04丨使用DDL创建数据库&数据表时需要注意什么?

DDL 是 DBMS 的核心组件，也是 SQL 的重要组成部分，DDL 的正确性和稳定性是整个 SQL 运行的重要基础。面对同一个需求，不同的开发人员创建出来的数据库和数据表可能千差万别，那么在设计数据库的时候，究竟什么是好的原则？我们在创建数据表的时候需要注意什么？

今天的内容，你可以从以下几个角度来学习：

1. 了解 DDL 的基础语法，它如何定义数据库和数据表；
2. 使用 DDL 定义数据表时，都有哪些约束性；
3. 使用 DDL 设计数据库时，都有哪些重要原则。

DDL 的基础语法及设计工具

DDL 的英文全称是 Data Definition Language，中文是数据定义语言。它定义了数据库的结构和数据表的结构。

在 DDL 中，我们常用的功能是增删改，分别对应的命令是 CREATE、DROP 和 ALTER。需要注意的是，在执行 DDL 的时候，不需要 COMMIT，就可以完成执行任务。

1.对数据库进行定义

复制代码

```
CREATE DATABASE nba; // 创建一个名为 nba 的数据库
DROP DATABASE nba; // 删除一个名为 nba 的数据库
```

2.对数据表进行定义

创建表结构的语法是这样的：

复制代码

```
CREATE TABLE table_name
```

创建表结构

比如我们想创建一个球员表，表名为 player，里面有两个字段，一个是 player_id，它是 int 类型，另一个 player_name 字段是`varchar(255)`类型。这两个字段都不为空，且 player_id 是递增的。

那么创建的时候就可以写为：

复制代码

```
CREATE TABLE player  (
  player_id int(11) NOT NULL AUTO_INCREMENT,
  player_name varchar(255) NOT NULL
);
```

需要注意的是，语句最后以分号（;）作为结束符，最后一个字段的定义结束后没有逗号。数据类型中 int(11) 代表整数类型，显示长度为 11 位，括号中的参数 11 代表的是最大有效显示长度，与类型包含的数值范围大小无关。`varchar(255)`代表的是最大长度为 255 的可变字符串类型。`NOT NULL`表明整个字段不能是空值，是一种数据约束。`AUTO_INCREMENT`代表主键自动增长。

实际上，我们通常很少自己写 DDL 语句，可以使用一些可视化工具来创建和操作数据库和数据表。在这里我推荐使用 Navicat，它是一个数据库管理和设计工具，跨平台，支持很多种数据库管理软件，比如 MySQL、Oracle、MariaDB 等。基本上专栏讲到的数据库软件都可以使用 Navicat 来管理。

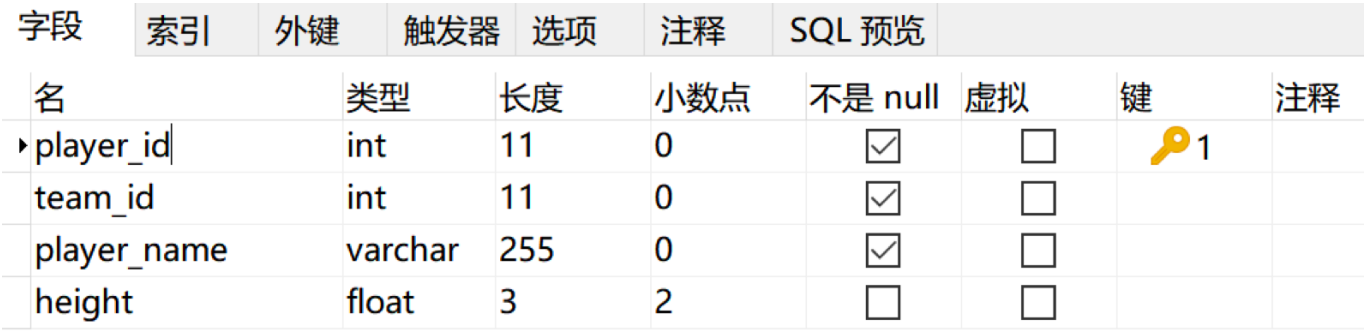
假如还是针对 player 这张表，我们想设计以下的字段：

字段	含义	类型
player_id	球员ID	int整数类型，最大显示长度11
team_id	球队ID	int整数类型，最大显示长度11
player_name	球员姓名	varchar字符串类型，最大长度255
height	身高	float浮点类型，一共存储3个有效数字，其中小数点长度为2

