开源项目CuteSqlite开发笔记（四）：SQLite字节码引擎(SQLite的Explain详解)

blog.csdn.net/leSneaker/article/details/134962326

1.执行摘要

SQLite的工作原理是将SQL语句转换为字节码，然后在虚拟机中运行该字节码。本文档描述了字节码引擎的工作原理。

本文档描述SQLite内部。使用SQLite进行常规应用程序开发时不需要这里提供的信息。本文档是为那些想要深入研究SQLite内部操作的人准备的。

字节码引擎不是SQLite的API。关于字节码引擎的详细信息从SQLite的一个版本到下一个版本会发生变化。使用SQLite的应用程序不应依赖于本文档中的任何细节。

2.介绍

SQLite的工作原理是将每个SQL语句转换为字节码，然后运行该字节码。SQLite中的预处理语句大多只是实现相应SQL所需的字节码。sqlite3_prepare_v2()接口是一个将SQL转换为字节码的编译器。sqlite3_step()接口是运行预准备语句中包含的字节码的虚拟机。

字节码虚拟机是SQLite的核心。想要了解SQLite内部如何操作的程序员必须熟悉字节码引擎。

在历史上，SQLite中的字节码引擎被称为“虚拟数据库引擎”或“VDBE”。本网站使用术语“字节码引擎”，“VDBE”，“虚拟机”和“字节码虚拟机”可互换，因为它们都意味着同样的事情。

本文还交替使用术语“字节码程序”和“预处理语句”，因为它们基本上是同一个东西。

2.1. VDBE源代码

字节码引擎的源代码位于vdbe. c源文件中。本文档中的操作码定义来自源文件中的注释。源代码注释是有关字节码引擎的规范信息源。如果有疑问，请参考源代码。

除了主要的vdbe. c源代码文件外，源代码树中还有其他帮助器代码文件，它们的名称都以“vdbe”开始-“Virtual DataBase Engine”的缩写。

请记住，操作码的名称和含义通常会从SQLite的一个版本到下一个版本发生变化。因此，如果您正在研究SQLite的EXPLAIN输出，则应该参考与运行EXPLAIN的SQLite版本对应的此文档版本（或vdbe. c源代码）。否则，操作码的描述可能不准确。本文档源自SQLite版本3.44.1签入d295 f48 e8 f367，日期为2023年11月22日。

2.2.指令格式

SQLite中的字节编码程序由一个或多个指令组成。每条指令都有一个操作码和五个操作数，分别命名为P1、P2、P3、P4和P5。P1、P2和P3操作数是32位有符号整数。这些操作数通常指寄存器。对于在b树游标上操作的指令，P1操作数通常是游标编号。对于跳转指令，P2通常是跳转目的地。P4可以是32位有符号整数、64位有符号整数、64位浮点值、字符串文字、Blob文字、指向排序序列比较函数的指针、或指向应用程序定义的SQL函数的实现的指针、或各种其它事物。P5是一个16位无符号整数，通常用于保存标志。P5标志的位有时会以微妙的方式影响操作码。例如，如果P5操作数的SQLITE_NULLEQ（0x0080）位在Eq操作码上设置，则NULL值彼此相等。 否则，NULL值彼此比较不同。

某些操作码使用所有五个操作数。一些操作码使用一个或两个。某些操作码不使用任何操作数。

字节码引擎在指令号0上开始执行。继续执行，直到看到Halt指令，或者直到程序计数器大于最后一条指令的地址，或者直到出现错误。当字节码引擎停止时，它分配的所有内存都将被释放，它可能打开的所有数据库游标都将被关闭。如果执行由于错误而停止，则终止任何挂起的事务，并回滚对数据库所做的更改。

ResultRow操作码会导致字节码引擎暂停，相应的sqlite3_step()调用会返回SQLITE_ROW。在调用ResultRow之前，字节编码的程序将把查询的单行结果加载到一系列寄存器中。诸如sqlite3_column_int()或sqlite3_column_text()之类的C语言API从这些寄存器中提取查询结果。字节码引擎在下一次调用sqlite3_step()时使用ResultRow之后的下一条指令继续执行。

2.3.寄存器

每个字节码程序都有固定（但可能很大）数量的寄存器。一个寄存器可以保存多种对象：

- 空值(NULL value)
- 带符号的64位整数(signed 64-bit integer)
- IEEE双精度（64位）浮点数
- 任意长度的字符串
- 任意长度的BLOB
- RowSet对象（请参见RowSetAdd、RowSetRead和RowSetTest操作码）
- Frame对象（由子程序使用-参见Program）

寄存器也可以是“Undefined”的，这意味着它根本不保存任何值。Undefined不同于NULL。根据编译时选项，试图读取未定义的寄存器通常会导致运行时错误。如果代码生成器（sqlite3_prepare_v2()）生成了一个读取Undefined寄存器的预处理语句，那么这就是代码生成器中的一个bug。

寄存器的编号从0开始。大多数操作码至少引用一个寄存器。

单个预准备语句中的寄存器数量在编译时是固定的。当一个预备语句被复位或终结时，所有寄存器的内容都被清除。

内部Mem对象存储单个寄存器的值。API中公开的抽象sqlite3_value对象实际上只是一个Mem对象或寄存器。

2.4. B树游标

预准备语句可以有零个或多个打开游标。每个游标由一个小整数标识，该整数通常是使用游标的操作码的P1参数。在同一索引或表上可以打开多个游标。所有游标都独立操作，即使游标指向相同的索引或表。虚拟机与数据库文件交互的唯一方式是通过游标。虚拟机中的指令可以创建新游标（例如：OpenRead或OpenWrite），从游标（列）读取数据，将游标前进到表中的下一个条目（例如：Next或Prev），等等。重置或终止预准备语句时，所有游标都将自动关闭。

2.5.子例程、协程和子程序

字节码引擎没有堆栈来存储子例程的返回地址。返回地址必须存储在寄存器中。因此，字节码子例程不是可重入的。

Gosub操作码将当前程序计数器存储到寄存器P1中，然后跳转到地址P2。返回操作码跳转到地址P1+1。因此，每个子例程都与两个整数相关联：子例程中入口点的地址和用于保存返回地址的寄存器编号。

Yield操作码将程序计数器的值与寄存器P1中的整数值交换。此操作码用于实现协程。协同程序通常用于实现子查询，根据需要从子查询中提取内容。

触发器需要可重入。由于字节码子例程是不可重入的，因此必须使用不同的机制来实现触发器。每个触发器都使用单独的字节码程序实现，该程序具有自己的操作码、程序计数器和寄存器集。Program操作码调用触发器子程序。Program指令为子程序的每次调用分配并重新分配一个新的寄存器集，因此子程序可以是可重入和递归的。子程序使用Param操作码来访问调用字节码程序的寄存器中的内容。

2.6.自更改代码

一些操作码是自我改变的。例如，Init操作码（总是每个字节码程序中的第一个操作码）递增其P1操作数。随后的Once操作码将其P1操作数与Init操作码的P1值进行比较，以确定是否应跳过随后的一次性初始化代码。另一个例子是String 8操作码，它将P4操作数从UTF-8转换为正确的数据库字符串编码，然后将自身转换为String操作码。

