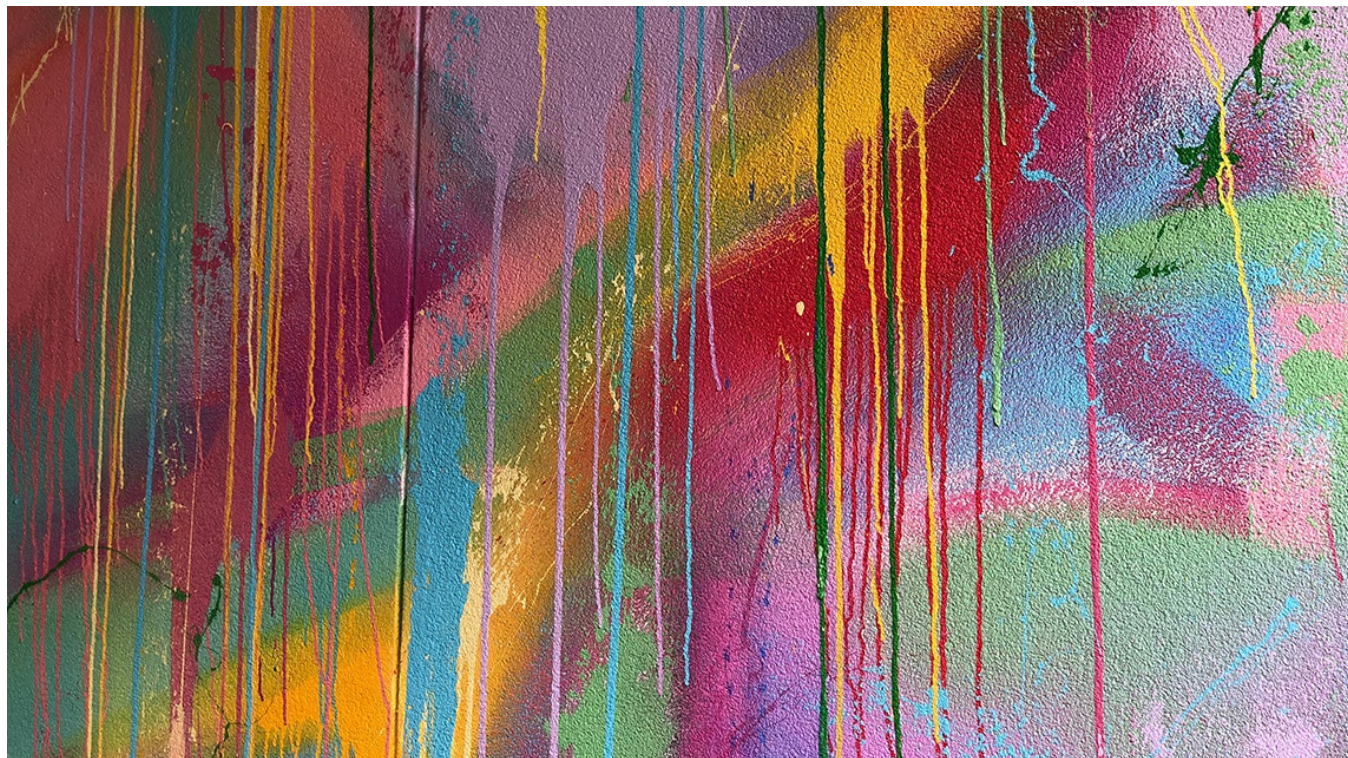


04 | 使用DDL创建数据库&数据表时需要注意什么？

2019-06-19 陈旻



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

```
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：

```
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：

```
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：

字段	索引	外键	触发器	选项	注释	SQL 预览		
名		类型	长度	小数点	不是 null	虚拟	键	注释
▶ player_id		int	11	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1	
team_id		int	11	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
player_name		varchar	255	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
height		float	3	2	<input type="checkbox"/>	<input type="checkbox"/>		

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：

字段	索引	外键	触发器	选项	注释	SQL 预览	
名		字段				索引类型	索引方法
▶ player_name		`player_name`				UNIQUE	BTREE

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：

```
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字段的字符集是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，我们将字符集设置为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：

```
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)

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

2.修改字段名，将age字段改成player_age

```
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)

```
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

