022-01-08

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。示例 11 展示了如何设置列表元素间隔。

示例 11：设置列表元素间隔

```
\dbNewDatabase{clist-db}{label=clist}
\dbNewStyle{clist-base}{clist-db}{
  before-code = {\dbIfEmptyF{\begin{enumerate}}},
  after-code  = {\dbIfEmptyF{\end{enumerate}}},
  item-code   = \item \dbuse{label},
}
\dbNewStyle[clist-base]{clist-style1}{clist-db}{
  label/sep = {{,~}}
}
\dbNewStyle[clist-base]{clist-style2}{clist-db}{
  label/sep = {{,~}, {,~}, ~and~}
}
\dbitemkv{clist-db}{label={a, b, c}}
\dbitemkv{clist-db}{label={1, 2, 3}}
\dbshow{clist-style1}{clist-db}
\dbshow{clist-style2}{clist-db}
```

1. a, b, c
2. 1, 2, 3

1. a, b and c
2. 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。示例 12 展示了如何自定义日期间隔符。

示例 12：设置日期间隔符

```
\dbNewDatabase{date-db}{date=date}
\dbNewStyle{date-style1}{date-db}{
  item-code = \dbuse{date}\quad,
  date/sep  = -,
}
\dbNewStyle{date-style2}{date-db}{
  item-code = \dbuse{date}\quad,
  date/sep  = {\$, +, !, \$},
}
\dbitemkv{date-db}{date=2020/01/02}
\dbitemkv{date-db}{date=2022/07/12}
\dbshow{date-style1}{date-db}
\dbshow{date-style2}{date-db}
```

2020-01-02　2022-07-12　$2020+01!02$　$2022+07!12$

<attr>/format-code ☆

New: 2022-01-14

⟨attr⟩/format-code = ⟨*format code*⟩ (initially unset, no default)

该选项用来更精细地控制日期的输出格式。在 ⟨*format code*⟩中，#1 代表年份，#2 代表月份，#3 代表天。示例 13 演示了如何使用该选项。

示例 13：任意日期格式

```
\dbNewDatabase{date-db}{date=date}
\dbNewStyle{date-style}{date-db}{
  item-code        = \dbuse{date},
  date/format-code = {日：#3\quad 月：#2\quad 年：#1}
}
\dbitemkv{date-db}{date=2022/01/01}
\dbshow{date-style}{date-db}
```

日：1　月：1　年：2022

<attr>/zfill ★

New: 2022-01-08

⟨attr⟩/zfill = ⟨true|*false*⟩ (initially true, default true)

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。示例 14 展示了补零和不补零的日期。

示例 14：月份和天补零

```
\dbNewDatabase{date-db}{date=date}
\dbNewStyle {zfill-style}{date-db}{
  item-code = \dbuse{date},
}
\dbNewStyle{nofill-style}{date-db}{
  item-code  = \dbuse{date},
  date/zfill = false,
}
\dbitemkv{date-db}{date=2022/01/01}
\dbshow {zfill-style}{date-db}
\dbshow{nofill-style}{date-db}
```

2022/01/01 2022/1/1

\dbdatesep

New: 2022-01-13

\dbdatesep {⟨*separator*⟩}

设置内部解析日期时的分隔符，默认为 /，即存储数据的格式为 yyyy/mm/dd。示例 15 演示了使用两种格式存储数据，但实际上分隔符并没有被存储，而是被内部用来解析年、月和日然后存储为三个整数。

示例 15：设置日期解析格式

```
\dbNewDatabase{inner-date-db}{date=date}
\dbNewStyle{inner-date-style}{inner-date-db}{
  item-code = \dbuse{date}\quad,
}
```

```
\dbitemkv{inner-date-db}{date=2020/01/20}
\dbdatesep{-}
\dbitemkv{inner-date-db}{date=2022-01-10}
\dbshow{inner-date-style}{inner-date-db}
```

2020/01/20 2022/01/10

2.3 过滤器

过滤器是一些条件的组合，只有满足过滤器指定条件的数据才会被展现出来。

\dbNewReviewPoints

New: 2022-01-05

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

定义名为 ⟨*name*⟩ 的复习间隔列表。⟨*points*⟩ 是一系列整数或整数表达式，用于设置日期的过滤器。示例 16 中，预定义了一个复习间隔列表 review 和一个时间锚点 2022/02/06，筛选时，将当前条目的日期与描点相比较，计算得到时间间隔 ⟨*interval*⟩ = ⟨*date anchor*⟩ − ⟨*date cmp*⟩，只有当 ⟨*interval*⟩ 的值在 ⟨*points*⟩ 中时，条件才成立。

示例 16：按时间间隔筛选

```
\dbNewDatabase{filter-db}{date=date}
\dbNewReviewPoints{review}{2, 5}
\dbNewRawFilter*{review-filter}{filter-db}{date}{review|2022/02/06}
\dbNewStyle{filter-style}{filter-db}{
  item-code = \dbuse{date}\quad,
  filter    = review-filter,
}
\dbitemkv{filter-db}{date=2022/01/30}
\dbitemkv{filter-db}{date=2022/02/01}
\dbitemkv{filter-db}{date=2022/02/04}
\dbshow{filter-style}{filter-db}
```

2022/02/01 2022/02/04

dbFilters

New: 2022-01-05

Updated: 2022-01-16

```
\begin{dbFilters}   {⟨database⟩}
  ⟨code⟩
\end{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。星号版本在定义条件的同时会定义一个与条件同名且只使用这个条件的过滤器。示例 17 中定义 greater 条件的同时也定义了一个同名的过滤器，你可以直接在 filter 中使用这个过滤器。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。

示例 17：定义与条件同名的过滤器

```
\dbNewDatabase{filter-db}{count=int}
\begin{dbFilters}*{filter-db}
```

```
  \dbNewCond{greater}{count}{\dbval > 3}
\end{dbFilters}
\dbNewStyle{filter-style}{filter-db}{
  filter    = greater,
  item-code = \dbuse{count}\quad,
}
\dbitemkv{filter-db}{count=2}
\dbitemkv{filter-db}{count=5}
\dbshow{filter-style}{filter-db}
```

```
5
```

\dbNewConditional
\dbNewCond
\dbNewConditional*
\dbNewCond*

New: 2022-01-05
Updated: 2022-01-16

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

\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。\dbNewCond 是 \dbNewConditional 的别名。

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。该选项依赖于 l3regex。示例 18 演示了部分匹配和全部匹配的区别，part 过滤器匹配所有含数字的 name，而 all 过滤器匹配全部为数字的 name。

示例 18：匹配字符串

```
\dbNewDatabase{filter-db}{name=str}
\begin{dbFilters}*{filter-db}
  \dbNewCond{part}{name}{\d+}
  \dbNewCond*{all}{name}{\d+}
\end{dbFilters}
\dbNewStyle{part-style}{filter-db}{
  before-code = Match part:~,
  item-code   = \dbuse{name}\quad,
  filter      = part,
}
\dbNewStyle{all-style}{filter-db}{
  before-code = Match all:~,
  item-code   = \dbuse{name}\quad,
  filter      = all,
}
\dbitemkv{filter-db}{name=123}
\dbitemkv{filter-db}{name=int12}
```

```
\dbitemkv{filter-db}{name=variable}
\dbshow{part-style}{filter-db}
\dbshow {all-style}{filter-db}
```

Match part: 123 int12 Match all: 123

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。示例 19 中过滤器 or 匹配含有 hard **或者** red 的标签，而过滤器 and 匹配含有 hard **并且** 含有 red 的标签。

示例 19：筛选列表

```
\dbNewDatabase{filter-db}{label=clist}
\begin{dbFilters}*{filter-db}
  \dbNewCond  {or}{label}{hard, red}
  \dbNewCond*{and}{label}{hard, red}
\end{dbFilters}
\def\emph#1{\textit{\textbf{#1}}}
\dbNewStyle{base-style}{filter-db}{
  before-code = {
    \begin{minipage}[t]{.3\textwidth}
      All items
  },
  after-code  = {\end{minipage}},
  item-code   = \par\dbarabic.~\dbuse{label},
}
\dbNewStyle[base-style] {or-style}{filter-db}{
  before-code = {
    \begin{minipage}[t]{.3\textwidth}
      Match \emph{any} of hard \emph{or} red
  },
  filter      = or,
}
\dbNewStyle[base-style]{and-style}{filter-db}{
  before-code = {
    \begin{minipage}[t]{.3\textwidth}
      Match \emph{all} of hard \emph{and} red
  },
  filter      = and,
}
\dbitemkv{filter-db}{label={hard, red}}
\dbitemkv{filter-db}{label={hard, blue}}
\dbitemkv{filter-db}{label={easy, blue}}
\dbitemkv{filter-db}{label={easy, red}}
\dbitemkv{filter-db}{label={hard, red, flat}}
\dbshow {base-style}{filter-db}
\dbshow   {or-style}{filter-db}
\dbshow  {and-style}{filter-db}
```

All items	Match ***any*** of hard ***or*** red	Match ***all*** of hard ***and*** red
1. hard, red	1. hard, red	1. hard, red
2. hard, blue	2. hard, blue	2. hard, red, flat
3. easy, blue	3. easy, red	
4. easy, red	4. hard, red, flat	
5. hard, red, flat		

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。示例 20 展示了如何使用该选项。

示例 20：根据日期表达式过滤