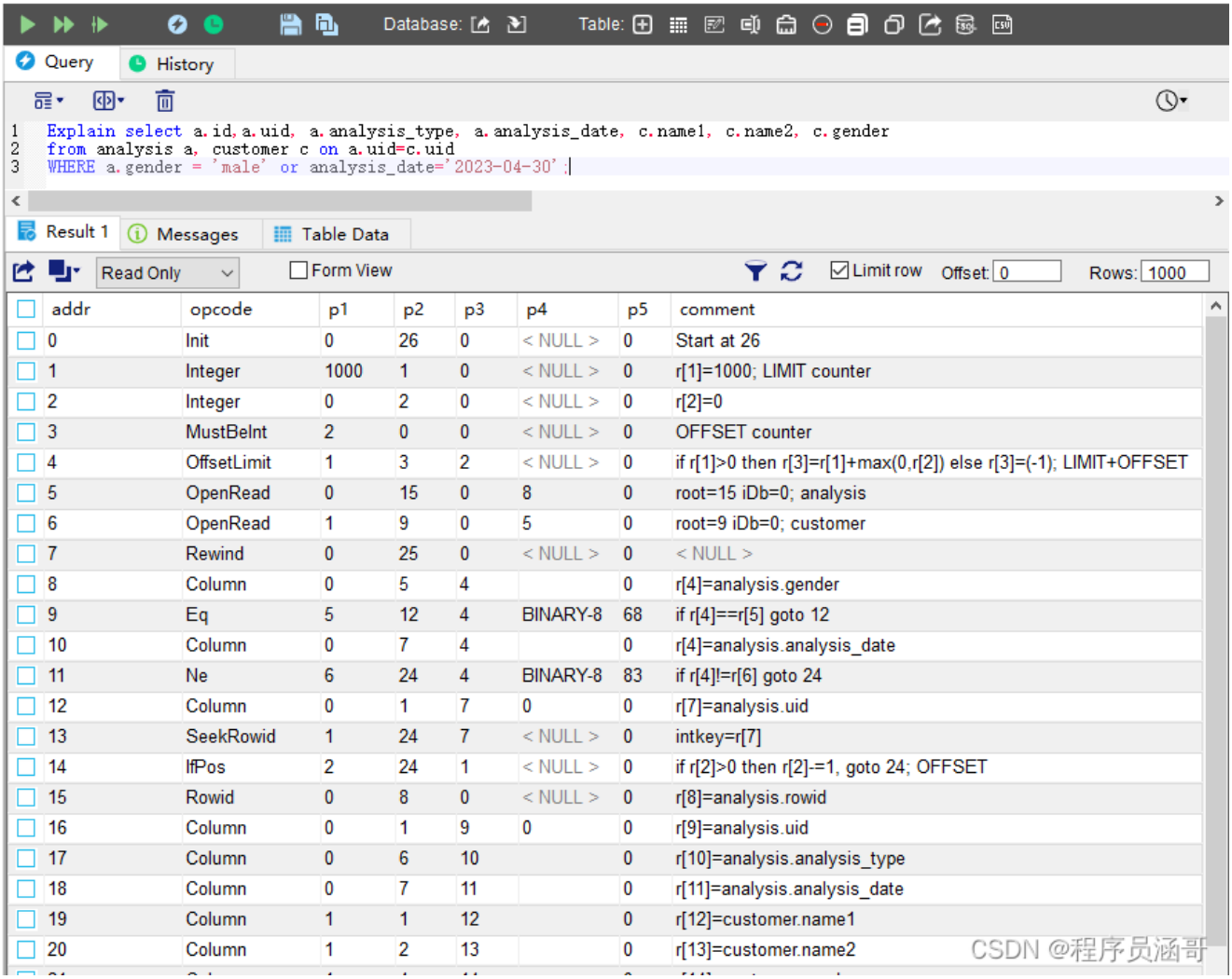
3.查看字节码

SQLite解释的每个SQL语句都会产生一个用于虚拟机的程序。但是如果SQL语句以关键字EXPLAIN开头，虚拟机将不会执行该程序。相反，程序的指令将被返回，每行一条指令，就像查询结果一样。此功能对于调试和了解虚拟机的操作方式非常有用。举例来说：

```
$ sqlite3 ex1.db
sqlite> explain delete from tbl1 where two<20;
addr  opcode         p1    p2    p3    p4             p5  comment
----  -------------  ----  ----  ----  -------------  --  -------------
0     Init           0     12    0                    00  Start at 12
1     Null           0     1     0                    00  r[1]=NULL
2     OpenWrite      0     2     0     3              00  root=2 iDb=0; tbl1
3     Rewind         0     10    0                    00
4       Column         0     1     2                    00  r[2]=tbl1.two
5       Ge             3     9     2     (BINARY)       51  if r[2]>=r[3] goto 9
6       Rowid          0     4     0                    00  r[4]=rowid
7       Once           0     8     0                    00
8       Delete         0     1     0     tbl1           02
9     Next           0     4     0                    01
10    Noop           0     0     0                    00
11    Halt           0     0     0                    00
12    Transaction    0     1     1     0              01  usesStmtJournal=0
13    TableLock      0     2     1     tbl1           00  iDb=0 root=2 write=1
14    Integer        20    3     0                    00  r[3]=20
15    Goto           0     1     0                    00
```

任何应用程序都可以运行EXPLAIN查询来获得类似于上面的输出。然而，显示循环结构的缩进不是由SQLite核心生成的。命令行shell包含用于缩进循环的额外逻辑。此外，EXPLAIN输出中的“comment”列仅在使用-DSQLITE_ENABLE_EXPLAIN_COMMENTS选项编译SQLite时才提供。

(译者注：CuteSqlite编译加上了-DSQLITE_ENABLE_EXPLAIN_COMMENTS，可以查看Explain的comment字段)



下载地址：https://github.com/shinehanx/CuteSqlite/releases

当使用SQLITE_DEBUG编译时选项编译SQLite时，可以使用额外的PRAGMA命令，这些命令对于调试和探索VDBE的操作非常有用。例如，可以启用vdbe_trace杂注，以便在执行操作码时在标准输出上打印每个VDBE操作码的反汇编。这些调试杂注包括：

4.操作码

目前，虚拟机定义了187个操作码。下表描述了所有当前定义的操作码。此表是通过扫描文件vdbe. c中的源代码自动生成的。

记住：VDBE操作码不是SQLite接口定义的一部分。从SQLite的一个版本到下一个版本，操作码的数量及其名称和含义都会发生变化。下表中显示的操作码适用于日期为2023-11-22的SQLite版本3.44.1签入d295 f48 e8 f367。

操作码名称	描述
Abortable	验证是否可以发生中止。如果此时的Abort可能导致数据库损坏，请断言。此操作码仅出现在调试版本中。 如果没有写操作，或者存在活动语句日志，则中止是安全的。
Add	将寄存器P1中的值与寄存器P2中的值相加，并将结果存储在寄存器P3中。如果任何一个输入为NULL，则结果为NULL。
AddImm	将常数P2与寄存器P1中的值相加。结果总是一个整数。 要强制任何寄存器为整数，只需添加0。
Affinity	从P1开始，对P2寄存器范围应用亲和性。 P4是一个P2个字符长的字符串。该字符串的第N个字符表示应该用于该范围中的第N个存储单元的列亲和性。
AggFinal	P1是作为聚合或窗口函数的累加器的内存位置。执行聚合的终结器函数，并将结果存储在P1中。 P2是step函数采用的参数数量，P4是指向此函数的FuncDef的指针。此操作码不使用P2参数。它只是用来消除可以接受不同数量参数的函数的歧义。P4参数仅在先前未调用step函数的情况下才需要。
AggInverse	对一个聚合执行xInverse函数。该函数有P5参数。P4是指向指定函数的FuncDef结构的指针。寄存器P3是累加器。 P5参数取自寄存器P2及其后继寄存器。
AgStep	对一个聚合执行xStep函数。该函数有P5参数。P4是指向指定函数的FuncDef结构的指针。寄存器P3是累加器。 P5参数取自寄存器P2及其后继寄存器。
AggStep	执行xStep（如果P1==0）或xInverse（如果P1！0)函数的集合。该函数有P5参数。P4是指向指定函数的FuncDef结构的指针。寄存器P3是累加器。 P5参数取自寄存器P2及其后继寄存器。 此操作码最初编码为OP_AggStep0。在第一次评估时，存储在P4中的FuncDef被转换为sqlite3_context并更改操作码。这样，sqlite3_context的初始化只发生一次，而不是在每次调用step函数时。

AggStep1	调用xValue（）函数并将结果存储在寄存器P3中。 P2是step函数采用的参数数量，P4是指向此函数的FuncDef的指针。此操作码不使用P2参数。它只是用来消除可以接受不同数量参数的函数的歧义。P4参数仅在先前未调用step函数的情况下才需要。
And	对寄存器P1和P2中的值进行逻辑与运算，并将结果写入寄存器P3。 如果P1或P2为0（false），则结果为0，即使另一个输入为NULL。NULL和true或两个NULL给予NULL输出。
AutoCommit	将数据库自动提交标志设置为P1（1或0）。如果P2为true，则回滚任何当前活动的btree事务。如果有任何活动的VM（除此之外），则ROLLBACK失败。如果存在活动写入VM或使用共享缓存的活动VM，则COMMIT失败。 此指令将导致VM停止。
BeginSubrtn	标记子例程的开始，该子例程可以在线输入或可以使用Gosub调用。子例程应该由返回指令终止，该返回指令的P1操作数与此操作码的P2操作数相同，并且P3设置为1。如果子例程是以内联方式输入的，那么返回将简单地失败。但是如果子例程是使用Gosub输入的，那么Return将跳回到Gosub之后的第一条指令。 此例程的工作原理是将NULL加载到P2寄存器中。当返回地址寄存器包含NULL时，返回指令是一个空操作，只是福尔斯下降到下一个指令（假设返回操作码的P3值为1）。因此，如果子例程是以内联方式输入的，则返回将导致内联执行继续。但是如果子程序是通过Gosub进入的，那么Return将导致返回Gosub后面的地址。 此操作码与EXP相同。它有一个不同的名称，只是为了使字节码更容易阅读和验证。
BitAnd	对寄存器P1和P2中的值进行逐位AND运算，并将结果存储在寄存器P3中。如果任何一个输入为NULL，则结果为NULL。
BitNot	将寄存器P1的内容解释为整数。将P1值的一补数存储到寄存器P2中。如果P1持有NULL，则在P2中存储NULL。
BitOr	对寄存器P1和P2中的值进行按位或运算，并将结果存储在寄存器P3中。如果任何一个输入为NULL，则结果为NULL。
Blob	P4指向P1字节长的数据blob。将此blob存储在寄存器P2中。如果P4是NULL指针，则在P2中构造一个P1字节长的填充零的blob。

