**新建\打开数据库**  
sqlite3 databasefilename

**sqlite中命令:**  
以.开头,大小写敏感（数据库对象名称是大小写不敏感的）  
.exit退出  
.help 查看帮助 针对命令  
.database 显示数据库信息；包含当前数据库的位置  
.tables 或者 .table 显示表名称  没有表则不显示  
.schema 命令可以查看创建数据对象时的SQL命令；  
.schema databaseobjectname查看创建该数据库对象时的SQL的命令；如果没有这个数据库对象就不显示内容，不会有错误提示  
.read FILENAME 执行指定文件中的SQL语句  
.headers on/off  显示表头 默认off

.mode list|column|insert|line|tabs|tcl|csv   改变输出格式，具体如下

**SQL CREATE TABLE 语法**

CREATE TABLE 表名称

(

列名称1 数据类型,

列名称2 数据类型,

列名称3 数据类型,

....

)

数据类型（data\_type）规定了列可容纳何种数据类型。下面的表格包含了SQL中最常用的数据类型：

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| * integer(size) * int(size) * smallint(size) * tinyint(size) | 仅容纳整数。在括号内规定数字的最大位数。 |
| * decimal(size,d) * numeric(size,d) | 容纳带有小数的数字。  "size" 规定数字的最大位数。"d" 规定小数点右侧的最大位数。 |
| char(size) | 容纳固定长度的字符串（可容纳字母、数字以及特殊字符）。  在括号中规定字符串的长度。 |
| varchar(size) | 容纳可变长度的字符串（可容纳字母、数字以及特殊的字符）。  在括号中规定字符串的最大长度。 |
| date(yyyymmdd) | 容纳日期。 |

**SQL CREATE TABLE 实例**

本例演示如何创建名为 "Person" 的表。

该表包含 5 个列，列名分别是："Id\_P"、"LastName"、"FirstName"、"Address" 以及 "City"：

CREATE TABLE Persons

(

Id\_P int,

LastName varchar(255),

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

Id\_P 列的数据类型是 int，包含整数。其余 4 列的数据类型是 varchar，最大长度为 255 个字符。

**SQL 约束**

约束用于限制加入表的数据的类型。

可以在创建表时规定约束（通过 CREATE TABLE 语句），或者在表创建之后也可以（通过 ALTER TABLE 语句）。

我们将主要探讨以下几种约束：

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* CHECK
* DEFAULT

## SQL NOT NULL 约束

NOT NULL 约束强制列不接受 NULL 值。

NOT NULL 约束强制字段始终包含值。这意味着，如果不向字段添加值，就无法插入新纪录或者更新记录。

下面的 SQL 语句强制 "Id\_P" 列和 "LastName" 列不接受 NULL 值：

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

## SQL PRIMARY KEY 约束

PRIMARY KEY 约束唯一标识数据库表中的每条记录。

主键必须包含唯一的值。

主键列不能包含 NULL 值。

每个表应该都一个主键，并且每个表只能有一个主键。

CREATE TABLE Persons

(

Id\_P int NOT NULL PRIMARY KEY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

## SQL UNIQUE 约束

UNIQUE 约束唯一标识数据库表中的每条记录。

UNIQUE 和 PRIMARY KEY 约束均为列或列集合提供了唯一性的保证。

PRIMARY KEY 拥有自动定义的 UNIQUE 约束。

请注意，每个表可以有多个 UNIQUE 约束，但是每个表只能有一个 PRIMARY KEY 约束。

CREATE TABLE Persons

(

Id\_P int NOT NULL UNIQUE,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

## SQL FOREIGN KEY 约束

一个表中的 FOREIGN KEY 指向另一个表中的 PRIMARY KEY。

**SQLite3没有**

## SQL CHECK 约束

CHECK 约束用于限制列中的值的范围。

如果对单个列定义 CHECK 约束，那么该列只允许特定的值。

如果对一个表定义 CHECK 约束，那么此约束会在特定的列中对值进行限制。

CREATE TABLE Persons

(

Id\_P int NOT NULL CHECK (Id\_P>0),

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

## SQL DEFAULT 约束

DEFAULT 约束用于向列中插入默认值。

如果没有规定其他的值，那么会将默认值添加到所有的新纪录。

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255) DEFAULT 'Sandnes'

)