```
\dbNewDatabase{filter-db}{date=date}
\dbNewRawFilter{date-filter}{filter-db}{date}{\dbval >= 2022/02/01}
\dbNewStyle{filter-style}{filter-db}{
  item-code = \dbuse{date}\quad,
  filter    = date-filter,
}
\dbitemkv{filter-db}{date=2022/01/30}
\dbitemkv{filter-db}{date=2022/02/01}
\dbitemkv{filter-db}{date=2022/02/04}
\dbshow{filter-style}{filter-db}
```

2022/02/01 2022/02/04

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（见示例 16）。

\dbNewRawFilter

New: 2022-01-16

```
\dbNewRawFilter  {⟨name⟩} {⟨database⟩} {⟨attr⟩} {⟨cond spec⟩} [⟨filter info⟩]
\dbNewRawFilter* {⟨name⟩} {⟨database⟩} {⟨attr⟩} {⟨cond spec⟩} [⟨filter info⟩]
```

等同于

```
\begin{dbFilters}*        {⟨database⟩}
  \dbNewCond      {⟨name⟩}              {⟨attr⟩} {⟨cond spec⟩} [⟨filter info⟩]
  \dbNewCond*     {⟨name⟩}              {⟨attr⟩} {⟨cond spec⟩} [⟨filter info⟩]
\end{dbFilters}
```

该命令用来快捷地定义单个过滤器。示例 21 展示了如何使用该命令，它和示例 17 本质上是相同的。

示例 21：定义与条件同名的过滤器

```
\dbNewDatabase{filter-db}{count=int}
\dbNewRawFilter{greater}{filter-db}{count}{\dbval > 3}
\dbNewStyle{filter-style}{filter-db}{
  filter    = greater,
  item-code = \dbuse{count}\quad,
}
\dbitemkv{filter-db}{count=2}
\dbitemkv{filter-db}{count=5}
\dbshow{filter-style}{filter-db}
```

5

\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。⟨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。使用示例见示例 1.

2.4 存储和使用数据

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。示例 22 展示了如何存储数据。

示例 22：存储数据

```
\dbNewDatabase{ques-db}{date=date, ques=tl, ans=tl}
\dbNewStyle{ques-style}{ques-db}{
  item-code       = {%
    \par\dbuse{date}
    \par\dbarabic.~\dbuse{ques}
    \par\textbf{答案：}~\dbuse{ans}
  },
  item-after-code = \medskip,
  date/code*      = \zhdate{#1},
}
\begin{dbitem}{ques-db}[date=2022/01/01]
  \dbsave{ques}{地球到月亮的距离}
  \dbsave{ans} {384,401 公里}
\end{dbitem}
\begin{dbitem}{ques-db}[date=2022/01/02]
  \dbsave{ques}{鲁迅的本名}
  \dbsave{ans} {周树人}
\end{dbitem}
\dbshow{ques-style}{ques-db}
```

2022 年 1 月 1 日
1. 地球到月亮的距离
答案： 384,401 公里

2022 年 1 月 2 日
2. 鲁迅的本名
答案： 周树人

\dbitemkv

New: 2022-01-13

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

只使用 ⟨attr-val list⟩ 来存储数据。

\dbsave
\dbsave*

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

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

\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 ⋆

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

\dbuse {⟨attr⟩}

\dbuse 用来展示数据，只能在 item-code, item-before-code, item-after-code 选项中使用。\dbuse 是可展开的。

2.5 条件判别式

\dbIfEmptyT ⋆
\dbIfEmptyF ⋆
\dbIfEmptyTF ⋆

New: 2022-01-05

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

该判别式用来判断当前数据库是否为空。示例 23 演示了如何使用该判别式来预防空的列表环境。

示例 23：预防空列表环境

```
\dbNewDatabase{test-db}{text=tl}
\dbNewRawFilter{alph}{test-db}{text}{\d}
\dbNewStyle{base-style}{test-db}{
  before-code = \dbIfEmptyTF{Empty db}{\begin{enumerate}},
  after-code  = \dbIfEmptyF{\end{enumerate}}\medskip,
  item-code   = \item \dbuse{text},
}
\dbNewStyle[base-style]{empty-style}{test-db}{
  raw-filter=!alph
}
\dbitemkv{test-db}{text={$1 + 1 = 2$.}}
\dbitemkv{test-db}{text={I have 2 pens.}}
\dbshow {base-style}{test-db}
\dbshow{empty-style}{test-db}
```

1. $1 + 1 = 2$.
2. I have 2 pens.

Empty db

\dbIfLastT ⋆
\dbIfLastF ⋆
\dbIfLastTF ⋆

New: 2022-01-17

\dbIfLastTF {⟨true code⟩} {⟨false code⟩}
\dbIfLastT {⟨true code⟩}
\dbIfLastF {⟨false code⟩}

该判别式用来判断当前是否为数据库要展示的最后一条数据。示例 24 演示了如何设置条目之间的间隔。

示例 24：设置条目之间的间隔

```
\dbNewDatabase{last-db}{text=tl}
\dbNewStyle{last-style}{last-db}{
  item-code  = {%
    \par\dbuse{text}\par%
    \dbIfLastF{\textcolor{red}{\hrulefill separator\hrulefill}}%
  },
}
\dbitemkv{last-db}{text=This is the first paragraph.}
\dbitemkv{last-db}{text=This is the second paragraph.}
\dbitemkv{last-db}{text=This is the last paragraph.}
\dbshow{last-style}{last-db}
```

This is the first paragraph.

separator

This is the second paragraph.

separator

This is the last paragraph.

2.6 表达式函数

\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 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。示例 25 展示了如何筛选 3 的倍数。

示例 25：筛选 3 的倍数

```
\dbNewDatabase{expr-db}{n=int}
\dbNewRawFilter{mod}{expr-db}{n}{\dbIntMod{\dbval}{3} = 0}
\dbNewStyle{expr-style}{expr-db}{
  item-code = \dbuse{n}\quad,
  filter    = mod,
```

```
}
\dbitemkv{expr-db}{n=2}
\dbitemkv{expr-db}{n=3}
\dbitemkv{expr-db}{n=6}
\dbitemkv{expr-db}{n=7}
\dbshow{expr-style}{expr-db}
```

3 6

2.7 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 ⋆
\dbtoday ⋆
\dbDatabase ⋆
\dbFilterName ⋆
\dbFilterInfo ⋆
\dbIndex ⋆
\dbarabic ⋆
\dbalph ⋆
\dbAlph ⋆
\dbroman ⋆
\dbRoman ⋆

New: 2022-01-05

\dbval	当前属性的值
\dbtoday	当天的日期
\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 \dbuse{⟨id⟩}
\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

\dbtoday 使用 \dbdatesep 确定的分隔符来展示日期。见示例 ??.

示例 26：特殊命令

```
\dbNewDatabase{special-db}{name=str}
\dbNewRawFilter*{number}{special-db}{name}{\d+}[name that is a number]
\dbNewStyle{special-style}{special-db}{
  before-code = {
    Date: \dbtoday \\
    Database: \dbDatabase
    Filter: \dbFilterName \\
    Filter info: \dbFilterInfo \par
    \begin{tabular}{@{}lllllll}
      Index & arabic & alph & Alph & roman & Roman & value \\
  },
  after-code  = \end{tabular},
  item-code*  = {%
    \dbIndex & \dbarabic & \dbalph & \dbAlph &
    \dbroman & \dbRoman & \dbuse{name} \\
  },
  sort        = name,
```

```
  filter      = number,
}
\dbitemkv{special-db}{name=test}
\dbitemkv{special-db}{name=12}
\dbitemkv{special-db}{name=int2}
\dbitemkv{special-db}{name=99}
\dbshow{special-style}{special-db}
```

Date: 2022/1/17

Database: special-dbFilter: number

Filter info: name that is a number

Index	arabic	alph	Alph	roman	Roman	value
2	1	a	A	i	I	12
4	2	b	B	ii	II	99

Package dbshow v1.5*

⟹ 中文版本

Changkai Li <lichangkai225@qq.com>

2022/01/17

3 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, 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.

3.1 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 Interfaces

4.1 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.

\dbshow
New: 2022-01-05

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

Show the ⟨*database*⟩ with ⟨*style*⟩.

\dbclear
New: 2022-01-07

```
\dbclear {⟨database⟩}
```

Clear the content of ⟨*database*⟩.

4.2 \dbNewStyle and Style Options

\dbNewStyle
New: 2022-01-05
Updated: 2022-01-15

```
\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}{}`.