Cast	强制寄存器P1中的值为P2定义的类型。 • P2=='A' → BLOB • P2=='B' → TEXT • P2=='C' → NUMERIC • P2=='D' → INTEGER • P2=='E' → REAL 此例程不会更改NULL值。它仍然是NULL。
Checkpoint	检查点数据库P1。如果P1当前未处于WAL模式，则这是无操作。参数P2是SQLITE_CLKPOINT_PASSIVE、FULL、RESTART或TRUNCATE之一。如果检查点返回SQLITE_忙碌或未返回，则分别将1或0写入到P3中。将检查点之后WAL中的页面数写入到[P3+1]中，并将检查点完成后WAL中已被检查的页面数写入到[P3+2]中。但是，如果出现错误，则会将[P3+1]和[P3+2]初始化为-1。
Clear	删除数据库文件中根页由P1给出的数据库表或索引的所有内容。但是，与Destroy不同的是，不要从数据库文件中删除表或索引。如果P2==0，则被清除的表位于主数据库文件中。如果P2==1，则要清除的表位于辅助数据库文件中，该文件用于存储使用CREATE TEMPORARY TABLE创建的表。 如果P3值为非零，则行更改计数将增加表中被清除的行数。如果P3大于零，则存储在寄存器P3中的值也增加表中被清除的行数。 别名：Destroy
Close	关闭以前作为P1打开的游标。如果P1当前未打开，则此指令为空操作。
ClrSubtype	从寄存器P1中清除子类型。
CollSeq	P4是指向CollSeq对象的指针。如果对用户函数或聚合函数的下一次调用调用sqlite3GetFuncCollSeq（），则将返回此排序规则序列。这被内置的min（）、max（）和nullif（）函数使用。 如果P1不为零，则它是一个寄存器，如果当前行不是最小值或最大值，则随后的min（）或max（）聚合将设置为1。此指令将P1寄存器初始化为0。 实现上述函数所使用的接口（用于检索此操作码设置的排序规则序列）并不公开。只有内置函数才能访问此功能。

Column	将游标P1指向的数据解释为使用MakeRecord指令构建的结构。(See MakeRecord操作码用于获取有关数据格式的其他信息。）从该记录中提取第P2列。如果记录中的值小于（P2+1），则提取NULL。 所提取的值存储在寄存器P3中。 如果记录包含的字段少于P2，则提取NULL。或者，如果P4参数是P4_MEM，则使用P4参数的值作为结果。 如果在P5中设置了OPFLAG_LENGTHARG位，则保证结果仅由length（）函数或等效函数使用。不加载大blob的内容，从而节省CPU周期。如果设置了OPFLAG_TYPEOFARG位，则结果将仅由typeof（）函数或IS NULL或IS NOT NULL运算符或等效运算符使用。在这种情况下，可以省略所有内容加载。
ColumnsUsed	此操作码（仅在SQLite使用SQLITE_ENABLE_COLUMN_USED_MASK编译时存在）标识使用游标P1的表或索引的哪些列。P4是一个64位整数（P4_INT64），其中前63位是游标实际使用的表或索引的前63列中的每一列。如果使用第64位之后的任何列，则设置高阶位。
Compare	比较reg（P1）中寄存器的两个向量。reg（P1+P3-1）（将此向量称为“A”）和在reg（P2）中。reg（P2+P3-1）（“B”）。保存比较结果，供下一个跳转指令使用。 如果P5设置了OPFLAG_PERMUTE位，则比较顺序由最近的Permutation运算符确定。如果OPFLAG_PERMUTE位清零，则按顺序比较寄存器。 P4是一个KeyInfo结构，它定义了比较的排序序列和排序顺序。该排列仅适用于寄存器。KeyInfo元素按顺序使用。 比较是排序比较，所以NULL比较相等，NULL小于数字，数字小于字符串，字符串小于blob。 此操作码必须紧跟一个Jump操作码。
Concat	将寄存器P1中的文本添加到寄存器P2中的文本末尾，并将结果存储在寄存器P3中。如果P1或P2文本为NULL，则将NULL存储在P3中。 P3 = P2|| P1 P1和P3是同一寄存器是非法的。有时候，如果P3和P2是同一个寄存器，实现就可以避免memcpy（）。

Copy	复制寄存器P1.. P1+P3到寄存器P2中。P2+P3。 如果P5的0x0002位被设置，则还清除目标中的MEM_Subtype标志。P5的0x0001位表示此复制操作码不能合并。0x0001位由查询规划器使用，在查询执行期间不起作用。 此指令生成值的深层副本。副本由任何字符串或blob常量组成。这就是SCOPY。
Count	将游标P1打开的表或索引中的条目数（整数值）存储在寄存器P2中。 如果P3==0，则获得精确计数，这涉及访问表的每个btree页。但如果P3非零，则基于当前光标位置返回估计值。
Btree	如果P1==0，则在主数据库文件中分配新的b树;如果P1==1，则在TEMP数据库文件中分配新的b树;如果P1>1，则在附加数据库中分配新的b树。对于rowid表，P3参数必须为1（BTREE_INTKEY）；对于索引或WITHOUT ROWID表，P3参数必须为2（BTREE_BLOBKEY）。新b树的根页号存储在寄存器P2中。
CursorHint	向游标P1提供一个提示，它只需要返回满足P4中Expr的行。P4表达式中的TK_REGISTER项指的是当前保存在寄存器中的值。P4表达式中的TK_COLUMN项引用游标P1所指向的b树中的列。
CursorLock	锁定游标P1所指向的btree，使其他游标无法写入该btree。
CursorUnlock	删除游标P1所指向的btree，以便其他游标可以写入它。
DecrJumpZero	寄存器P1必须保存整数。递减P1中的值，如果新值正好为零，则跳转到P2。
DeferredSeek	P1是打开索引游标，P3是相应表上的游标。此操作码将P3表游标延迟查找到与P1当前行对应的行。 这是一个延迟的搜索。在使用游标读取记录之前，实际上什么都不会发生。这样，如果没有读操作发生，就不会发生不必要的I/O。 P4可以是整数数组（类型P4_INTRAY），P3表中的每一列都包含一个条目。如果数组项a（i）为非零，则从游标P3阅读列a（i）-1等效于执行延迟寻道，然后从P1阅读列i。该信息存储在P3中，并用于将对P3的读取重定向到P1，从而可能避免查找和读取游标P3的需要。

Delete	删除P1光标当前指向的记录。 如果设置了P5参数的OPFLAG_SAVEPOSITION位，则光标将指向表中的下一条或上一条记录。如果它指向下一条记录，那么下一条Next指令将是一个空操作。因此，在这种情况下，可以从Next循环中删除一条记录。如果P5的OPFLAG_SAVEPOSITION位被清除，则光标将处于未定义状态。 如果在P5上设置了OPFLAG_AUXOND位，则表明此删除是与删除表行及其所有关联索引项相关联的几个删除之一。这些删除中的恰好一个是“主要”删除。其他的都在OPFLAG_FORWARD游标上，或者用AUX标志标记。 如果设置了P2的OPFLAG_NCHANGE（0x01）标志（NB：P2而不是P5），则递增行改变计数（否则不递增）。 如果P2（不是P5！）的OPFLAG_ISNOOP（0x 40）标志则运行用于删除的pre-update-hook，但btree在其他方面保持不变。当Delete之后紧跟着一个具有相同键的Insert时，会发生这种情况，从而导致btree条目被覆盖。 P1不能是伪表。它必须是一个具有多行的真实的表。 如果P4不为NULL，则它指向Table对象。在这种情况下，可以调用更新挂钩或预更新挂钩，或者同时调用两者。在这种情况下，在调用此操作码之前，必须使用NotFound定位P1游标。具体地说，如果配置了一个钩子，则在P4不为NULL时调用pre-update钩子。如果配置了一个update-hook，P4不为NULL，并且在P2中设置了OPFLAG_NCHANGE标志，则调用该update-hook。 如果在P2中设置了OPFLAG_ISUPDATE标志，则P3包含存储器单元的地址，该存储器单元包含行的rowid将通过更新被设置为的值。

Destroy	删除数据库文件中根页由P1给出的整个数据库表或索引。 如果P3==0，则被销毁的表位于主数据库文件中。如果P3==1，则要销毁的表位于辅助数据库文件中，该文件用于存储使用CREATE TEMPORARY TABLE创建的表。 如果启用了AUTOVACUUM，则可能会将另一个根页移动到新删除的根页中，以使所有根页在数据库的开头保持连续。被移动的根页的前一个值（其在移动发生之前的值）存储在寄存器P2中。如果不需要页面移动（因为被删除的表已经是数据库中的最后一个表），则在寄存器P2中存储零。如果AUTOVACUUM被禁用，则寄存器P2中存储一个零。 如果调用此操作码时存在任何活动的读取器VM，则会抛出错误。这样做是为了避免在AUTOVACUUM数据库中移动根页面时更新现有游标所带来的困难。即使数据库不是AUTOVACUUM数据库，也会抛出此错误，以避免在自动真空和非自动真空模式之间引入不兼容性。 标签：Clear
Divide	将寄存器P1中的值除以寄存器P2中的值，并将结果存储在寄存器P3中（P3=P2/P1）。如果寄存器P1中的值为零，则结果为NULL。如果任何一个输入为NULL，则结果为NULL。
DropIndex	删除描述数据库P1中名为P4的索引的内部（内存中）数据结构。这是在从磁盘删除索引后调用的（使用Destroy操作码），以保持模式的内部表示与磁盘上的一致。
DropTable	删除描述数据库P1中名为P4的表的内部（内存中）数据结构。这是在从磁盘中删除表后调用的（使用Destroy操作码），以保持模式的内部表示与磁盘上的一致。
DropTrigger	删除数据库P1中描述名为P4的触发器的内部（内存中）数据结构。这是在从磁盘中删除触发器后调用的（使用Destroy操作码），以保持模式的内部表示与磁盘上的一致。
ElseEq	此操作码必须跟在Lt或Gt比较运算符之后。可以有零个或多个OP_ReleaseReg操作码介入，但在此指令与前一个Lt或Gt之间不允许出现其他操作码。 如果在与先前的Lt或Gt相同的两个操作数上的Eq比较的结果为真，则跳转到P2。如果前面两个操作数的Eq比较结果为false或NULL，则失败。