其中 player_id 是数据表 player 的主键，且自动增长，也就是 player_id 会从 1 开始，然

后每次加 1。player_id、team_id、player_name 这三个字段均不为空，height 字段可以为空。

按照上面的设计需求，我们可以使用 Navicat 软件进行设计，如下所示：



然后，我们还可以对 player_name 字段进行索引，索引类型为`Unique`。使用 Navicat 设置如下：

这样一张 player 表就通过可视化工具设计好了。我们可以把这张表导出来，可以看看这张表对应的 SQL 语句是怎样的。方法是在 Navicat 左侧用右键选中 player 这张表，然后选择“转储 SQL 文件”→“仅结构”，这样就可以看到导出的 SQL 文件了，代码如下：

复制代码

```
DROP TABLE IF EXISTS `player`;
CREATE TABLE `player`  (
  `player_id` int(11) NOT NULL AUTO_INCREMENT,
  `team_id` int(11) NOT NULL,
  `player_name` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `height` float(3, 2) NULL DEFAULT 0.00,
  PRIMARY KEY (`player_id`) USING BTREE,
  UNIQUE INDEX `player_name`(`player_name`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;
```

你能看到整个 SQL 文件中的 DDL 处理，首先先删除 player 表（如果数据库中存在该表的话），然后再创建 player 表，里面的数据表和字段都使用了反引号，这是为了避免它们的

名称与 MySQL 保留字段相同，对数据表和字段名称都加上了反引号。

其中 player_name 字段的字符编码是 utf8，排序规则是`utf8_general_ci`，代表对大小写不敏感，如果设置为`utf8_bin`，代表对大小写敏感，还有许多其他排序规则这里不进行介绍。

因为 player_id 设置为了主键，因此在 DDL 中使用`PRIMARY KEY`进行规定，同时索引方法采用 BTREE。

因为我们对 player_name 字段进行索引，在设置字段索引时，我们可以设置为`UNIQUE INDEX`（唯一索引），也可以设置为其他索引方式，比如`NORMAL INDEX`（普通索引），这里我们采用`UNIQUE INDEX`。唯一索引和普通索引的区别在于它对字段进行了唯一性的约束。在索引方式上，你可以选择`BTREE`或者`HASH`，这里采用了`BTREE`方法进行索引。我会在后面介绍`BTREE`和`HASH`索引方式的区别。

整个数据表的存储规则采用 InnoDB。之前我们简单介绍过 InnoDB，它是 MySQL5.5 版本之后默认的存储引擎。同时，我们将字符编码设置为 utf8，排序规则为`utf8_general_ci`，行格式为`Dynamic`，就可以定义数据表的最后约定了：

复制代码

```
ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;
```

你能看出可视化工具还是非常方便的，它能直接帮我们将数据库的结构定义转化成 SQL 语言，方便数据库和数据表结构的导出和导入。不过在使用可视化工具前，你首先需要了解对于 DDL 的基础语法，至少能清晰地看出来不同字段的定义规则、索引方法，以及主键和外键的定义。

修改表结构

在创建表结构之后，我们还可以对表结构进行修改，虽然直接使用 Navicat 可以保证重新导出的数据表就是最新的，但你也有必要了解，如何使用 DDL 命令来完成表结构的修改。

1. 添加字段，比如我在数据表中添加一个 age 字段，类型为`int(11)`

复制代码

```
ALTER TABLE player ADD (age int(11));
```

2. 修改字段名，将 age 字段改成`player_age`

复制代码

```
ALTER TABLE player RENAME COLUMN age to player_age
```

3. 修改字段的数据类型，将`player_age`的数据类型设置为`float(3,1)`

复制代码

```
ALTER TABLE player MODIFY (player_age float(3,1));
```

4. 删除字段, 删除刚才添加的`player_age`字段

复制代码

```
ALTER TABLE player DROP COLUMN player_age;
```

数据表的常见约束

当我们创建数据表的时候，还会对字段进行约束，约束的目的在于保证 RDBMS 里面数据的准确性和一致性。下面，我们来看下常见的约束有哪些。

首先是主键约束。

主键起的作用是唯一标识一条记录，不能重复，不能为空，即 UNIQUE+NOT NULL。一个数据表的主键只能有一个。主键可以是一个字段，也可以由多个字段复合组成。在上面的例子中，我们就把 player_id 设置为了主键。