**CREATE INDEX 语句用于在表中创建索引。**

**在不读取整个表的情况下，索引使数据库应用程序可以更快地查找数据。**

## 索引

您可以在表中创建索引，以便更加快速高效地查询数据。

用户无法看到索引，它们只能被用来加速搜索/查询。

**注释：**更新一个包含索引的表需要比更新一个没有索引的表更多的时间，这是由于索引本身也需要更新。因此，理想的做法是仅仅在常常被搜索的列（以及表）上面创建索引。

### SQL CREATE INDEX 语法

在表上创建一个简单的索引。允许使用重复的值：

CREATE INDEX index\_name

ON table\_name (column\_name)

**注释：**"column\_name" 规定需要索引的列。

### SQL CREATE UNIQUE INDEX 语法

在表上创建一个唯一的索引。唯一的索引意味着两个行不能拥有相同的索引值。

CREATE UNIQUE INDEX index\_name

ON table\_name (column\_name)

## CREATE INDEX 实例

本例会创建一个简单的索引，名为 "PersonIndex"，在 Person 表的 LastName 列：

CREATE INDEX PersonIndex

ON Person (LastName)

如果您希望以**降序**索引某个列中的值，您可以在列名称之后添加保留字 **DESC**：

CREATE INDEX PersonIndex

ON Person (LastName DESC)

假如您希望索引不止一个列，您可以在括号中列出这些列的名称，用逗号隔开：

CREATE INDEX PersonIndex

ON Person (LastName, FirstName)

## SQL DROP INDEX 语句

我们可以使用 DROP INDEX 命令删除表格中的索引。

DROP INDEX index\_name

## ALTER TABLE 语句

ALTER TABLE 语句用于在已有的表中添加、修改或删除列。

### SQL ALTER TABLE 语法

如需在表中添加列，请使用下列语法:

ALTER TABLE table\_name

ADD column\_name datatype

重命名:

ALTER TABLE 旧表名 RENAME TO 新表名

而修改一列无法像其他数据库那样直接以“ALTER TABLE 表名 ADD COLUMN 列名 数据类型”的方式来完成，所以要换种思路，具体步骤看下面

--1.将表名改为临时表

ALTER TABLE "Student" RENAME TO "\_Student\_old\_20140409";

--2.创建新表

CREATE TABLE "Student" (

"Id"  INTEGER PRIMARY KEY AUTOINCREMENT,

"Name"  Text);

--3.导入数据

INSERT INTO "Student" ("Id", "Name") SELECT "Id", "Title" FROM "\_Student\_old\_20140409";

--4.更新sqlite\_sequence

UPDATE "sqlite\_sequence" SET seq = 3 WHERE name = 'Student';

由于在Sqlite中使用自增长字段,引擎会自动产生一个sqlite\_sequence表,用于记录每个表的自增长字段的已使用的最大值，所以要一起更新下。如果有没有设置自增长，则跳过此步骤。

--5.删除临时表(可选)

DROP TABLE \_Student\_old\_20140409;

## AUTO INCREMENT 字段

我们通常希望在每次插入新纪录时，自动地创建主键字段的值。

我们可以在表中创建一个 auto-increment 字段。

CREATE TABLE Persons

(

P\_Id int PRIMARY KEY AUTOINCREMENT,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

## SQL CREATE VIEW 语句

### 什么是视图？

在 SQL 中，视图是基于 SQL 语句的结果集的可视化的表。

视图包含行和列，就像一个真实的表。视图中的字段就是来自一个或多个数据库中的真实的表中的字段。我们可以向视图添加 SQL 函数、WHERE 以及 JOIN 语句，我们也可以提交数据，就像这些来自于某个单一的表。

**注释：**数据库的设计和结构不会受到视图中的函数、where 或 join 语句的影响。

### SQL CREATE VIEW 语法

CREATE VIEW view\_name AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

**注释：**视图总是显示最近的数据。每当用户查询视图时，数据库引擎通过使用 SQL 语句来重建数据。

## SQL CREATE VIEW 实例

可以从某个查询内部、某个存储过程内部，或者从另一个视图内部来使用视图。通过向视图添加函数、join 等等，我们可以向用户精确地提交我们希望提交的数据。

样本数据库 Northwind 拥有一些被默认安装的视图。视图 "Current Product List" 会从 Products 表列出所有正在使用的产品。这个视图使用下列 SQL 创建：

CREATE VIEW CurrentProductList AS

SELECT ProductID,ProductName

FROM Products

我们可以查询上面这个视图：

SELECT \* FROM CurrentProductList

## SQL DROP TABLE 语句

DROP TABLE 语句用于删除表（表的结构、属性以及索引也会被删除）：

DROP TABLE 表名称