4.2.1 General Options

filter
New: 2022-01-05

`filter = ⟨filter⟩` (initially `-none-`, no default)

Set the ⟨*filter*⟩ defined by \dbCombineFilters. Example 1 shows how to define conditionals and combine them into a filter.

Example 1: Filter Items

```
\dbNewDatabase{filter-db}{name=str, count=int}
\begin{dbFilters}{filter-db}
  \dbNewCond {cond1}{count}{\dbval > 3}
  \dbNewCond*{cond2}{name} {\d+}
  \dbCombCond{filter-and}{cond1 && cond2}
  \dbCombCond{filter-or} {cond1 || cond2}
\end{dbFilters}
\dbitemkv{filter-db}{name=123, count=4}
\dbitemkv{filter-db}{name=ab3, count=2}
\dbitemkv{filter-db}{name=bag, count=5}
\dbNewStyle{filter-and-style}{filter-db}{
  filter      = filter-and,
  before-code = \par Filter And\par,
  item-code   = {\dbuse{name}: \dbuse{count}\quad},
}
\dbNewStyle{filter-or-style}{filter-db}{
  filter      = filter-or,
  before-code = \par Filter Or\par,
  item-code   = {\dbuse{name}: \dbuse{count}\quad},
}
\dbshow{filter-and-style}{filter-db}
\dbshow{filter-or-style} {filter-db}
```

Filter And
123: 4
Filter Or
123: 4 bag: 5

raw-filter

New: 2022-01-06

`raw-filter` = ⟨*conditional expression*⟩ (initially unset, no default)

Set anonymous filter with conditionals defined by `\dbNewConditional`. Example 2 shows how to simplify the code of example 1 with `raw-filter` option.

Example 2: Using Anonymous Filter

```
\dbNewDatabase{filter-db}{name=str, count=int}
\begin{dbFilters}{filter-db}
  \dbNewCond {cond1}{count}{\dbval > 3}
  \dbNewCond*{cond2}{name} {\d+}
\end{dbFilters}
\dbitemkv{filter-db}{name=123, count=4}
\dbitemkv{filter-db}{name=ab3, count=2}
\dbitemkv{filter-db}{name=bag, count=5}
\dbNewStyle{filter-and-style}{filter-db}{
  raw-filter  = {cond1 && cond2},
  before-code = \par Filter And\par,
  item-code   = {\dbuse{name}: \dbuse{count}\quad},
}
\dbNewStyle{filter-or-style}{filter-db}{
  raw-filter  = {cond1 || cond2},
  before-code = \par Filter Or\par,
```

```
  item-code    = {\dbuse{name}: \dbuse{count}\quad},
}
\dbshow{filter-and-style}{filter-db}
\dbshow{filter-or-style} {filter-db}
```

Filter And
123: 4
Filter Or
123: 4 bag: 5

sort

New: 2022-01-05

`sort = { ⟨attr spec1⟩, ⟨attr spec2⟩, … }` (initially unset, no default)

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. Example 3 shows how to sort items by `count` in descending order and for the same `count`, sort by `name` in ascending order.

Example 3: Multi-level Sorting

```
\dbNewDatabase{sort-db}{name=str, count=int}
\dbNewStyle{sort-style}{sort-db}{
  sort = {count*, name},
  item-code = {\dbuse{name}: \dbuse{count}\quad}
}
\dbitemkv{sort-db}{name=bag, count=1}
\dbitemkv{sort-db}{name=box, count=1}
\dbitemkv{sort-db}{name=tag, count=2}
\dbitemkv{sort-db}{name=pen, count=3}
\dbshow{sort-style}{sort-db}
```

pen: 3 tag: 2 bag: 1 box: 1

before-code ☆

New: 2022-01-05

`before-code = ⟨before code⟩` (initially empty, no default)

Set the ⟨*code*⟩ that is executed **before** displaying the database. See Example 4.

after-code ☆

New: 2022-01-05

`after-code = ⟨after code⟩` (initially empty, no default)

Set the ⟨*code*⟩ that is executed **after** displaying the database. See Example 4.

Example 4: Set the Before and After Code of the Database

```
\dbNewDatabase{wrap-db}{text=tl}
\dbNewStyle{wrap-style}{wrap-db}{
  before-code = \textit{before code}\quad,
  after-code  = \textit{after code},
  item-code   = \dbarabic.~\dbuse{text}\quad
}
\dbitemkv{wrap-db}{text=text1}
\dbitemkv{wrap-db}{text=text2}
\dbitemkv{wrap-db}{text=text3}
\dbshow{wrap-style}{wrap-db}
```

before code 1. text1 2. text2 3. text3 *after code*

item-code ☆

New: 2022-01-05

item-code = ⟨*item code*⟩ (initially unset, no default)

Set the code that display a record. You can use \dbuse to denote the value of attribute. Example 5 shows how to display an acronym glossary table.

Example 5: Display Database Items

```
\dbNewDatabase{item-db}{acronym=str, desc=tl}
\dbNewStyle{item-style}{item-db}{
  before-code = {\dbIfEmptyF{\begin{description}}},
  after-code  = {\dbIfEmptyF{\end{description}}},
  item-code   = {\item[\dbuse{acronym}] \dbuse{desc}},
  sort        = acronym,
}
\dbitemkv{item-db}{acronym=PM,  desc={Prime Minister}}
\dbitemkv{item-db}{acronym=CBD, desc={Central Business District}}
\dbitemkv{item-db}{acronym=DL,  desc={Deep Learning}}
\dbshow{item-style}{item-db}
```

CBD Central Business District

DL Deep Learning

PM Prime Minister

item-code* ☆

New: 2022-01-17

item-code* = ⟨*item code*⟩ (initially unset, no default)

The ⟨*item code*⟩ will be expanded through \protected@edef before it is used. 6 shows how to display data with table using the expanded code.

Example 6: Display Data with Table

```
\dbNewDatabase{tab-db}{name=str, count=int}
\dbNewStyle{tab-style}{tab-db}{
  before-code = {%
    \begin{tabular}{ll}
      name & count \\
  },
  after-code  = \end{tabular},
  item-code*  = {%
    \textcolor{red}{\dbuse{name}} & \dbuse{count} \\
  },
}
\dbitemkv{tab-db}{name=bag, count=100}
\dbitemkv{tab-db}{name=pig, count=20}
\dbshow{tab-style}{tab-db}
```

name	count
bag	100
pig	20

item-before-code ☆

New: 2022-01-08

Updated: 2022-01-14

item-before-code = ⟨*before code*⟩ (initially empty, no default)

Set the ⟨*code*⟩ that is executed **before** displaying a item. See Example 7.

item-after-code ☆

New: 2022-01-08

Updated: 2022-01-14

item-after-code = ⟨*after code*⟩ (initially empty, no default)

Set the ⟨*code*⟩ that is executed **after** displaying the item. See Example 7.

Example 7: Set the Before and After Code of the Item

```
\dbNewDatabase{item-wrap-db}{text=tl}
\dbNewStyle{item-wrap-style}{item-wrap-db}{
  item-before-code = \begingroup\ttfamily<,
  item-after-code  = >\endgroup,
  item-code        = \dbuse{text},
}
\dbitemkv{item-wrap-db}{text=example}
\dbshow{item-wrap-style}{item-wrap-db}
```

```
<example>
```

4.2.2 Attribute Options

<attr>/code ☆

New: 2022-01-14

⟨*attr*⟩/code = ⟨*code*⟩ (initially `#1`, no default)

Set the style code of ⟨*attr*⟩. In ⟨*code*⟩, `#1` is replaced with the value of ⟨*attr*⟩. Example 8 prints `name` whose `count` is less than 10 in teal color, otherwise in red color.