EndCoroutine	寄存器P1中地址处的指令是Yield。跳转到该Yield的P2参数。跳转后，寄存器P1变为未定义。 主条目：InitCoroutine
Eq	比较寄存器P1和P3中的值。如果reg（P3）==reg（P1），则跳转到地址P2。 P5的SQLITE_AFF_MASK部分必须是关联字符-SQLITE_AFF_TEXT、SQLITE_AFF_INTEGER等。在进行比较之前，会尝试根据此亲和性强制两个输入。如果SQLITE_AFF_MASK为0x 00，则使用数值关联。注意，相似性转换被存储回输入寄存器P1和P3。因此，此操作码可能会导致寄存器P1和P3的持久更改。 一旦发生了任何转换，并且两个值都不是NULL，则比较这些值。如果两个值都是blob，则使用memcmp（）确定比较结果。如果两个值都是文本，则使用P4中指定的相应排序函数进行比较。如果没有指定P4，则使用memcmp（）比较文本字符串。如果两个值都是数值，则使用数值比较。如果这两个值的类型不同，那么数字被认为小于字符串，字符串被认为小于blob。 如果在P5中设置了SQLITE_NULLEQ，那么比较的结果总是true或false，并且永远不会为NULL。如果两个操作数都为NULL，则比较结果为true。如果任一操作数为NULL，则结果为false。如果两个操作数都不为NULL，则结果与从P5中省略SQLITE_NULLEQ标志时的结果相同。 此操作码保存比较结果以供新的Jump操作码使用。
Expire	导致预编译语句过期。当使用sqlite3_step（）执行过期的语句时，它将自动重新准备自己（如果它最初是使用sqlite3_step_v2（）创建的），或者它将因SQLITE_SCHEMA而失败。 如果P1为0，则所有SQL语句都将过期。如果P1不为零，则只有当前正在执行的语句过期。 如果P2为0，则SQL语句立即过期。如果P2为1，则允许正在运行的SQL语句继续运行直至完成。P2==1的情况发生在CREATE INDEX或类似的模式更改发生时，这可能有助于语句运行得更快，但不会影响操作的正确性。

Filter	从r[P3]开始计算P4寄存器中包含的密钥的散列。检查是否在寄存器P1托管的布隆过滤器中找到该散列。如果不存在，则可以跳到P2。否则就泡汤了。 假阴性是无害的。即使值在布隆过滤器中，失败也总是安全的。假阴性会导致更多的CPU周期被使用，但它仍然应该产生正确的答案。然而，一个错误的答案很可能来自一个假阳性-如果跳跃是在它应该失败的时候进行的。
FilterAdd	从r[P3]开始计算P4寄存器上的散列，并将该散列添加到r[P1]中包含的布隆过滤器。
FinishSeek	如果光标P1先前通过DeferredSeek移动，则立即完成该寻道操作，不再延迟。如果游标寻道已经发生，则此指令为空操作。
FkCheck	如果存在任何未解决的外键约束冲突，则停止并返回SQLITE_CONSTRAINT错误。如果没有外键约束冲突，这是一个空操作。 当预准备语句退出时，也会检查FK约束冲突。此操作码用于在返回结果（如行更改计数或RETURNING子句的结果）之前引发外键约束错误。
FkCounter	将“约束计数器”递增P2（P2可以是负的或正的）。如果P1非零，则数据库约束计数器递增（延迟外键约束）。否则，如果P1为零，则语句计数器递增（立即外键约束）。
FkIfZero	此操作码测试外键约束计数器当前是否为零。如果是，跳转到指令P2。否则，继续执行下一条指令。 如果P1不为零，则在数据库约束计数器为零（对延迟约束违反进行计数）时进行跳转。如果P1为零，则在语句constraint-counter为零时进行跳转（立即外键约束违反）。
Found	如果P4==0，则寄存器P3保存由MakeRecord构造的blob。如果P4>0，则寄存器P3是形成解包记录的P4个寄存器中的第一个。 游标P1位于索引btree上。如果由P3和P4标识的记录是P1中任何条目的前缀，则跳转到P2，并且P1左指向匹配条目。 此操作使光标处于可以向前移动的状态。Next指令可以工作，但Prev指令不行。 另请参阅：NotFound、NoConflict、NotFound。搜索结果

Function	使用从寄存器P2和后继寄存器中获取的参数创建用户函数（P4是指向sqlite3_context对象的指针，该对象包含指向要运行的函数的指针）。参数的数量在P4指向的sqlite3_context对象中。该函数的结果存储在寄存器P3中。寄存器P3不能是功能输入之一。 P1是一个32位位掩码，指示函数的每个参数是否在编译时被确定为常量。如果第一个参数是常量，则P1的位0被设置。这用于确定与使用sqlite3_set_auxdata（）API的用户函数参数相关联的Meta数据是否可以安全地保留，直到下一次调用此操作码。 另请参阅：AggStep、AggFinal、PureFunc
Ge	这就像Lt操作码一样工作，只是如果寄存器P3的内容大于或等于寄存器P1的内容，则进行跳转。有关更多信息，请参见Lt操作码。
Gosub	将当前地址写入寄存器P1，然后跳转到地址P2。
Goto	无条件跳转到地址P2。执行的下一条指令将是从程序开始的索引P2处的指令。 P1参数实际上没有被这个操作码使用。但是，它有时会被设置为1而不是0，作为对命令行shell的一个提示，即这个后藤是循环的底部，并且从P2到当前行的行应该缩进，以便EXPLAIN输出。
Gt	这就像Lt操作码一样工作，只是如果寄存器P3的内容大于寄存器P1的内容，则进行跳转。有关更多信息，请参见Lt操作码。
Halt	立即退出。所有打开的光标等都自动关闭。 P1是sqlite3_exec（）、sqlite3_reset（）或sqlite3_finalize（）返回的结果代码。对于正常的暂停，这应该是SQLITE_OK（0）。对于错误，它可以是其他值。如果P1！= 0，则P2将决定是否回滚当前事务。如果P2==OE_Fail，则不回滚。如果P2==OE_Rollback，则执行回滚。如果P2==OE_Abort，则回退在执行VDBE期间发生的所有更改，但不回滚事务。 如果P4不为空，则它是一个错误消息字符串。 P5是一个介于0和4之间的值，它修改P4字符串。 0：（无更改）1：NOT NULL约束失败：P4 2：UNIQUE约束失败：P4 3：NULL约束失败：P4 4：FOREIGN KEY约束失败：P4 如果P5不为零且P4为NULL，则省略“：”之后的所有内容。 在每个程序的最后都插入了一条隐含的“Halt 0 0 0”指令。因此，跳过程序的最后一条指令与执行Halt相同。

HaltIfNull	检查寄存器P3中的值。如果它为NULL，则使用参数P1、P2和P4暂停，就好像这是一条暂停指令。如果寄存器P3中的值不为NULL，则此例程为空操作。P5参数应为1。
IdxDelete	从寄存器P2开始的P3寄存器的内容形成解压缩的索引键。此操作码从游标P1打开的索引中删除该条目。 如果P5不为零，则在没有找到匹配的索引条目时引发SQLITE_CORRUPT_INDEX错误。当运行UPDATE或UPDATE语句并且没有找到要更新或删除的索引条目时，会发生这种情况。对于IdxDelete的某些用途（例如：EXCEPT运算符），没有找到匹配条目并不重要。对于这些情况，P5为零。另外，如果在writable_schema模式下，请不要引发此（自纠正和非关键）错误。
IdxGE	以P3开始的P4寄存器值形成省略PRIMARY KEY的解压缩索引键。将此键值与P1当前指向的索引进行比较，忽略末尾的PRIMARY KEY或ROWID字段。 如果P1索引条目大于或等于键值，则跳转到P2。否则，执行下一条指令。
IdxGT	以P3开始的P4寄存器值形成省略PRIMARY KEY的解压缩索引键。将此键值与P1当前指向的索引进行比较，忽略末尾的PRIMARY KEY或ROWID字段。 如果P1索引条目大于键值，则跳转到P2。否则，执行下一条指令。

