《精通SQL——结构化查询语言详解》看到42页

[基础 1](#_Toc257334974)

[SQL数据类型 1](#_Toc257334975)

[三范式 1](#_Toc257334976)

[创建 和 删除 数据库 2](#_Toc257334977)

[表的种类 2](#_Toc257334978)

[创建数据库 3](#_Toc257334979)

[删除数据库 3](#_Toc257334980)

[数据定义语言DDL 3](#_Toc257334981)

[创建表 3](#_Toc257334982)

[修改表 3](#_Toc257334983)

[删除表 3](#_Toc257334984)

[创建索引 3](#_Toc257334985)

[创建试图 3](#_Toc257334986)

[数据查询语言DQL 4](#_Toc257334987)

[数据操作语言DML 4](#_Toc257334988)

[数据控制语言DCL 4](#_Toc257334989)

[其它语言要素 4](#_Toc257334990)

[PL/SQL 4](#_Toc257334991)

# 基础

## SQL数据类型

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字符 | |  |  | | --- | --- | | char(size) | size<=255 | | varchar(size) |  | | nchar | char的拓展, 支持多字节和Unicode | | nvarchar | varchar的拓展, 支持多字节和Unicode | |
| 数字 | 精度: 整数部分和小数部分的总长度  范围: 小数部分长度, 范围等于0表示为整数, 不能是负数, 也不能大于精度.   |  |  | | --- | --- | | bit | 单比特数据, 只能取值0或1, 如果输入0或1以外的值将被视为1. 不能定义为null值. (-2147483648到2147483647) | | int | 4比特的整数 | | real | 4比特的浮点数, 可以指定精度, 但没有范围 | | double precision | 双精度浮点数 | | float | 可精确到第15位小数, 占8字节空间, 可以指定精度而不指定范围. | | smallint | 2比特的整数 (-32768到32767) | | tinyint | 1比特的整数 (0到255) | | number  或  decimal | 可存放实型和整型.  number(size) 最大数字的位数由括号中的参数size设置;  number(size, d) 最大数字的位数由括号中的参数size设置, 而d为小数的位数; | |
| 货币 | |  |  | | --- | --- | | money | 8比特数据, 数据是一个有4为小数的decimal | | snallmoney | 4比特数据, 存储的货币值范围比money数据类型小 | |
| 日期 | |  |  | | --- | --- | | date | yyyy-MM-dd | | time  time(3) | hh:mm:ss  hh:mm:ss.000 | | timestamp  timestamp(3) | yyyy-MM-dd hh:mm:ss  yyyy-MM-dd hh:mm:ss.000 | | time with time zone | 与time数据类型相同, 只是还包括专用于UTC和时区的信息. | | timestamp with time zone | 与timestamp数据类型相同, 只是还包括专用于UTC和时区的信息. | |
| 二进制 |  |
| 文本和图形 |  |
| 自定义 |  |

## 三范式

1 第一范式（1NF）: (我的理解: **每一列都是不可分割的数据项**)

在任何一个关系数据库中，第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库。

所谓第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。如果出现重复的属性，就可能需要定义一个新的实体，新的实体由重复的属性构成，新实体与原实体之间为一对多关系。在第一范式（1NF）中表的每一行只包含一个实例的信息。例如，对于图3-2 中的员工信息表，不能将员工信息都放在一列中显示，也不能将其中的两列或多列在一列中显示；员工信息表的每一行只表示一个员工的信息，一个员工的信息在表中只出现一次。简而言之，第一范式就是无重复的列。

2 第二范式（2NF）(我的理解: **要有主键**)

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。

第二范式（2NF）要求数据库表中的每个实例或行必须可以被唯一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的唯一标识。如图3-2 员工信息表中加上了员工编号（emp\_id）列，因为每个员工的员工编号是唯一的，因此每个员工可以被唯一区分。这个唯一属性列被称为主关键字或主键、主码。

第二范式（2NF）要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的唯一标识。简而言之，第二范式就是非主属性非部分依赖于主关键字。

3 第三范式（3NF）(我的理解: **避免数据冗余**)

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，

第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。例如，存在一个部门信息表，其中每个部门有部门编号（dept\_id）、部门名称、部门简介等信息。那么在图3-2的员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部门有关的信息再加入员工信息表中。如果不存在部门信息表，则根据第三范式（3NF）也应该构建它，否则就会有大量的数据冗余。简而言之，第三范式就是属性不依赖于其它非主属性。

# 创建 和 删除 数据库

## 表的种类

|  |  |
| --- | --- |
| **创建永久表**  保存存储在数据库中的SQL数据. | create table |
| **创建全局临时表**  只有在SQL会话的上下文引用该表的定义时, 实际的表才存在, 对话结束后, 表就不存在了.  不能从一个SQL会话访问在另一个会话中创建的表. | create global temporary table |
| **创建局部临时表**  与全局临时表不同的是: 在SQL会话的任何地方都可以访问全局临时表, 而局部临时表只有在相关的SQL模块内才能被访问. | create local temporary table |

## 创建数据库

## 删除数据库

# 数据定义语言DDL

## 创建表

创建基本表

非空约束

指定缺省值

## 修改表

增加新列

删除列

修改列

## 删除表

重命名表

删除表

## 创建索引

创建简单的非族索引

创建多字段非族索引

创建唯一索引

创建族索引

销毁索引

## 创建试图

创建简单的视图

利用视图 简化表的复杂连接

利用视图 简化复杂查询

销毁视图

# 数据查询语言DQL

简单查询

去除重复查询

排序

函数

聚合

多表查询

子查询

# 数据操作语言DML

插入

更新

删除

# 数据控制语言DCL

安全性

完整性

# 其它语言要素

1. 存储过程

2. 触发器

3. 游标

4. 事务控制

5. 嵌入式SQL

# PL/SQL