Example 8: Set Style Code of Attribute

```
\dbNewDatabase{attr-code-db}{name=str, count=int}
\begin{dbFilters}{attr-code-db}
  \dbNewCond{large}{count}{\dbval >= 10}
\end{dbFilters}
\dbNewStyle{base-style}{attr-code-db}{
  item-code  = \dbuse{name}:~\dbuse{count}\quad,
}
\dbNewStyle[base-style]{large-style}{attr-code-db}{
  raw-filter = large,
  name/code  = \textcolor{red}{#1},
}
\dbNewStyle[base-style]{small-style}{attr-code-db}{
  raw-filter = !large,
  name/code  = \textcolor{teal}{#1},
}
\dbitemkv{attr-code-db}{name=bag, count=1}
\dbitemkv{attr-code-db}{name=pen, count=12}
\dbitemkv{attr-code-db}{name=pig, count=5}
\dbitemkv{attr-code-db}{name=egg, count=50}
\dbshow{large-style}{attr-code-db}
\dbshow{small-style}{attr-code-db}
```

pen: 12 egg: 50 bag: 1 pig: 5

<attr>/code* ☆ ⟨attr⟩/code* = ⟨code⟩ (initially unset, no default)

New: 2022-01-14

Set the style code of ⟨attr⟩. In ⟨code⟩, #1 is replaced with the **expanded** value of ⟨attr⟩. This is useful to commands that require special format of the arguments.

Example 9 shows how to print Chinese date with \zhdate of package zhnumber. \zhdate require argument to be the date of yyyy/mm/dd, which is the default date format of dbshow. But we need to expand the date through date/code* first because \zhdate cannot parse the token list other than yyyy/mm/dd.

Example 9: Show Chinese Date

```
% \usepackage{zhnumber}
\dbNewDatabase{exp-db}{date=date, event=tl}
\dbNewStyle{exp-style}{exp-db}{
  item-code  = \par\makebox[4cm][l]{\dbuse{date}}\dbuse{event},
  date/code* = \zhdate{#1},
}
\dbitemkv{exp-db}{date=2020/12/31, event=eat}
\dbitemkv{exp-db}{date=2021/01/01, event=sleep}
\dbshow{exp-style}{exp-db}
```

2020 年 12 月 31 日 eat
2021 年 1 月 1 日 sleep

<attr>/before-code ☆ ⟨attr⟩/before-code = ⟨before code⟩ (initially empty, no default)

New: 2022-01-05

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

<attr>/after-code ☆ ⟨attr⟩/after-code = ⟨after code⟩ (initially empty, no default)

New: 2022-01-05

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

The style code execution order of attribute is:

1. <attr>/before-code
2. <attr>/code or <attr>/code*
3. <attr>/after-code

<attr>/item-code ☆ ⟨attr⟩/item-code = ⟨item code⟩ (initially #1, no default)

New: 2022-01-16

Set the style code of ⟨attr⟩. In ⟨item code⟩, #1 is replaced with the item of the comma list. Example 10 shows how to set the style code of the item of the comma list.

Example 10: Set the Style Code of Clist Item

```
\dbNewDatabase{clist-db}{name=str, label=clist}
\dbNewStyle{clist-style}{clist-db}{
  item-code       = \par\dbuse{name}:~\dbuse{label},
  label/item-code = (\textcolor{red}{\textit{#1}}),
```

```
}
\dbitemkv{clist-db}{name=pig,  label={animal, meat}}
\dbitemkv{clist-db}{name=Alex, label={person, male}}
\dbshow{clist-style}{clist-db}
```

pig: (*animal*), (*meat*)
Alex: (*person*), (*male*)

<attr>/item-code* ☆

New: 2022-01-16

⟨attr⟩/item-code* = ⟨*item code*⟩ (initially unset, no default)

Set the style code of ⟨*attr*⟩. In ⟨*item code*⟩, `#1` is replaced with the **expanded** item of the comma list.

<attr>/item-before-code ☆

New: 2022-01-05

⟨attr⟩/item-before-code = ⟨*before code*⟩ (initially empty, no default)

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

<attr>/item-after-code ☆

New: 2022-01-05

⟨attr⟩/item-after-code = ⟨*after code*⟩ (initially empty, no default)

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

The style code execution order of comma list item is:

1. <attr>/item-before-code
2. <attr>/item-code or <attr>/item-code*
3. <attr>/item-after-code

<attr>/sep ☆

New: 2022-01-05

Updated: 2022-01-08

⟨attr⟩/sep = ⟨*separator*⟩ (initially `{,~}` for `clist` and `/` for `date`, no default)

```
⟨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. Example 11 shows how to use this option to customize the separators between comma list items.

Example 11: Set Separator Between Items of Comma List

```
\dbNewDatabase{clist-db}{label=clist}
\dbNewStyle{clist-base}{clist-db}{
  before-code = {\dbIfEmptyF{\begin{enumerate}}},
  after-code  = {\dbIfEmptyF{\end{enumerate}}},
  item-code   = \item \dbuse{label},
}
\dbNewStyle[clist-base]{clist-style1}{clist-db}{
  label/sep = {{,~}}
}
\dbNewStyle[clist-base]{clist-style2}{clist-db}{
  label/sep = {{,~}, {,~}, ~and~}
}
\dbitemkv{clist-db}{label={a, b, c}}
\dbitemkv{clist-db}{label={1, 2, 3}}
\dbshow{clist-style1}{clist-db}
\dbshow{clist-style2}{clist-db}
```

1. a, b, c
2. 1, 2, 3

1. a, b and c
2. 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. Example 12 shows how to customize the date separators.

Example 12: Set Date Separators

```
\dbNewDatabase{date-db}{date=date}
\dbNewStyle{date-style1}{date-db}{
  item-code = \dbuse{date}\quad,
  date/sep  = -,
}
\dbNewStyle{date-style2}{date-db}{
  item-code = \dbuse{date}\quad,
  date/sep  = {\$, +, !, \$},
}
\dbitemkv{date-db}{date=2020/01/02}
\dbitemkv{date-db}{date=2022/07/12}
\dbshow{date-style1}{date-db}
\dbshow{date-style2}{date-db}
```

2020-01-02 2022-07-12 $2020+01!02$ $2022+07!12$

<attr>/format-code ☆

New: 2022-01-14

⟨attr⟩/format-code = ⟨*format code*⟩ (initially unset, no default)

Use this option to get fine-grained control over date formatting. In ⟨*format code*⟩, #1 represents for the ⟨*year*⟩, #2 represents for the ⟨*month*⟩ and #3 represents for the ⟨*day*⟩. Example 13 shows how to format the date with this option.

Example 13: Date Formatting

```
\dbNewDatabase{date-db}{date=date}
\dbNewStyle{date-style}{date-db}{
  item-code        = \dbuse{date},
  date/format-code = {日: #3\quad 月: #2\quad 年: #1}
}
\dbitemkv{date-db}{date=2022/01/01}
\dbshow{date-style}{date-db}
```

日: 1 月: 1 年: 2022

<attr>/zfill ★

New: 2022-01-08

⟨attr⟩/zfill = ⟨true|*false*⟩ (initially true, default true)

Only for attributes of type date. Control whether to fill zero on the left of the month or day. Example 14 shows the differences.

Example 14: Control the Leading Zero of the Date

```
\dbNewDatabase{date-db}{date=date}
\dbNewStyle {zfill-style}{date-db}{
  item-code = \dbuse{date},
}
\dbNewStyle{nofill-style}{date-db}{
  item-code  = \dbuse{date},
  date/zfill = false,
}
\dbitemkv{date-db}{date=2022/01/01}
\dbshow {zfill-style}{date-db}
\dbshow{nofill-style}{date-db}
```

2022/01/01 2022/1/1

\dbdatesep

New: 2022-01-13

\dbdatesep {⟨*separator*⟩}

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. Example 15 shows how to store dates with two formats. However, the separators are not really stored but used to parse the year, month and day, which will be stored internally as three integers.

Example 15: Set Date Parsing Format

```
\dbNewDatabase{inner-date-db}{date=date}
\dbNewStyle{inner-date-style}{inner-date-db}{
  item-code = \dbuse{date}\quad,
}
\dbitemkv{inner-date-db}{date=2020/01/20}
\dbdatesep{-}
```