IdxInsert	寄存器P2保存使用MakeRecord指令生成的SQL索引键。这个操作码将该键写入索引P1。条目的数据为nil。 如果P4不为零，则它是reg（P2）的解包键中的值的数量。在这种情况下，P3是用于解包密钥的第一寄存器的索引。解包密钥的可用性有时可以是一种优化。 如果P5设置了OPFLAG_APPEND位，则这是对b树层的一个提示，表明此插入可能是一个追加。 如果P5设置了OPFLAG_NCHANGE位，则此指令将使更改计数器递增。如果OPFLAG_NCHANGE位清零，则更改计数器不变。 如果设置了P5的OPFLAG_USESEEKRESULT标志，则通过避免游标P1上不必要的寻道，实现可能会运行得更快。但是，只有在游标上没有先前的寻道或最近的寻道使用了与P2等效的键时，才必须设置OPFLAG_USESEEKRESULT标志。 此指令仅适用于索引。表的等效指令是插入。
IdxLE	以P3开头的P4寄存器值形成省略PRIMARY KEY或ROWID的解压缩索引键。将此键值与P1当前指向的索引进行比较，忽略P1索引上的PRIMARY KEY或ROWID。 如果P1索引条目小于或等于键值，则跳转到P2。否则，执行下一条指令。
IdxLT	以P3开头的P4寄存器值形成省略PRIMARY KEY或ROWID的解压缩索引键。将此键值与P1当前指向的索引进行比较，忽略P1索引上的PRIMARY KEY或ROWID。 如果P1索引条目小于键值，则跳转到P2。否则，执行下一条指令。
IdxRowid	在寄存器P2中写入一个整数，该整数是游标P1指向的索引关键字末尾的记录中的最后一个条目。这个整数应该是这个索引条目指向的表条目的rowid。 参见：Rowid，MakeRecord。
If	如果寄存器P1中的值为真，则跳转到P2。如果该值是数值且非零，则将其视为true。如果P1中的值为NULL，则当且仅当P3为非零时进行跳转。

IfNoHope	寄存器P3是P4寄存器中的第一个，它形成一个解压缩的记录。游标P1是一个索引btree。P2是跳跃目的地。换句话说，此操作码的操作数与NotFound和IdxGT的操作数相同。 此操作码只是一种优化尝试。如果这个操作码总是福尔斯，仍然可以得到正确的答案，但是需要执行额外的工作。 游标P1的seekHit标志中的值N意味着存在一个键P3：N，它将匹配索引中的某个记录。我们想知道记录P3：P4是否可能匹配索引中的某个记录。如果不可能，我们可以跳过一些工作。因此，如果seekHit小于P4，则尝试通过运行NotFound来查找是否可能匹配。 此操作码用于多列键的IN子句处理。如果一个IN子句被附加到键的一个元素而不是最左边的元素，并且如果在整个键上最近一次查找没有匹配，那么可能是左边的一个键元素禁止了匹配，因此无论检查多少IN子句元素，都“没有希望”任何匹配。在这种情况下，我们使用此操作码提前放弃IN子句搜索。操作码的名称来自这样一个事实，即如果“没有希望”实现匹配，则进行跳转。 标签：NotFound，SeekHit
IfNot	如果寄存器P1中的值为假，则跳转到P2。如果该值的数值为零，则将其视为false。如果P1中的值为NULL，则当且仅当P3为非零时进行跳转。
IfNotOpen	如果游标P1未打开，或者如果使用NullRow操作码将P1设置为NULL行，则跳转到指令P2。否则，失败。
IfNotZero	寄存器P1必须包含整数。如果寄存器P1的内容最初大于零，则递减寄存器P1中的值。如果它是非零（负或正），然后也跳到P2。 如果寄存器P1最初为零，则保持不变并下降。
IfNullRow	检查游标P1，看看它当前是否指向NULL行。如果是，则将寄存器P3设置为NULL并立即跳转到P2。如果P1不在NULL行上，则不做任何更改。 如果P1不是一个打开的游标，那么这个操作码就是一个no-op。
IfPos	寄存器P1必须包含整数。如果寄存器P1的值为1或更大，则从P1中的值减去P3并跳转到P2。 如果寄存器P1的初始值小于1，则该值不变，控制传递到下一条指令。
IfSmaller	估计表P1中的行数。如果估计值小于约2**（0.1*P3），则跳转到P2。

IncrVacuum	在P1数据库上执行增量真空程序的单个步骤。如果真空已完成，跳转至指令P2。否则，继续执行下一条指令。
Init	程序包含此操作码的单个实例作为第一个操作码。 如果（通过sqlite3_trace（））接口启用了跟踪，则在跟踪回调中发出P4中包含的UTF-8字符串。如果P4为空，则使用sqlite3_sql（）返回的字符串。 如果P2不为零，跳转到指令P2。 递增P1的值，以便Once操作码在第一次为此运行进行评估时跳转。 如果P3不为零，则它是在遇到SQLITE_CORRUPT错误时要跳转到的地址。
InitCoroutine	设置寄存器P1，使其屈服于位于地址P3的协程。 如果P2！0，则协程实现将立即遵循此操作码。所以跳过协程实现来处理P2。 参见：EndCoroutine

Insert	在游标P1的表中写入一个条目。如果新条目不存在或现有条目的数据被覆盖，则会创建新条目。数据是存储在寄存器P2中的MEM_Blob值。密钥存储在寄存器P3中。键必须是MEM_Int。 如果设置了P5的OPFLAG_NCHANGE标志，则递增行改变计数（否则不递增）。如果设置了P5的OPFLAG_LASTROWID标志，则存储rowid以供sqlite3_last_insert_rowid（）函数随后返回（否则不修改）。 如果设置了P5的OPFLAG_USESEEKRESULT标志，则通过避免游标P1上不必要的寻道，实现可能会运行得更快。但是，只有在游标上没有先前的寻道或最近的寻道使用了等于P3的键时，才必须设置OPFLAG_USESEEKRESULT标志。 如果设置了OPFLAG_ISUPDATE标志，则此操作码是UPDATE操作的一部分。否则（如果标志是清除的），则此操作码是一个重复操作的一部分。区别只对更新钩子重要。 参数P4可以指向表结构，或者可以是NULL。如果它不是NULL，则在成功插入之后调用更新钩子（sqlite3.xUpdateCallback）。 （备注/TODO：如果P1是一个伪游标，而P2是动态分配的，则P2的所有权被转移到伪游标，寄存器P2变为临时寄存器。如果光标改变，则寄存器P2的值将随之改变。确保这不会导致任何问题）。 此指令仅适用于表。索引的等效指令是IdxInsert。
Int64	P4是指向64位整数值的指针。将该值写入寄存器P2。
IntCopy	将寄存器P1中保存的整数值转移到寄存器P2中。 这是SCopy的优化版本，仅适用于整数值。
Integer	32位整数值P1被写入寄存器P2。

IntegrityCk	分析当前打开的数据库。在寄存器P1中存储描述任何问题的错误消息的文本。如果没有发现问题，则在寄存器P1中存储NULL。寄存器P3包含比最大允许错误数小1的错误。最多报告reg（P3）错误。换句话说，一旦发现reg（P1）错误，分析就会停止。Reg（P1）将更新剩余的错误数。 数据库中所有表的根页编号都是存储在P4_INTRAY参数中的整数。 如果P5不为零，则对辅助数据库文件而不是主数据库文件进行检查。 此操作码用于实现integrity_check杂注。
IsNull	如果寄存器P1中的值为NULL，则跳转到P2。
IsTrue	此操作码实现IS TRUE、IS NOT TRUE、IS NOT TRUE和IS NOT NOT TRUE运算符。 将寄存器P1中的值解释为布尔值。将该布尔值（0或1）存储在寄存器P2中。或者，如果寄存器P1中的值为NULL，则将P3存储在寄存器P2中。如果P4为1，则将答案颠倒。 其逻辑概括如下： • 如果P3==0且P4==0，则r[P2]：= r[P1]为真 • 如果P3==1且P4==1，则r[P2]：= r[P1]是 • 如果P3==0且P4==1，则r[P2]：= r[P1]不为真 • 如果P3==1且P4==0，则r[P2]：= r[P1]不相等

IsType	如果btree中的列的类型是由P5位掩码指定的类型之一，则跳转到P2。 P1通常是btree上的游标，对于该btree，行解码高速缓存至少通过列P3有效。换句话说，对于列P3或更大的列，应该存在先前的列。如果游标无效，则此操作码可能会给予虚假结果。如果btree行的列数少于P3，则使用P4作为数据类型。 如果P1为-1，则P3为寄存器编号，数据类型取自该寄存器中的值。 P5是数据类型的位掩码。SQLITE_INTEGER是最低有效位（0x 01）。SQLITE_FLOAT是0x 02位。SQLITE_TEXT为0x 04。SQLITE_BLOB为0x 08。SQLITE_NULL为0x 10。 P1：当P1>=0时，此操作码不能可靠地区分NULL和真实的。如果数据库包含一个NaN值，这个操作码会认为数据类型是真实的，而实际上它应该是NULL。当P1<0并且值已经存储在寄存器P3中时，该操作码确实可以可靠地区分NULL和真实的。只有当P1>=0时，问题才会出现。 当且仅当由P1和P3确定的值的数据类型对应于P5位掩码中的一个位时，跳转到地址P2。
JournalMode	将数据库P1的日记模式更改为P3。P3必须是PAGER_JOURNALMODE_XXX值之一。如果在各种回滚模式（删除，截断，持久化，关闭和内存）之间切换，这是一个简单的操作。不需要IO。 如果切换到WAL模式或退出WAL模式，则程序更加复杂。 将包含最终日志模式的字符串写入寄存器P2。
Jump	跳转到地址P1、P2或P3处的指令，具体取决于在最近的比较指令中P1向量分别小于、等于还是大于P2向量。 此操作码必须紧跟在Compare操作码之后。
Last	P1的Rowid或Column或Prev指令的下一次使用将引用数据库表或索引中的最后一个条目。如果表或索引为空且P2>0，则立即跳转到P2。如果P2为0或者表或索引不为空，则执行以下指令。 此操作码将光标配置为以相反的顺序从末尾向开头移动。换句话说，光标被配置为使用Prev，而不是Next。
Le	这就像Lt操作码一样工作，只是如果寄存器P3的内容小于或等于寄存器P1的内容，则进行跳转。有关更多信息，请参见Lt操作码。