```
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。

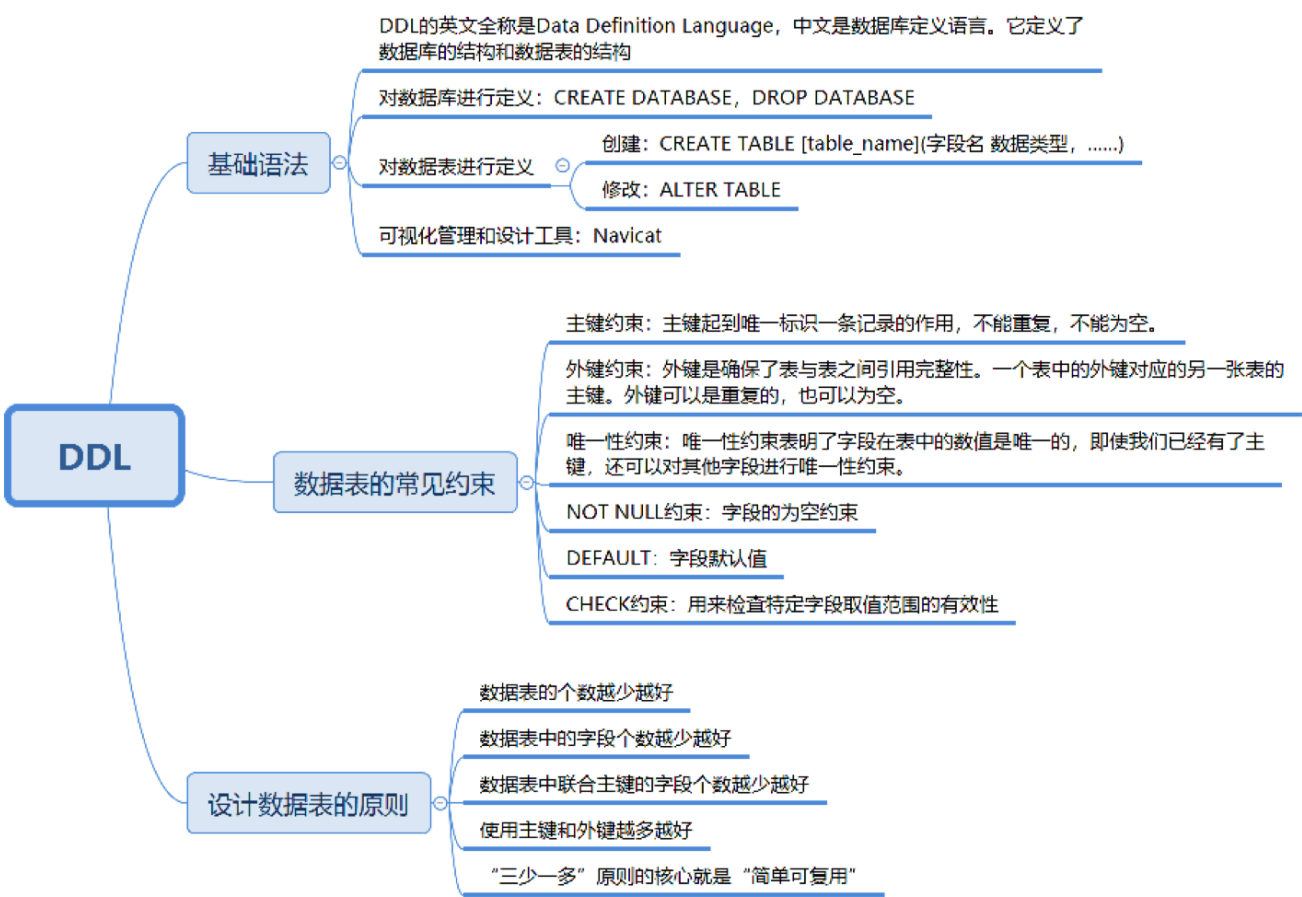
总结

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？

欢迎你在评论区写下你的答案，我会和你一起交流，也欢迎把这篇文章分享给你的朋友或者同事。

SQL 必知必会

从入门到数据实战

陈旸

清华大学计算机博士



新版升级：点击「👤请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言



我知道了恩

外键多了会有很多维护问题吧？

2019-06-19

作者回复

是否使用外键确实会有一些争议。我来解释下关于外键的使用：

首先，外键本身是为了实现强一致性，所以如果需要正确性>性能的话，还是建议使用外键，它可以让我们在数据库的层面保证数据的完整性和一致性。

当然不用外键，你也可以在业务层进行实现。不过，这样做也同样存在一定的风险，因为这样，就会让业务逻辑会与数据具备一定的耦合性。也就是业务逻辑和数据必须同时修改。而且在工作中，业务层可能会经常发生变化。