```
\dbitemkv{inner-date-db}{date=2022-01-10}
\dbshow{inner-date-style}{inner-date-db}
```

2020/01/20 2022/01/10

4.3 Data Filters

Filter is a combination of conditionals that is used to filter the data you want to display.

\dbNewReviewPoints

New: 2022-01-05

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

Define ⟨*points*⟩ to filter dates by intervals. something. ⟨*points*⟩ is a list of ⟨*intexpr*⟩. In example 16, a ⟨*date anchor*⟩ and a list of ⟨*points*⟩ is defined. During the filtering process, ⟨*interval*⟩ is calculated by ⟨*interval*⟩ = ⟨*date anchor*⟩ − ⟨*date cmp*⟩, ⟨*date cmp*⟩ is the ⟨*date*⟩ of current item. Then each ⟨*intexpr*⟩ in ⟨*points*⟩ is compared with ⟨*interval*⟩ to test if they are equal.

Example 16: Filter with Review Points

```
\dbNewDatabase{filter-db}{date=date}
\dbNewReviewPoints{review}{2, 5}
\dbNewRawFilter*{review-filter}{filter-db}{date}{review|2022/02/06}
\dbNewStyle{filter-style}{filter-db}{
  item-code = \dbuse{date}\quad,
  filter    = review-filter,
}
\dbitemkv{filter-db}{date=2022/01/30}
\dbitemkv{filter-db}{date=2022/02/01}
\dbitemkv{filter-db}{date=2022/02/04}
\dbshow{filter-style}{filter-db}
```

2022/02/01 2022/02/04

dbFilters

New: 2022-01-05

Updated: 2022-01-16

```
\begin{dbFilters}   {⟨database⟩}
  ⟨code⟩
\end{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. Starred version will define a filter of the same name of conditional as soon as you define it. In example 17, line 3 define the conditional and the filter named `greater` so that you can use it in option `option`. Filters are independent in different databases, which means the same name of filters is allowed in different databases.

Example 17: Define Conditional and Filter at Same Time

```
\dbNewDatabase{filter-db}{count=int}
\begin{dbFilters}*{filter-db}
  \dbNewCond{greater}{count}{\dbval > 3}
```

```
\end{dbFilters}
\dbNewStyle{filter-style}{filter-db}{
  filter    = greater,
  item-code = \dbuse{count}\quad,
}
\dbitemkv{filter-db}{count=2}
\dbitemkv{filter-db}{count=5}
\dbshow{filter-style}{filter-db}
```

5

\dbNewConditional
\dbNewCond
\dbNewConditional*
\dbNewCond*

New: 2022-01-05

Updated: 2022-01-16

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

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*⟩, which is shown in example 18: filter `part` match `names` that contain numbers and filter `all` match `names` that is composed of digits.

Example 18: Match Strings

```
\dbNewDatabase{filter-db}{name=str}
\begin{dbFilters}*{filter-db}
  \dbNewCond{part}{name}{\d+}
  \dbNewCond*{all}{name}{\d+}
\end{dbFilters}
\dbNewStyle{part-style}{filter-db}{
  before-code = Match part:~,
  item-code   = \dbuse{name}\quad,
  filter      = part,
}
\dbNewStyle{all-style}{filter-db}{
  before-code = Match all:~,
  item-code   = \dbuse{name}\quad,
  filter      = all,
}
\dbitemkv{filter-db}{name=123}
\dbitemkv{filter-db}{name=int12}
```

```
\dbitemkv{filter-db}{name=variable}
\dbshow{part-style}{filter-db}
\dbshow {all-style}{filter-db}
```

Match part: 123 int12 Match all: 123

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. In example 19, filter `or` match labels that contain hard **or** red, and filter `and` match labels that contain hard **and** red.

Example 19: Filter Lists

```
\dbNewDatabase{filter-db}{label=clist}
\begin{dbFilters}*{filter-db}
  \dbNewCond  {or}{label}{hard, red}
  \dbNewCond*{and}{label}{hard, red}
\end{dbFilters}
\def\emph#1{\textit{\textbf{#1}}}
\dbNewStyle{base-style}{filter-db}{
  before-code = {
    \begin{minipage}[t]{.3\textwidth}
      All items
  },
  after-code  = {\end{minipage}},
  item-code   = \par\dbarabic.~\dbuse{label},
}
\dbNewStyle[base-style] {or-style}{filter-db}{
  before-code = {
    \begin{minipage}[t]{.3\textwidth}
      Match \emph{any} of hard \emph{or} red
  },
  filter      = or,
}
\dbNewStyle[base-style]{and-style}{filter-db}{
  before-code = {
    \begin{minipage}[t]{.3\textwidth}
      Match \emph{all} of hard \emph{and} red
  },
  filter      = and,
}
\dbitemkv{filter-db}{label={hard, red}}
\dbitemkv{filter-db}{label={hard, blue}}
\dbitemkv{filter-db}{label={easy, blue}}
\dbitemkv{filter-db}{label={easy, red}}
\dbitemkv{filter-db}{label={hard, red, flat}}
\dbshow {base-style}{filter-db}
\dbshow   {or-style}{filter-db}
\dbshow  {and-style}{filter-db}
```

All items
1. hard, red
2. hard, blue
3. easy, blue
4. easy, red
5. hard, red, flat

Match ***any*** of hard ***or*** red
1. hard, red
2. hard, blue
3. easy, red
4. hard, red, flat

Match ***all*** of hard ***and*** red
1. hard, red
2. hard, red, flat

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`. Example 20 shows how to use the ⟨*date expr*⟩ to filter the data.

Example 20: Filter with Date Expression

```
\dbNewDatabase{filter-db}{date=date}
\dbNewRawFilter{date-filter}{filter-db}{date}{\dbval >= 2022/02/01}
\dbNewStyle{filter-style}{filter-db}{
  item-code = \dbuse{date}\quad,
  filter    = date-filter,
}
\dbitemkv{filter-db}{date=2022/01/30}
\dbitemkv{filter-db}{date=2022/02/01}
\dbitemkv{filter-db}{date=2022/02/04}
\dbshow{filter-style}{filter-db}
```

2022/02/01 2022/02/04

Starred form defines the conditional with review points defined by `\dbNewReviewPoints` and ⟨*date*⟩ is the date to be compared (see example 16).

`\dbNewRawFilter`

New: 2022-01-16

```
\dbNewRawFilter  {⟨name⟩} {⟨database⟩} {⟨attr⟩} {⟨cond spec⟩} [⟨filter info⟩]
\dbNewRawFilter* {⟨name⟩} {⟨database⟩} {⟨attr⟩} {⟨cond spec⟩} [⟨filter info⟩]
```

Equal to

```
\begin{dbFilters}*        {⟨database⟩}
  \dbNewCond      {⟨name⟩}              {⟨attr⟩} {⟨cond spec⟩} [⟨filter info⟩]
  \dbNewCond*     {⟨name⟩}              {⟨attr⟩} {⟨cond spec⟩} [⟨filter info⟩]
\end{dbFilters}
```

Use this command to quickly define a ⟨*filter*⟩ that has the same name with the ⟨*conditional*⟩. Example 21 is actually the same with example 17, but it is more concise.

Example 21: Define Conditional and Filter at Same Time

```
\dbNewDatabase{filter-db}{count=int}
\dbNewRawFilter{greater}{filter-db}{count}{\dbval > 3}
\dbNewStyle{filter-style}{filter-db}{
  filter    = greater,
  item-code = \dbuse{count}\quad,
}
\dbitemkv{filter-db}{count=2}
\dbitemkv{filter-db}{count=5}
\dbshow{filter-style}{filter-db}
```

5

\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`. Supported operators are `&&`, `||`, `!`. You can set the option `filter` to ⟨*name*⟩ to apply the filter when you display the database. See example 1.

4.4 Store and Use Data

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. Example 22 shows how to store data with `dbitem`.

Example 22: Store Date

```
\dbNewDatabase{ques-db}{date=date, ques=tl, ans=tl}
\dbNewStyle{ques-style}{ques-db}{
  item-code       = {%
    \par\dbuse{date}
    \par\dbarabic.~\dbuse{ques}
    \par\textbf{Answer:}~\dbuse{ans}
  },
  item-after-code = \medskip,
}
\begin{dbitem}{ques-db}[date=2022/01/01]
  \dbsave{ques}{Distance from earth to moon?}
  \dbsave{ans} {384,401 kilometers.}
\end{dbitem}
\begin{dbitem}{ques-db}[date=2022/01/02]
  \dbsave{ques}{The number of English letters?}
  \dbsave{ans} {26.}
\end{dbitem}
\dbshow{ques-style}{ques-db}
```

2022/01/01

1. Distance from earth to moon?

Answer: 384,401 kilometers.

2022/01/02

2. The number of English letters?

Answer: 26.

\dbitemkv

New: 2022-01-13

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

Store data with ⟨*attr-val list*⟩.