LoadAnalysis	读取数据库P1的sqlite_stat1表，并将该表的内容加载到内部索引哈希表中。这将导致在准备所有后续查询时使用分析。
Lt	比较寄存器P1和P3中的值。如果reg（P3） 如果P5的SQLITE_JUMPIFNULL位被设置，并且reg（P1）或reg（P3）为NULL，则进行跳转。如果清除了SQLITE_JUMPIFNULL位，则在任一操作数为NULL时失效。 P5的SQLITE_AFF_MASK部分必须是关联字符-SQLITE_AFF_TEXT、SQLITE_AFF_INTEGER等。在进行比较之前，会尝试根据此亲和性强制两个输入。如果SQLITE_AFF_MASK为0x 00，则使用数值关联。注意，相似性转换被存储回输入寄存器P1和P3。因此，此操作码可能会导致寄存器P1和P3的持久更改。 一旦发生了任何转换，并且两个值都不是NULL，则比较这些值。如果两个值都是blob，则使用memcmp（）确定比较结果。如果两个值都是文本，则使用P4中指定的相应排序函数进行比较。如果没有指定P4，则使用memcmp（）比较文本字符串。如果两个值都是数值，则使用数值比较。如果这两个值的类型不同，那么数字被认为小于字符串，字符串被认为小于blob。 此操作码保存比较结果以供新的Jump操作码使用。
MakeRecord	将从P1开始的P2寄存器转换为记录格式，用作数据库表中的数据记录或索引中的键。列操作码可以稍后解码记录。 P4可以是P2个字符长的字符串。字符串的第N个字符指示应用于索引键的第N个字段的列亲和性。 从字符到亲缘关系的映射由sqliteInt.h中定义的SQLITE_AFF_ macros给出。 如果P4为NULL，则所有索引字段都具有亲和性BLOB。 P5的含义取决于是否启用了SQLITE_ENABLE_NULL_TRIM编译时选项： * 如果启用了SQLITE_ENABLE_NULL_TRIM，则P5是可以进行空值修剪的最右侧表的索引。 * 如果省略SQLITE_ENABLE_NULL_TRIM，则如果允许MakeRecord操作码接受serial_type 10的无更改记录，则P5具有值OPFLAG_NOCHNG_MAGIC。此值仅在assert() 内部使用，不会影响最终结果。

MaxPgcnt	尝试将数据库P1的最大页数设置为P3中的值。不要让最大页面计数低于当前页面计数，如果P3==0，则不要更改最大页面计数值。在寄存器P2中存储更改后的最大页计数。
MemMax	P1是此VM的根帧中的寄存器（如果此指令正在子程序中执行，则根帧与当前帧不同）。将寄存器P1的值设置为其当前值和寄存器P2中的值的最大值。 如果存储单元最初不是整数，则此指令抛出错误。
Move	移动寄存器P1中的P3值。P1+P3-1输入寄存器P2。P2+P3-1。寄存器P1.. P1+P3-1保持NULL。这是寄存器范围P1的错误。P1+P3-1和P2。P2+P3-1重叠。P3小于1是错误的。
Multiply	将寄存器P1中的值乘以寄存器P2中的值，并将结果存储在寄存器P3中。如果任何一个输入为NULL，则结果为NULL。
MustBeInt	强制寄存器P1中的值为整数。如果P1中的值不是整数，并且不能在不丢失数据的情况下转换为整数，则立即跳转到P2，或者如果P2==0，则引发SQLITE_MISMATCH异常。
Ne	这就像Eq操作码一样工作，只是如果寄存器P1和P3中的操作数不相等，则进行跳转。有关更多信息，请参见Eq操作码。
NewRowid	获取一个新的整数记录号（也称为“rowid”），用作表的键。记录号以前没有用作游标P1所指向的数据库表中的键。新记录号被写入寄存器P2。 如果P3>0，则P3是该VDBE的根帧中的寄存器，其保持最大的先前生成的记录号。不允许新记录编号小于此值。当此值达到最大值时，将生成SQLITE_FULL错误。P3寄存器用“生成的记录号”更新。此P3机制用于帮助实现AUTOINCREMENT特性。

Next	前进游标P1，使其指向表或索引中的下一个键/数据对。如果没有更多的键/值对，则执行以下指令。但如果光标前进成功，则立即跳转到P2。 下一个操作码仅在用于定位光标的SeekGT、SeekGE或Rewind操作码之后有效。Next不允许跟在SeekLT、SeekLE或Last之后。 P1游标必须用于真实的表，而不是伪表。P1必须在此操作码之前打开，否则程序将发生故障。 P3值是对btree实现的提示。如果P3==1，这意味着P1是一个SQL索引，如果该索引是唯一的，则可以省略此指令。P3通常为0。P3总是0或1。 如果P5为正并且进行跳转，则准备语句中的事件计数器编号P5-1递增。 参见：Prev
NoConflict	如果P4==0，则寄存器P3保存由MakeRecord构造的blob。如果P4>0，则寄存器P3是形成解包记录的P4个寄存器中的第一个。游标P1位于索引btree上。如果由P3和P4标识的记录包含任何NULL值，则立即跳转到P2。如果记录的所有项都不是NULL，则进行检查以确定P1索引btree中的任何行是否具有匹配的键前缀。如果没有匹配项，则立即跳转到P2。如果存在匹配项，则将P1光标保留在匹配行上。 此操作码类似于NotFound，不同之处在于，如果搜索关键字输入的任何部分为NULL，则始终采用分支。 此操作将使光标处于无法向任何方向前进的状态。换句话说，Next和Prev操作码在此操作后不起作用。 另请参阅：NotFound、Found、NotFound sts
Noop	什么都别做此指令通常用作跳转目的地。
Not	将寄存器P1中的值解释为布尔值。将布尔补码存储在寄存器P2中。如果寄存器P1中的值为NULL，则NULL被存储在P2中。

NotExists	P1是在SQL表btree上打开的游标的索引（带有整数键）。P3是整数rowid。如果P1不包含rowid为P3的记录，则立即跳转到P2。如果P2为0，则引发SQLITE_CORRUPT错误。如果P1确实包含rowid为P3的记录，则将光标保持指向该记录，并继续执行下一条指令。 SeekRowid操作码执行相同的操作，但也允许P3寄存器包含非整数值，在这种情况下，总是进行跳转。此操作码要求P3始终包含整数。 NotFound操作码对索引btree（具有任意多值键）执行相同的操作。 此操作码使游标处于无法向任何方向前进的状态。换句话说，Next和Prev操作码在此操作码之后将不起作用。 参见：Found、NotFound、NoConflict、SeekRowid
NotFound	如果P4==0，则寄存器P3保存由MakeRecord构造的blob。如果P4>0，则寄存器P3是形成解包记录的P4个寄存器中的第一个。 游标P1位于索引btree上。如果由P3和P4标识的记录不是P1中任何条目的前缀，则跳转到P2。如果P1确实包含其前缀与P3/P4记录匹配的条目，则控制福尔斯下降到下一条指令，并且P1左指向匹配条目。 此操作将使光标处于无法向任何方向前进的状态。换句话说，Next和Prev操作码在此操作后不起作用。 参见：Found、Nothists、NoConflict、IfNoHope
NotNull	如果寄存器P1中的值不为NULL，则跳转到P2。
Null	将NULL写入寄存器P2。如果P3大于P2，则也将NULL写入寄存器P3以及P2和P3之间的每个寄存器。如果P3小于P2（通常P3为零），则只有寄存器P2被设置为NULL。 如果P1值为非零，则还设置MEM_Cleared标志，以便即使在Ne或Eq上设置了SQLITE_NULLEQ，NULL值也不会比较相等。
NullRow	将光标P1移动到空行。当游标位于空行上时发生的任何列操作将始终写入NULL。 如果游标P1以前没有打开过，那么现在将其打开为一个特殊的伪游标，该伪游标总是为每一列返回NULL。

Offset	在寄存器r[P3]中存储数据库文件中的字节偏移量，该偏移量是游标P1当前指向的记录的有效负载的开始。 P2是sqlite_offset（）函数参数的列号。此操作码本身不使用P2，但代码生成器使用P2值。此操作码的P1、P2和P3操作数与列的操作数相同。 此操作码仅在SQLite使用-DSQLITE_ENABLE_OFFSET_SQL_FFSET选项编译时可用。
OffsetLimit	此操作码执行与LIMIT和OFFSET处理相关的常用计算。r[P1]保存极限计数器。r[P3]保存偏移计数器。操作码计算LIMIT和OFFSET的组合值，并将该值存储在r[P2]中。计算的r[P2]值是为了完成查询而需要访问的行的总数。 如果r[P3]为零或负数，则意味着没有OFFSET，r[P2]被设置为LIMIT r[P1]的值。 如果r[P1]为零或负数，则表示没有LIMIT，r[P2]被设置为-1。 否则，r[P2]被设置为r[P1]和r[P3]的和。
Once	在每次调用字节码程序时，第一次遇到此操作码时，执行下一条指令。在同一次调用中，在第二次和所有后续遇到时跳转到P2。 顶级程序通过比较P1操作数与程序开始时Init操作码上的P1操作数来确定第一次调用。如果P1值不同，则失败并使此操作码的P1等于Init的P1。如果P1值相同，则跳转。 对于子程序，VdbeFrame中有一个位掩码，用于确定是否应进行跳转。位掩码是必要的，因为自修改代码技巧不适用于递归触发器。
OpenAutoindex	此操作码的工作原理与OpenEphemeral相同。它有一个不同的名字来区分它的用途。使用此操作码创建的表将用于连接中自动创建的临时索引。
OpenDup	打开一个新的游标P1，它指向与游标P2相同的临时表。P2游标必须已由先前的OpenEphemeral操作码打开。只能复制临时游标。重复的临时游标用于实体化视图的自联接。