其次还有外键约束。

外键确保了表与表之间引用的完整性。一个表中的外键对应另一张表的主键。外键可以是重复的，也可以为空。比如 player_id 在 player 表中是主键，如果你想设置一个球员比分表即 player_score，就可以在 player_score 中设置 player_id 为外键，关联到 player 表中。

除了对键进行约束外，还有字段约束。

唯一性约束。

唯一性约束表明了字段在表中的数值是唯一的，即使我们已经有了主键，还可以对其他字段进行唯一性约束。比如我们在 player 表中给 player_name 设置唯一性约束，就表明任何两个球员的姓名不能相同。需要注意的是，唯一性约束和普通索引（NORMAL INDEX）之间是有区别的。唯一性约束相当于创建了一个约束和普通索引，目的是保证字段的正确性，而普通索引只是提升数据检索的速度，并不对字段的唯一性进行约束。

NOT NULL 约束。对字段定义了 NOT NULL，即表明该字段不应为空，必须有取值。

DEFAULT，表明了字段的默认值。如果在插入数据的时候，这个字段没有取值，就设置为默认值。比如我们将身高 height 字段的取值默认设置为 0.00，即`DEFAULT 0.00`。

CHECK 约束，用来检查特定字段取值范围的有效性，CHECK 约束的结果不能为 FALSE，比如我们可以对身高 height 的数值进行 CHECK 约束，必须≥0，且＜3，即`CHECK(height>=0 AND height<3)`。

设计数据表的原则

我们在设计数据表的时候，经常会考虑到各种问题，比如：用户都需要什么数据？需要在数据表中保存哪些数据？哪些数据是经常访问的数据？如何提升检索效率？

如何保证数据表中数据的正确性，当插入、删除、更新的时候该进行怎样的约束检查？

如何降低数据表的数据冗余度，保证数据表不会因为用户量的增长而迅速扩张？

如何让负责数据库维护的人员更方便地使用数据库？

除此以外，我们使用数据库的应用场景也各不相同，可以说针对不同的情况，设计出来的数据表可能千差万别。那么有没有一种设计原则可以让我们来借鉴呢？这里我整理了一个“**三少一多**”原则：

1.**数据表的个数越少越好**

RDBMS 的核心在于对实体和联系的定义，也就是 E-R 图（Entity Relationship Diagram），数据表越少，证明实体和联系设计得越简洁，既方便理解又方便操作。

2.**数据表中的字段个数越少越好**

字段个数越多，数据冗余的可能性越大。设置字段个数少的前提是各个字段相互独立，而不是某个字段的取值可以由其他字段计算出来。当然字段个数少是相对的，我们通常会在数据冗余和检索效率中进行平衡。

3.**数据表中联合主键的字段个数越少越好**

设置主键是为了确定唯一性，当一个字段无法确定唯一性的时候，就需要采用联合主键的方式（也就是用多个字段来定义一个主键）。联合主键中的字段越多，占用的索引空间越大，不仅会加大理解难度，还会增加运行时间和索引空间，因此联合主键的字段个数越少越好。

4.**使用主键和外键越多越好**

数据库的设计实际上就是定义各种表，以及各种字段之间的关系。这些关系越多，证明这些实体之间的冗余度越低，利用度越高。这样做的好处在于不仅保证了数据表之间的独立性，还能提升相互之间的关联使用率。