4.5 \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, item-before-code, item-after-code.

4.6 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. Example 23 shows how to avoid an empty list environment.

Example 23: Avoid Empty List Environment

```
\dbNewDatabase{test-db}{text=tl}
\dbNewRawFilter{alph}{test-db}{text}{\d}
\dbNewStyle{base-style}{test-db}{
  before-code = \dbIfEmptyTF{Empty db}{\begin{enumerate}},
  after-code  = \dbIfEmptyF{\end{enumerate}}\medskip,
  item-code   = \item \dbuse{text},
}
\dbNewStyle[base-style]{empty-style}{test-db}{
  raw-filter=!alph
}
\dbitemkv{test-db}{text={$1 + 1 = 2$.}}
\dbitemkv{test-db}{text={I have 2 pens.}}
\dbshow {base-style}{test-db}
\dbshow{empty-style}{test-db}
```

1. $1 + 1 = 2$.
2. I have 2 pens.

Empty db

\dbIfLastT ⋆
\dbIfLastF ⋆
\dbIfLastTF ⋆

New: 2022-01-17

\dbIfLastTF {⟨true code⟩} {⟨false code⟩}
\dbIfLastT {⟨true code⟩}
\dbIfLastF {⟨false code⟩}

Test if the current item is the last item to display. Example 24 shows how to set the separator between items.

Example 24: Separator between Items

```
\dbNewDatabase{last-db}{text=tl}
\dbNewStyle{last-style}{last-db}{
  item-code  = {%
    \par\dbuse{text}\par%
    \dbIfLastF{\textcolor{red}{\hrulefill separator\hrulefill}}%
  },
}
\dbitemkv{last-db}{text=This is the first paragraph.}
\dbitemkv{last-db}{text=This is the second paragraph.}
\dbitemkv{last-db}{text=This is the last paragraph.}
\dbshow{last-style}{last-db}
```

This is the first paragraph.

separator

This is the second paragraph.

separator

This is the last paragraph.

4.7 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 equal to \int_abs:n

\dbIntSign is equal to \int_sign:n

\dbIntDivRound is equal to \int_div_round:nn

\dbIntDivTruncate is equal to \int_div_truncate:nn

\dbIntMax is equal to \int_max:nn

\dbIntMin is equal to \int_min:nn

\dbIntMod is equal to \int_mod:nn

\dbFpSign is equal to \fp_sign:n

See detailed documentation of interface3. Example 25 shows how to filter multiples of 3.

Example 25: Filter Multiples of 3

```
\dbNewDatabase{expr-db}{n=int}
\dbNewRawFilter{mod}{expr-db}{n}{\dbIntMod{\dbval}{3} = 0}
\dbNewStyle{expr-style}{expr-db}{
  item-code = \dbuse{n}\quad,
  filter    = mod,
```

```
}
\dbitemkv{expr-db}{n=2}
\dbitemkv{expr-db}{n=3}
\dbitemkv{expr-db}{n=6}
\dbitemkv{expr-db}{n=7}
\dbshow{expr-style}{expr-db}
```

```
3  6
```

4.8 Special Macros

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

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

New: 2022-01-05

\dbval	Attribute value, only accessible in \dbNewConditional.
\dbtoday	Date of today.
\dbDatabase	Database name.
\dbFilterName	Filter name.
\dbFilterInfo	Filter information.
\dbIndex	Record index, identical to \dbuse{⟨*id*⟩}
\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.

\dbtoday show the current date with the separator defined by \dbdatesep. See example 26.

Example 26: Special Commands

```
\dbNewDatabase{special-db}{name=str}
\dbNewRawFilter*{number}{special-db}{name}{\d+}[name that is a number]
\dbNewStyle{special-style}{special-db}{
  before-code = {
    Date: \dbtoday \\
    Database: \dbDatabase
    Filter: \dbFilterName \\
    Filter info: \dbFilterInfo \par
    \begin{tabular}{@{}lllllll}
      Index & arabic & alph & Alph & roman & Roman & value \\
  },
  after-code  = \end{tabular},
  item-code*  = {%
    \dbIndex & \dbarabic & \dbalph & \dbAlph &
    \dbroman & \dbRoman & \dbuse{name} \\
  },
  sort        = name,
```

```
  filter      = number,
}
\dbitemkv{special-db}{name=test}
\dbitemkv{special-db}{name=12}
\dbitemkv{special-db}{name=int2}
\dbitemkv{special-db}{name=99}
\dbshow{special-style}{special-db}
```

Date: 2022/1/17

Database: special-dbFilter: number

Filter info: name that is a number

Index	arabic	alph	Alph	roman	Roman	value
2	1	a	A	i	I	12
4	2	b	B	ii	II	99

5 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 }
```

5.1 Variants and Variables

```
\cs_generate_variant:Nn \msg_warning:nnnn      { nnnx }
\cs_generate_variant:Nn \keys_set:nn           { nv }
\cs_generate_variant:Nn \tl_put_right: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`.)

5.2 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_\tl_to_str:n {#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_\tl_to_str:n {#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_\tl_to_str:n {#1}_\tl_to_str:n {#2} }
    { \msg_fatal: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_\tl_to_str:n {#1}_\tl_to_str:n {#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_\tl_to_str:n {#1}_\tl_to_str:n {#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_fatal:nnn`
`\__dbshow_sep_fatal:xxx`

Check the value of `<attr>/sep` is valid.

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

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

(*End definition for* `\__dbshow_sep_fatal: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\}.
}
```

5.3 Create Database

`\__dbshow_process_default_value:w`

Create map from ⟨*attr*⟩ to ⟨*type*⟩ and map from ⟨*attr*⟩ to ⟨*default value*⟩. Note that only one expansion is needed to get the correct default value from `\l__dbshow_tmp_default`.

#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:cno { 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* 22.)

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

`\__dbshow_get_type:nV`

#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* 22.)

5.4 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 35.*)

\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 35.*)

5.5 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
__dbshow_filter_int:cccnn
Filter integers.

#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_gset_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
    #5
    \clist_map_inline:Nn \l__dbshow_filter_tmp_clist {
      \int_compare:nNnT { \l__dbshow_filter_diff_int } = {##1}
        { #4 \clist_map_break: }
    }
  } {
```

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_new_protected: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 }
  \tl_set:Nx \l__dbshow_type_tl { \__dbshow_get_type:nV {#1} \l__dbshow_attr_tl }
  \use:c
    { __dbshow_filter_\l__dbshow_type_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 {
  \__dbshow_check_database:n {#1}
  \__dbshow_check_attr:nn {#1} {#3}
  \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}
  }
  \bool_if_exist:cF { g__dbshow_cond_exist_bool_#1_#2 }
    { \bool_set_false:c { g__dbshow_cond_exist_bool_#1_#2 } }
  \bool_if:cF { g__dbshow_cond_exist_bool_#1_#2 }
    { \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 { [^!=&<>()\ ]+ } {#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_if_in:cnF { g__dbshow_filter_run_seq_#1_#2 }
        { g__dbshow_filter_hook_#1_##1:n }
      {
        \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|-|\d|\_)+ }
    { \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
\dbNewCond
\dbCombineConditionals
\dbCombCond

Environment to define conditionals and filters.

#1 : ⟨*database*⟩

```
\NewDocumentEnvironment { dbFilters } { s m } {
  \seq_if_exist:cF { g__dbshow_cond_seq_#2 }
    { \seq_new:c { g__dbshow_cond_seq_#2 } }
```

##1 : ⟨*star*⟩
##2 : ⟨*filter/conditional*⟩
##3 : ⟨*attr*⟩
##4 : ⟨*expr*⟩
##5 : ⟨*filter info*⟩

```
  \DeclareDocumentCommand { \dbNewConditional } { s m m m O{} } {
    \__dbshow_new_conditional:nnnnn {#2} {##2} {##3} {##4} {##1}
    \IfValueT {#1} {
      \dbCombCond{##2}{##2}[##5]
    }
  }
```

##1 : ⟨*filter*⟩

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

```
  \DeclareDocumentCommand { \dbCombineConditionals } { m m O{} } {
    \tl_gset:cn { g__dbshow_filter_info_tl_#2_##1 } {##3}
    \__dbshow_combine_conditional:nnn {#2} {##1} {##2}
  }
  \cs_set_eq:NN \dbNewCond  \dbNewConditional
  \cs_set_eq:NN \dbCombCond \dbCombineConditionals
} {}
```

(*End definition for* `dbFilters` *and others. These functions are documented on page 31.*)

`\dbNewRawFilter`
`\dbNewRawFilter*`

Define filter with single conditional.