OpenEphemeral	打开一个新的游标P1到一个临时表。即使主数据库是只读的，游标也总是以读/写方式打开。临时表在游标关闭时自动删除。 如果游标P1已经在临时表上打开，则清除该表（擦除所有内容）。 P2是临时表中的列数。如果P4==0，则光标指向BTree表;如果P4不为0，则光标指向BTree索引。如果P4不为NULL，则它指向定义索引中键格式的KeyInfo结构。 P5参数可以是btree. h中定义的BTREE_* 标志的掩码。这些标志控制btree操作的各个方面。自动添加BTREE_OMIT_JOURNAL和BTREE_SINGLE标志。 如果P3为正数，则reg[P3]会稍微修改，以便它可以用作Insert的零长度数据。这是一种优化，避免了额外的Blob操作码来初始化该寄存器。
OpenPseudo	打开一个新的游标，它指向一个包含单行数据的伪表。这一行的内容是存储器寄存器P2的内容。换句话说，游标P1成为寄存器P2中包含的MEM_Blob内容的别名。 这个操作码创建的伪表用于保存排序器(sorter)的单行输出，以便可以使用列操作码将行分解为各个列。列操作码是唯一一个与伪表一起工作的游标操作码。 P3是伪表将存储的记录中的字段数。
OpenRead	在数据库文件中为其根页为P2的数据库表打开只读游标。数据库文件由P3确定。P3==0表示主数据库，P3==1表示用于临时表的数据库，P3>1表示使用了相应的附加数据库。给予新游标一个标识符P1。P1值不需要是连续的，但所有P1值都应该是小整数。P1为负是错误的。 允许的P5位： 0x 02 OPFLAG_SEEKEQ：此游标将仅用于相等查找（实现为SeekLE/ IdxLT的一对操作码SeekGE/ IdxGT） P4值可以是整数（P4_INT32）或指向KeyInfo结构的指针（P4_KEYINFO）。如果它是一个指向KeyInfo对象的指针，那么打开的表必须是一个索引b树，其中KeyInfo对象定义了该索引b树的内容和排序序列。否则，如果P4是一个整数值，那么打开的表必须是一个表b树，其列数不小于P4的值。 参见：OpenWrite、ReopenIdx

OpenWrite	在根页为P2的表或索引上打开名为P1的读/写游标（如果在P5中设置了OPFLAG_P2ISREG位，则根页保存在寄存器P2中-见下文）。 P4值可以是整数（P4_INT32）或指向KeyInfo结构的指针（P4_KEYINFO）。如果它是一个指向KeyInfo对象的指针，那么打开的表必须是一个索引b树，其中KeyInfo对象定义了该索引b树的内容和排序序列。否则，如果P4是一个整数值，那么打开的表必须是一个表b树，其列数不小于P4的值。 允许的P5位： • 0x 02 OPFLAG_SEEKEQ：此游标将仅用于相等查找（实现为SeekLE/ IdxLT的一对操作码SeekGE/ IdxGT） • 0x 08 OPFLAG_FORMATION：这个游标只用于在索引树中查找并随后删除条目。这是对存储引擎的提示，允许存储引擎忽略。该提示不被官方的SQLite b* 树存储引擎使用，但被COMDB 2使用。 • 0x 10 OPFLAG_P2ISREG：使用寄存器P2的内容作为根页，而不是P2本身的值。 此指令的工作方式类似于OpenRead，只是它以读/写模式打开游标。 参见：OpenRead、ReopenIdx
Or	对寄存器P1和P2中的值进行逻辑或运算，并将答案存储在寄存器P3中。 如果P1或P2为非零（真），则结果为1（真），即使另一个输入为NULL。NULL和false或两个NULL给予NULL输出。
Pagecount	将数据库P1中的当前页数写入存储单元P2。
Param	此操作码仅存在于通过Program指令调用的子程序中。将当前存储在调用（父）帧的存储单元中的值复制到当前帧地址空间中的单元P2。这是触发程序用来访问新的。*老 *价值观 通过将P1参数的值与调用Program指令的P1参数的值相加来确定父帧中单元的地址。
ParseSchema	从数据库P1的模式表中读取并解析与WHERE子句P4匹配的所有条目。如果P4是一个NULL指针，那么P1的整个模式将被重新解析。这个操作码调用解析器来创建一个新的虚拟机，然后运行这个新的虚拟机。因此，它是一个可重入的操作码。

Permutation	设置下一条指令中Compare运算符使用的排列。置换存储在P4操作数中。 置换仅对下一个操作码有效，该操作码必须是在P5中设置了OPFLAG_PERMUTE位的比较。 P4整数数组中的第一个整数是数组的长度，不会成为排列的一部分。
Prev	备份游标P1，使其指向其表或索引中的前一个键/数据对。如果没有先前的键/值对，则执行以下指令。但如果游标备份成功，则立即跳转到P2。 Prev操作码仅在用于定位光标的SeekLT、SeekLE或Last操作码之后才有效。Prev不允许跟随SeekGT、SeekGE或Rewind。 P1游标必须用于真实的表，而不是伪表。如果P1未打开，则行为未定义。 P3值是对btree实现的提示。如果P3==1，这意味着P1是一个SQL索引，如果该索引是唯一的，则可以省略此指令。P3通常为0。P3总是0或1。 如果P5为正并且进行跳转，则准备语句中的事件计数器编号P5-1递增。
Program	执行作为P4传递的触发程序（类型P4_SUBSIDE）。 P1包含存储单元的地址，该存储单元包含用作子程序的参数的值阵列中的第一个存储单元。P2包含子程序使用RAISE（）函数抛出IGNORE异常时跳转到的地址。寄存器P3包含此（父）VM中的存储器单元的地址，其用于在运行时分配子vdbe所需的存储器。 P4是指向包含触发器程序的VM的指针。 如果P5非零，则递归程序调用被启用。

PureFunc	使用从寄存器P2和后继寄存器中获取的参数创建用户函数（P4是指向sqlite3_context对象的指针，该对象包含指向要运行的函数的指针）。参数的数量在P4指向的sqlite3_context对象中。该函数的结果存储在寄存器P3中。寄存器P3不能是功能输入之一。 P1是一个32位位掩码，指示函数的每个参数是否在编译时被确定为常量。如果第一个参数是常量，则P1的位0被设置。这用于确定与使用sqlite3_set_auxdata（）API的用户函数参数相关联的Meta数据是否可以安全地保留，直到下一次调用此操作码。 这个操作码的工作方式与Function完全一样。唯一的区别是它的名字。此操作码用于函数必须完全不确定的地方。一些内置的日期/时间函数可以是确定性的，也可以是非确定性的，这取决于它们的参数。当这些函数以非确定性的方式使用时，它们将检查是否使用PureFunc而不是Function调用它们，如果是，它们将抛出错误。 参见：AggStep、AggFinal、Function
ReadCookie	从数据库P1中读取cookie编号P3并将其写入寄存器P2。P3==1是模式版本。P3==2是数据库格式。P3==3是推荐的寻呼机缓存大小，依此类推。P1==0是主数据库文件，P1==1是用于存储临时表的数据库文件。 在执行此指令之前，数据库上必须有一个读锁（必须启动一个事务或必须有一个打开的游标）。
Real	P4是指向64位浮点值的指针。将该值写入寄存器P2。
RealAffinity	如果寄存器P1保存的是整数，则将其转换为真实的值。 当从具有真实的亲和性的列中提取信息时，将使用此操作码。为了节省空间，这些列值仍然可以存储为整数，但是在提取之后，我们希望它们只有一个真实的值。