当然，很多互联网的公司，尤其是超大型的数据应用场景，大量的插入，更新和删除在外键的约束下会降低性能，同时数据库在水平拆分和分库的情况下，数据库端也做不到执行外键约束。另外，在高并发的情况下，外键的存在也会造成额外的开销。因为每次更新数据，都需要检查另外一张表的数据，也容易造成死锁。

所以在这种情况下，尤其是大型项目中后期，可以采用业务层来实现，取消外键提高效率。

不过在SQL学习之初，包括在系统最初设计的时候，还是建议你采用规范的数据库设计，也就是采用外键来对数据表进行约束。因为这样可以建立一个强一致性，可靠性高的数据库结构，也不需要业务层来实现过多的检查。

👍 49

当然在项目后期，业务量增大的情况下，你需要更多考虑到数据库性能问题，可以取消外键的约束，转移到业务层来实现。而且在大型互联网项目中，考虑到分库分表的情况，也会降低外键的使用。

不过在SQL学习，以及项目早期，还是建议你使用外键。在项目后期，你可以分析有哪些外键造成了过多的性能消耗。一般遵循2/8原则，会有20%的外键造成80%的资源效率，你可以只把这20%的外键进行开放，采用业务层逻辑来进行实现，当然你需要保证业务层的实现没有错误。不同阶段，考虑的问题不同。当用户和业务量增大的时候，对于大型互联网应用，也会通过减少外键的使用，来减低死锁发生的概率，提高并发处理能力。

2019-06-19



KaitoShy

👍 10

主键：唯一标识一条记录，不能重复，不能为空。

外键：确保了表于表之间引用的完整性，可以重复，可以为空

索引：提升数据检索速度

区别：主键是索引的一种，而且是唯一索引的一种。而并非在是索引，是两表之间的链接

疑问：1. 关于使用主键和外键越多越好有个疑问，是否涉及到数据库的维护成本，性能等问题呢？而我们现实生活中很少到外键约束，基本上用程序维持这种关系的。

2. 设计原则和之前了解的“数据库设计的三大范式”有什么关系呢？个人感觉是从不同纬度讲述了设计的方法，三大范式过于理论化，而老师的更具有实操性。不过还是希望老师，讲解一下。

2019-06-19



pepezzzz

👍 9

外键怎么会越多越好呢？现在的数模设计都是不用外键，会带来数据归档和数据操作等一系列问题。

2019-06-19



Ant

👍 6

老师您好，mysql的编码utf-8是3个字节的，它和我们传统的utf-8编码是不一样的，mysql中不是用了utf-8mb4来对应4字节的utf-8编码吗？我在项目中一般都会用utf-8mb4这种，请问下这两个在生产环境如何做选择呢？

2019-06-19



kyle

👍 5

外键越多越好吗？应该是维护麻烦的缘故，我们都没使用外键了！

2019-06-19



夜路破晓

👍 4

修改字段数据类型,报错,改写成:

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

2019-06-19





梁

👍 3

在阿里巴巴《**JAVA**开发手册》中有这么一条
“【强制】不得使用外键与级联，一切外键概念必须在应用层解决。”，说明指出
“外键与级联更新适用于单机低并发，不适合分布式、高并发集群;级联更新是强阻塞，存在数据库更新风暴的风险;外键影响数据库的插入速度。”
老师对此怎么看呢？

2019-06-22



木偶人King

👍 3

老师咱们这里
ALTER TABLE player RENAME COLUMN age to player_age
在mysql中报错。mysql中修改列名属性用**change**或者**modify**吧。
修改表名用**rename to**。

2019-06-20



夜路破晓

👍 3

类比自己，
主键就好比是我的身份证；
外键就好比我在各种团队组织中的身份，如在单位是员工和管理者、在家是儿子和丈夫、在协会是会员和委员等；
索引就好比是我的某些特征或者独树一帜的风格，玉树临风、风流倜傥之类的。