#1 : ⟨*star*⟩
#2 : ⟨*filter and conditional name*⟩
#3 : ⟨*database*⟩
#4 : ⟨*attr*⟩
#5 : ⟨*expr*⟩
#6 : ⟨*filter info*⟩

```
\DeclareDocumentCommand { \dbNewRawFilter } { s m m m m O{} } {
  \seq_if_exist:cF { g__dbshow_cond_seq_#3 }
    { \seq_new:c { g__dbshow_cond_seq_#3 } }
  \__dbshow_new_conditional:nnnnn {#3} {#2} {#4} {#5} {#1}
  \tl_gset:cn { g__dbshow_filter_info_tl_#3_#2 } {#6}
  \__dbshow_combine_conditional:nnn {#3} {#2} {#2}
}
```

(*End definition for* `\dbNewRawFilter` *and* `\dbNewRawFilter*`*. These functions are documented on page 34.*)

`\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 31.*)

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

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

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

(*End definition for* `\dbIntAbs` *and others. These functions are documented on page 37.*)

5.6 Style and Options

__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    = ,
    code             .code:n       = {
      \bool_gset_false:c { g__dbshow_style_attr_exp_bool_#1_#2_#3 }
      \cs_gset:cn { __dbshow_style_attr_code_#1_#2_#3:n } {##1}
    },
    code             .initial:n    = {##1},
    code*            .code:n       = {
      \bool_gset_true:c  { g__dbshow_style_attr_exp_bool_#1_#2_#3 }
      \cs_gset:cn { __dbshow_style_attr_code_#1_#2_#3:n } {##1}
    },
```

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    = ,
    item-code        .code:n       = {
      \bool_gset_false:c { g__dbshow_style_clist_item_exp_bool_#1_#2_#3 }
      \cs_gset:cn { __dbshow_style_clist_item_code_#1_#2_#3:n } {##1}
    },
    item-code        .initial:n    = {##1},
    item-code*       .code:n       = {
      \bool_gset_true:c  { g__dbshow_style_clist_item_exp_bool_#1_#2_#3 }
      \cs_gset:cn { __dbshow_style_clist_item_code_#1_#2_#3:n } {##1}
    },
```

Only for date.

```
    zfill            .bool_gset:c  = {
      g__dbshow_style_date_zfill_bool_#1_#2_#3
    },
    zfill            .initial:n    = true,
    zfill            .default:n    = true,
    format-code      .code:n       = {
      \cs_gset:cn { __dbshow_style_date_format_code_#1_#2_#3:nnn } {##1}
      \cs_generate_variant:cn
        { __dbshow_style_date_format_code_#1_#2_#3:nnn } { xxx }
    },
  }
  \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:nVn {#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    = ,
    after-code       .tl_gset:c    = { g__dbshow_style_after_tl_#1_#2 },
    after-code       .initial:n    = ,
    item-before-code .tl_gset:c    = { g__dbshow_style_item_before_tl_#1_#2 },
    item-before-code .initial:n    = ,
    item-after-code  .tl_gset:c    = { g__dbshow_style_item_after_tl_#1_#2 },
    item-after-code  .initial:n    = ,
    item-code        .code:n       = {
      \bool_gset_false:c { g__dbshow_style_item_exp_bool_#1_#2 }
      \tl_gset:cn { g__dbshow_style_item_tl_#1_#2 } {##1}
    },
    item-code        .initial:n    = ,
```

```
    item-code*       .code:n       = {
      \bool_gset_true:c { g__dbshow_style_item_exp_bool_#1_#2 }
      \tl_gset:cn { g__dbshow_style_item_tl_#1_#2 } {##1}
    },
  }
  \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 } {
  \__dbshow_check_database:n {#3}
  \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* 22.)

5.7 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_fatal: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`.)

5.8 Display Data

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

`#1` : ⟨*before code tl*⟩
`#2` : ⟨*after code tl*⟩
`#3` : ⟨*item code cs*⟩
`#4` : ⟨*exp boolean var*⟩
`#5` : ⟨*item*⟩

```
\cs_new:Nn \__dbshow_clist_wrapper:NNNNn {
  \bool_if:NTF #4
    { \exp_not:n { { #1\exp_args:Nx#3{#5}#2 }, } }
    { \exp_not:n { { #1#3{#5}#2 }, } }
}
```

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

`\__dbshow_clist_use:NNNNNN` Display a comma list.
`\__dbshow_clist_use:cccccc`

`#1` : ⟨*data clist*⟩
`#2` : ⟨*separator clist*⟩

#3 : ⟨*before code tl*⟩
#4 : ⟨*after code tl*⟩
#5 : ⟨*item code cs*⟩
#6 : ⟨*exp boolean var*⟩

```
\cs_new:Nn \__dbshow_clist_use:NNNNNN {
  \int_case:nnF { \clist_count:N #2 } {
    { 1 } {
      \clist_use:xx
        { \clist_map_tokens:Nn #1 { \__dbshow_clist_wrapper:NNNNn #3#4#5#6 } }
        { \clist_item:Nn #2 { 1 } }
    }
    { 3 } {
      \clist_use:xxxx
        { \clist_map_tokens:Nn #1 { \__dbshow_clist_wrapper:NNNNn #3#4#5#6 } }
        { \clist_item:Nn #2 { 1 } }
        { \clist_item:Nn #2 { 2 } }
        { \clist_item:Nn #2 { 3 } }
    }
  } {
    \__dbshow_sep_fatal:xxx
      { 1~or~3 }
      { \clist_count:N #2 }
      { \clist_use:Nn #2 { ,~ } }
  }
}
\cs_generate_variant:Nn \__dbshow_clist_use:NNNNNN { cccccc }
```

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

`\__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_fatal: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`
`\__dbshow_use_data_raw:nnnn`

Display Data. `\__dbshow_use_data:nnnn` wrap the ⟨*attr*⟩ data and `\__dbshow_use_data_raw:nnnn` display the underlying data.

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

```
\cs_new:Nn \__dbshow_use_data:nnnn {
  \bool_if:cTF { g__dbshow_style_attr_exp_bool_#4_#1_#2 } {
    \protected@edef\@dbshow@tmp{\__dbshow_use_data_raw:nnnn {#1} {#2} {#3} {#4}}
    \exp_args:Nno
    \use:c { __dbshow_style_attr_code_#4_#1_#2:n } { \@dbshow@tmp }
  } {
    \use:c { __dbshow_style_attr_code_#4_#1_#2:n }
    { \__dbshow_use_data_raw:nnnn {#1} {#2} {#3} {#4} }
  }
}
\cs_new:Nn \__dbshow_use_data_raw: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:cccccc { 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 }
        {   __dbshow_style_clist_item_code_    #4_#1_#2:n }
        { g__dbshow_style_clist_item_exp_bool_#4_#1_#2 }
    }
    { date }  {
      \cs_if_exist_use:cTF { __dbshow_style_date_format_code_#4_#1_#2:xxx } {
        { \__dbdate_get_year:n  { g__dbshow_data_#1_#2_#3 } }
        { \__dbdate_get_month:n { g__dbshow_data_#1_#2_#3 } }
        { \__dbdate_get_day:n   { g__dbshow_data_#1_#2_#3 } }
      } {
        \__dbshow_date_use:ncc { g__dbshow_data_#1_#2_#3 }
          { g__dbshow_style_attr_sep_#4_#1_#2 }
          { g__dbshow_style_date_zfill_bool_#4_#1_#2 }
```

```
      }
    }
  }
}
```

(*End definition for* `\__dbshow_use_data:nnnn` *and* `\__dbshow_use_data_raw: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 38.*)

`\__dbshow_show_filter:nnN` 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:nnN`.)

`\__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 38.*)

`\__dbshow_show_set_if_last:NN` Define conditional to check if the current item is the last item.

`\dbIfLastT`
`\dbIfLastF`
`\dbIfLastTF`

`#1` : ⟨*current index*⟩
`#2` : ⟨*count*⟩

```
\cs_new_protected:Nn \__dbshow_show_set_if_last:NN {
  \prg_set_conditional:Nnn \__dbshow_show_if_last: { T, F, TF } {
    \int_compare:nNnTF {#1} = {#2}
      { \prg_return_true: }
      { \prg_return_false: }
  }
  \cs_set_eq:NN \dbIfLastT  \__dbshow_show_if_last:T
  \cs_set_eq:NN \dbIfLastF  \__dbshow_show_if_last:F
  \cs_set_eq:NN \dbIfLastTF \__dbshow_show_if_last:TF
}
```

(*End definition for* `\__dbshow_show_set_if_last:NN` *and others. These functions are documented on page 36.*)

`\__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
  \int_zero_new:N \l__dbshow_show_count_int
  \int_set:Nn \l__dbshow_show_count_int { \clist_count:N #3 }
  \tl_clear_new:N \l__dbshow_item_tl
```