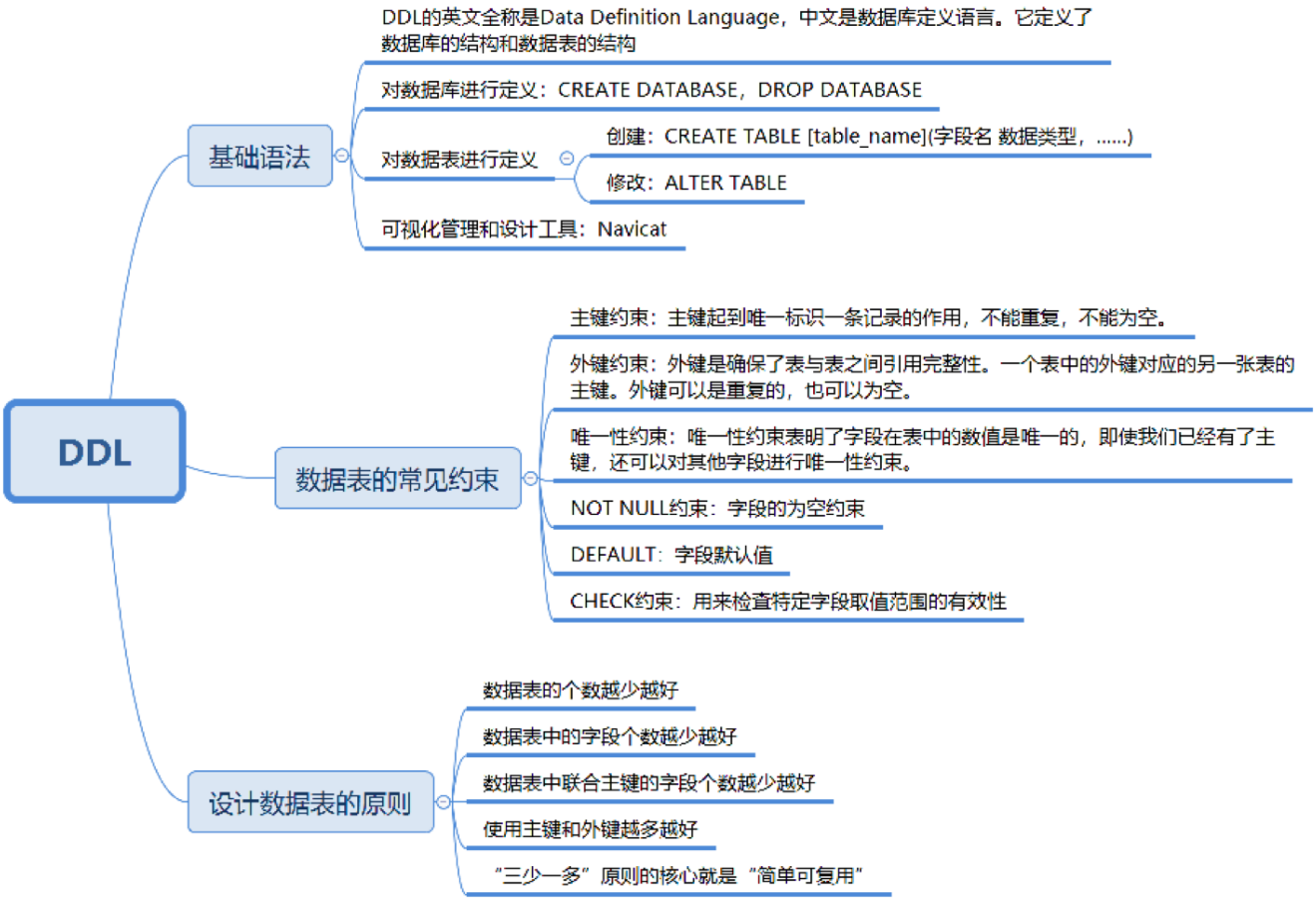
你应该能看出来“三少一多”原则的核心就是简单可复用。简单指的是用更少的表、更少的字段、更少的联合主键字段来完成数据表的设计。可复用则是通过主键、外键的使用来增强数据表之间的复用率。因为一个主键可以理解是一张表的代表。键设计得越多，证明它们之间的利用率越高。

总结

今天我们学习了 DDL 的基础语法，比如如何对数据库和数据库表进行定义，也了解了使用 Navicat 可视化管理工具来辅助我们完成数据表的设计，省去了手写 SQL 的工作量。

在创建数据表的时候，除了对字段名及数据类型进行定义以外，我们考虑最多的就是关于字段的约束，我介绍了 7 种常见的约束，它们都是数据表设计中会用到的约束：主键、外键、唯一性、NOT NULL、DEFAULT、CHECK 约束等。

当然，了解了如何操作创建数据表之后，你还需要动脑思考，怎样才能设计出一个好的数据表？设计的原则都有哪些？针对这个，我整理出了“三少一多”原则，在实际使用过程中，你需要灵活掌握，因为这个原则并不是绝对的，有时候我们需要牺牲数据的冗余度来换取数据处理的效率。



我们在创建数据表的时候，会对数据表设置主键、外键和索引。你能说下这三者的作用和区别吗？