2019-06-19

作者回复

可以这么理解，这么说也比较有趣。在数据库系统中实现他们会采用一些技术方式。比如索引，常用的算法有**BTree**何**Hash**索引，是一种数据结构来实现快速检索的目的。

2019-06-19



风中花

👍 2

老师好！可不可说下外键和冗余一个字段的的不同！有时候是可以到达一样的效果！我早期使用**orcal**数据哪会全是外键设计！我记得做数据很麻烦和插入数据也有困惑！那时的感觉！也许就是老师说的数据强一致的作用效果！现在大多数关系关联全是冗余字段！其实我一直没有明白真正的差异和区别！望老师能解惑！谢谢

2019-06-19



allea

👍 2

实际生产环境，更多的是用一个冗余字段取代外键吧？

2019-06-19

作者回复

分情况而定，如果外键或者这些属性字段不需要修改的话，可以使用冗余字段替代，达到通过“空间换时间”的效果

所以呢，如果冗余字段可控好维护，可以使用。如果涉及到级联删除/级联更新，冗余字段不可控，建议还是采用外键的方式。

2019-06-19

ALTER TABLE student DROP COLUMN player_age;

这里是 player 吧?

2019-06-19



Sam

👍 2

主键，在一个表中唯一标记一行数据，作用等效于唯一约束+not null；在MySQL InnoDB引擎，数据表按照主键来组织表数据，形成索引组织表；在Oracle中，也有IOT表，但不显式指定创建索引组织表，那么数据表默认是堆表，主键上另外创建一个主键索引；

外键，是指定两表之间的父子关系，子表外键上的数据，必须要在父表存在，在父表和子表的外键相关列上创建索引，会大大提高性能；

索引，是提高查询效率的关键手段。索引是跟数据表相互独立的数据库对象，除非是Oracle中的索引组织表或者MySQL中的主键索引。索引的类型有B+树，hash，bitmap，全文索引等等。在表数据更新时需要维护索引结构，对DML效率有影响。

2019-06-19



阡陌

👍 2

可是实际工作的时候更多的外键确实增大了维护难度。对于老师说的外键越多越好表示难以理解。老师可否细讲一下

2019-06-19



chengzise

👍 2

我的理解：

主键：确保本表每行数据的唯一性

外键：与外表建立连接

索引：加快本表数据的查找

2019-06-19



supermouse

👍 1

主键是表中的记录的唯一标识，不能重复，不能为空，用来区分同一张表中不同的记录。一张表的外键是另一张表的主键，起到关联的作用，同时可以降低数据的冗余度，提高数据利用率。索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息，索引的一个主要目的就是加快检索表中数据。对于经常用来作为查询条件的字段，应该建立索引。但是对着数据量的增加，索引占用的空间也会越来越大

2019-06-22



O...O

👍 1

银行业的数据库基本不用自增长的主键，不利于维护，基本上都是使用唯一标识字段uuid+日期+渠道流水(unique index)来保证数据的唯一性；老师您的这种方法很适合初学者和复习者。

2019-06-20



Hoo-Ah

👍 1

一张表只能有一个主键，或者是联合主键用来确保主键的唯一性；外键使用来关联其他表结构

的，比如一张表使用外键关联另一张表的主键用来显示表之间的关联关系，但是外键也有缺点就是在大并发的场景下对数据库频繁的增删查改会降低效率；索引是为了提高数据的查询效率，就好比一本书的目录。

mysql的索引使用的B+tree，最后一层之间的数据使用链表可以减少回表操作提高查询效率。

2019-06-19



ABC

👍 1

我的理解:

主键:用来在表中确定一条数据，唯一且不可为空;

外键:用来关联两个表的数据，对删除有约束，可以防止误删除数据(在实际删除时，也需要先删除外键关联的数据才能删除当前表的数据)。外键可以为空。

索引:用来加快查询的速度，有唯一索引和普通索引。

区别:

主键:确定唯一性，增删改的时候都会用到;查询单条记录的时候会用到，不可为空;

外键:查询和删除，修改，可为空;

索引:查询，统计数据。

2019-06-19



创建表结构那里 是不是没把player_id设为主键

👍 1

2019-06-19