`##1` : ⟨*index*⟩

```
  \clist_map_inline:Nn #3 {
    \int_incr:N \l__dbshow_show_int
    \__dbshow_show_set_if_last:NN \l__dbshow_show_int \l__dbshow_show_count_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 }
    }
    \bool_if:cTF { g__dbshow_style_item_exp_bool_#1_#2 } {
```

```
      \protected@edef\@dbshow@tmp{\tl_use:c { g__dbshow_style_item_before_tl_#1_#2 }}
      \tl_put_right:No \l__dbshow_item_tl { \@dbshow@tmp }
      \protected@edef\@dbshow@tmp{\tl_use:c { g__dbshow_style_item_       tl_#1_#2 }}
      \tl_put_right:No \l__dbshow_item_tl { \@dbshow@tmp }
      \protected@edef\@dbshow@tmp{\tl_use:c { g__dbshow_style_item_after_ tl_#1_#2 }}
    } {
      \tl_use:c { g__dbshow_style_item_before_tl_#1_#2 }
      \tl_use:c { g__dbshow_style_item_       tl_#1_#2 }
      \tl_use:c { g__dbshow_style_item_after_ tl_#1_#2 }
    }
  }
  \l__dbshow_item_tl
}
```

(*End definition for* `\__dbshow_show_item:nn`, `\dbIndex`, *and* `\dbuse`. *These functions are documented on page 38.*)

`\__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 36.*)

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

`\__dbshow_show:nnv`

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

```
\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} }
  \__dbshow_show_set_cond:N \l__dbshow_show_index_clist
  \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_style:nn   {#1} {#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* 22.)

5.9 Date Type

⟨@@=dbdate⟩

#1 : ⟨*date*⟩
#2 : ⟨*date sep*⟩

```
\msg_new:nnn { dbshow } { wrong-date-sep } {
  can~not~parse~the~date~'#1'~with~the~global~date~separator~'#2'~
  \msg_line_context:.~Please~set~the~correct~date~separator~with~
  \dbdatesep.
}
```

__dbdate_check_date_sep:nn
__dbdate_check_date_sep:Vn
Check if the global date separator is valid.

#1 : ⟨*date*⟩
#2 : ⟨*date sep*⟩

```
\cs_new:Nn \__dbdate_check_date_sep:nn {
  \int_zero_new:N \l__dbdate_sep_int
  \tl_map_inline:nn {#1} {
    \tl_if_eq:nnT {#2} {##1} { \int_incr:N \l__dbdate_sep_int }
    \int_compare:nNnT { \l__dbdate_sep_int } > { 2 } { \tl_map_break: }
  }
  \int_compare:nNnF { \l__dbdate_sep_int } = { 2 }
    { \msg_fatal:nnnn { dbshow } { wrong-date-sep } {#1} {#2} }
}
\cs_generate_variant:Nn \__dbdate_check_date_sep:nn { nV }
```

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

__dbdate_if_leap_p:n
__dbdate_if_leap:n*TF*
Check if the year is leap.

#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` `\__dbdate_clear_new:x` `\__dbdate_gclear_new:n` `\__dbdate_gclear_new:x`

Clear or create a new date variable.

`#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_get_year:n` `\__dbdate_get_year:x` `\__dbdate_get_month:n` `\__dbdate_get_month:x` `\__dbdate_get_day:n` `\__dbdate_get_day:x`

Extract year, month or day from ⟨*date var*⟩.

`#1` : ⟨*date var*⟩

```
\cs_new:Nn \__dbdate_get_year:n {
  \int_use:c { __dbdate_year_#1 }
}
\cs_new:Nn \__dbdate_get_month:n {
  \int_use:c { __dbdate_month_#1 }
}
\cs_new:Nn \__dbdate_get_day:n {
  \int_use:c { __dbdate_day_#1 }
}
\cs_generate_variant:Nn \__dbdate_get_year:n  { x }
\cs_generate_variant:Nn \__dbdate_get_month:n { x }
\cs_generate_variant:Nn \__dbdate_get_day:n   { x }
```

(*End definition for* `\__dbdate_get_year:n`, `\__dbdate_get_month:n`, *and* `\__dbdate_get_day:n`.)

`\__dbdate_set:nnnn` `\__dbdate_gset:nnnn`

Set the value of ⟨*date var*⟩.

`#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* 30.)

`\__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_check_date_sep:nV {#2} \g__dbdate_sep_tl
  \__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_check_date_sep:nV {#2} \g__dbdate_sep_tl
  \__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` Prepend 0 to single digit.
`\__dbdate_show_two:c`

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

`#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` Display date with the same separator.

`\__dbdate_use:nf`
`\__dbdate_use_zfill:nn`
`\__dbdate_use_zfill:nf`

```
\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* 19.)

```
\endinput
⟨/package⟩
```

Change History

1.1

2022-01-05 **Add macro**: \dbarabic, \dbalph, \dbAlph, \dbroman, \dbRoman 19, 38
2022-01-06 **Add option**: raw-filter 4, 23
2022-01-06 **Fix bug**: \dbIndex not defined 19, 38
2022-01-07 **Update doc**: improve example 20, 39

1.2

2022-01-07 **Add doc**: add comparison to datatool 3, 21
2022-01-07 **Add macro**: \dbclear 3, 22
2022-01-08 **Add options**: record-before-code, record-after-code 7, 25
2022-01-08 **Fix bug**: string sorting bug 5, 24
2022-01-08 **Remove macros**: \dbItemIfEmpty(TF), \dbClistItemIfEmpty(TF) 18, 37
2022-01-08 **Update macro**: make \dbuse fully-expandable 17, 36

1.3

2022-01-07 **Remove doc**: remove comparison to datatool 3, 21
2022-01-08 **Add macro**: \dbsave* 16, 36
2022-01-08 **Add option**: <attr>/zfill 11, 30
2022-01-08 **Remove dependency**: datatime2 . . . . . 3, 21
2022-01-08 **Update logic**: swap definition of starred and unstarred conditionals of date 15, 34
2022-01-09 **Add doc**: descripton for expansion 2, 21
2022-01-09 **Update option**: <attr>/sep 9, 28
2022-01-10 **Add macros**: expression function aliases 18, 37
2022-01-10 **Update doc**: truncated division 13, 32

1.4

2022-01-10 **Add check**: version of l3kernel 2, 21
2022-01-10 **Fix doc code**: wrong changes history sorting 72
2022-01-10 **Update macro**: change | to \dbshow_sep in weird argument of
\dbshow_process_default_value:w 3, 22
2022-01-13 **Add macro**: \dbdatesep 11, 30
2022-01-13 **Add macro**: \dbitemkv 16, 35
2022-01-13 **Update code**: merge code and doc into dbshow.dtx 40
2022-01-13 **Update env**: dbitem 16, 35
2022-01-14 **Fix bug**: data cannot contain \par 47

1.5

2022-01-14 **Add options**: <attr>/code, <attr>/code* 7, 26
2022-01-14 **Add option**: <attr>/format-code 11, 29
2022-01-14 **Remove option**: <attr>/wrapper 7, 26
2022-01-14 **Update options**: Rename record-before-code and record-after-code to item-before-code and
item-after-code 7, 25
2022-01-15 **Add check**: check if database is valid in \dbNewStyle 57
2022-01-15 **Add macros**: \dbNewCond, \dbCombCond 13, 32
2022-01-15 **Fix bug**: \dbIfEmptyF undefined 64
2022-01-15 **Fix bug**: can not use \dbdatesep midway 43
2022-01-15 **Update check code**: transform arguments to string before check 41
2022-01-16 **Add macro**: \dbNewRawFilter 15, 34
2022-01-16 **Add options**: <attr>/item-code, <attr>/item-code* 9, 27
2022-01-16 **Update env**: add starred version of dbFilters 12, 31
2022-01-17 **Add macros**: \dbIfLastT, \dbIfLastF, \dbIfLastTF 17, 36
2022-01-17 **Add option**: item-code* 6, 25
2022-01-17 **Remove doc**: Remove big example 20, 39

Add

v1.1 2022-01-05 **Add macro**: \dbarabic, \dbalph, \dbAlph, \dbroman, \dbRoman 19, 38

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

M

N

O

P

R