ReleaseReg	从服务中释放寄存器。在此操作码完成后，寄存器中的任何内容都不可靠。 释放的寄存器将是从P1开始的P2寄存器，除非P3的位ii被设置，则不释放寄存器P1+ii。换句话说，P3是要保留的寄存器的掩码。 释放寄存器将清除Mem.pScopyFrom指针。这意味着，如果释放寄存器的内容是使用SCopy设置的，则对SCopy的源寄存器值的更改将不再在sqlite3VdbeMemAboutToChange（）中生成断言错误。 如果设置了P5，则所有释放的寄存器都将其类型设置为MEM_Undefined，以便任何后续尝试读取释放的寄存器（在重新初始化之前）将生成断言故障。 P5应该在每次调用这个操作码时设置。然而，代码生成器中的某些地方将在使用寄存器之前释放寄存器，这是在寄存器在使用之前不会被重新分配用于其他目的的（有效）假设下，因此释放是安全的。 此操作码仅在测试和调试版本中可用。不为发布版本生成。此操作码的目的是帮助验证生成的字节码。此操作码实际上并不有助于计算答案。
Remainder	计算整数寄存器P2除以寄存器P1后的余数，并将结果存储在寄存器P3中。如果寄存器P1中的值为零，则结果为NULL。如果任一操作数为NULL，则结果为NULL。
ReopenIdx	ReopenIdx操作码的工作原理与OpenRead类似，不同之处在于它首先检查P1上的游标是否已经在同一个b树上打开，如果是，则该操作码将变为空操作。换句话说，如果游标已经打开，则不要重新打开它。 ReopenIdx操作码只能用于P5==0或P5==OPFLAG_SEEKEQ，并且P4是P4_KEYINFO对象。此外，P3值必须与相同游标编号的每个其他ReopenIdx或OpenRead相同。 允许的P5位： 0x 02 OPFLAG_SEEKEQ：此游标将仅用于相等查找（实现为SeekLE/ IdxLT的一对操作码SeekGE/ IdxGT） 标签：OpenRead，OpenWrite
ResetCount	更改计数器的值被复制到数据库句柄更改计数器（由后续调用sqlite3_changes（）返回）。然后，VM内部更改计数器重置为0。这是由触发程序使用的。

ResetSorter	从临时表或排序器中删除游标P1上打开的所有内容。 此操作码仅适用于用于排序并使用OpenEphemeral或SorterOpen打开的游标。
ResultRow	寄存器P1至P1+P2-1包含单行结果。此操作码会导致sqlite3_step（）调用以SQLITE_ROW返回码终止，并设置sqlite3_stmt结构以提供对r（P1）的访问。r（P1+P2-1）值作为结果行。
Return	跳转到寄存器P1中存储的地址。如果P1是一个返回地址寄存器，那么这就完成了从子程序的返回。 如果P3为1，则仅当寄存器P1保存整数值时才进行跳转，否则执行福尔斯到下一个操作码，并且Return变为空操作。如果P3为0，则寄存器P1必须保存整数，否则引发assert（）。当这个操作码与BeginSubrtn结合使用时，P3应该设置为1，否则设置为0。 寄存器P1中的值不受此操作码的影响。 字节码引擎不使用P2。但是，如果P2为正并且小于当前地址，则CLI中的“EXPLAIN”输出格式化程序将从P2操作码中删除所有操作码，直到不包括当前Return。P2应该是子例程中的第一个操作码，该操作码将从该子例程返回。因此，P2值是字节码缩进提示。参见wherecode. c和shell. c中的tag-20220407 a。
Rewind	下一次对P1使用Rowid或Column或Next指令将引用数据库表或索引中的第一个条目。如果表或索引为空，则立即跳转到P2。如果表或索引不为空，则执行以下指令。 如果P2为零，则断言P1表永远不为空，因此永远不会进行跳转。 此操作码使光标配置为从开始到结束按向前顺序移动。换句话说，光标被配置为使用Next，而不是Prev。
RowCell	P1和P2都是打开的游标。两者都必须在相同类型的表上打开-intkey或index。此操作码用作将当前行从P2复制到P1的一部分。 如果在intkey表上打开游标，则寄存器P3包含与P1中的新记录一起使用的rowid。如果在索引表上打开它们，则不使用P3。 此操作码后面必须跟有一个Insert或InsertIdx操作码，并设置OPFLAG_PREFORMAT标志，以完成插入操作。

RowData	将游标P1当前所指向的行的完整行内容写入寄存器P2。没有对数据进行解释。它只是复制到P2寄存器中，就像在数据库文件中找到的那样。 如果游标P1是一个索引，则内容是行的键。如果游标P2是一个表格，那么提取的内容就是数据。 如果P1游标必须指向真实的表（而不是伪表）的有效行（而不是NULL行）。 如果P3！= 0，则允许此操作码将临时指针设置为数据库页面。这意味着，一旦游标移动，输出寄存器的内容将无效-包括由“保存”当前游标位置的其他游标引起的移动，以便它们可以写入同一表。如果P3==0，则将数据的副本复制到内存中。P3！= 0更快，但P3==0更安全。 如果P3！= 0，则P2寄存器的内容不适合在OP_Result中使用，并且任何OP_Result都将使P2寄存器内容无效。P2寄存器的内容被操作码（如Function）或任何指向同一表的游标的使用所无效。
Rowid	在寄存器P2中存储一个整数，该整数是P1当前所指向的表项的键。 P1可以是普通表，也可以是虚拟表。曾经有一个单独的OP_VRrowid操作码用于虚拟表，但现在这一个操作码适用于两种表类型。
RowSetAdd	将寄存器P2保存的整数值插入寄存器P1保存的RowSet对象。 如果P2不是整数，则断言失败。
RowSetRead	从P1中的RowSet对象中提取最小值，并将该值放入寄存器P3。或者，如果RowSet对象P1最初为空，则保持P3不变并跳转到指令P2。

RowSetTest	假设寄存器P3保存64位整数值。如果寄存器P1包含RowSet对象，并且该RowSet对象包含P3中保存的值，则跳转到寄存器P2。否则，将P3中的整数插入RowSet并继续执行下一个操作码。 RowSet对象针对在不同阶段中插入整数集合的情况进行了优化，每个集合不包含重复项。每个集合由唯一的P4值标识。第一个集合必须具有P4==0，最后一个集合必须具有P4==-1，并且对于所有其他集合必须具有P4>0。 这允许优化：（a）当P4==0时，不需要测试P3的RowSet对象，因为它保证不包含它，（b）当P4==-1时，不需要插入值，因为它永远不会被测试，以及（c）当插入作为集合X的一部分的值时，不需要搜索以查看相同的值是否先前作为集合X的一部分被插入（仅当它先前作为某个其它集合的一部分被插入时）。
Savepoint	打开、释放或回滚由参数P4命名的保存点，具体取决于P1的值。 打开新保存点集P1==0（SAVEPOINT_开始）。释放（提交）现有保存点集P1==1（SAVEPOINT_RELEASE）。回滚现有保存点集P1==2（SAVEPOINT_ROLLBACK）。
SCopy	将寄存器P1浅拷贝到寄存器P2中。 此指令生成值的浅拷贝。如果值是字符串或blob，则副本只是指向原始值的指针，因此如果原始值发生变化，副本也会发生变化。更糟糕的是，如果原始文件被释放，副本将变得无效。因此，程序必须保证在副本的生命周期内，原始文件不会被更改。使用“复制”创建完整的副本。
SeekEnd	将光标P1定位在btree的末尾，以便将新条目追加到btree上。 假设游标仅用于追加，因此如果游标有效，则游标必须已经指向btree的末尾，因此不会对游标进行任何更改。

SeekGE	如果游标P1引用SQL表（使用整数键的B树），则使用寄存器P3中的值作为键。如果游标P1引用SQL索引，则P3是用作解压缩索引键的P4寄存器数组中的第一个。 重新定位光标P1，使其指向大于或等于键值的最小条目。如果没有大于或等于键的记录，并且P2不为零，则跳转到P2。 如果游标P1是使用OPFLAG_SEEKEQ标志打开的，那么这个操作码要么落在与键完全匹配的记录上，要么跳到P2。当游标为OPFLAG_SEEKEQ时，此操作码后面必须跟一个具有相同参数的IdxLE操作码。如果此操作码成功，则IdxGT操作码将被跳过，但IdxGT操作码将用于后续循环迭代。OPFLAG_SEEKEQ标志是对btree层的一个提示，表明这是一个相等搜索。 此操作码使光标配置为从开始到结束按向前顺序移动。换句话说，光标被配置为使用Next，而不是Prev。 另请参阅：Found、NotFound、SeekLt、SeekGt、SeekLe
SeekGT	如果游标P1引用SQL表（使用整数键的B树），则使用寄存器P3中的值作为键。如果游标P1引用SQL索引，则P3是用作解压缩索引键的P4寄存器数组中的第一个。 重新定位光标P1，使其指向大于键值的最小条目。如果没有大于键的记录，并且P2不为零，则跳到P2。 此操作码使光标配置为从开始到结束按向前顺序移动。换句话说，光标被配置为使用Next，而不是Prev。 另请参阅：Found、NotFound、SeekLt、SeekGe、SeekLe
SeekHit	如有必要，请增大或减小游标P1的seekHit值，使其不小于P2且不大于P3。 seekHit整数表示索引中已知至少有一个匹配项的最大项。如果seekHit值小于索引查找中相等项的总数，则可以运行IfNoHope操作码来查看是否可以提前放弃IN循环，从而节省工作。这是“提前退出”优化的一部分。 P1必须是有效的b树游标。

SeekLE	如果游标P1引用SQL表（使用整数键的B树），则使用寄存器P3中的值作为键。如果游标P1引用SQL索引，则P3是用作解压缩索引键的P4寄存器数组中的第一个。 重新定位光标P1，使其指向小于或等于键值的最大条目。如果没有小于或等于键的记录，并且P2不为零，则跳转到P2。 此操作码将光标配置为以相反的顺序从末尾向开头移动。换句话说，光标被配置为使用Prev，而不是Next。 如果游标P1是使用OPFLAG_SEEKEQ标志打开的，那么这个操作码要么落在与键完全匹配的记录上，要么跳到P2。当游标为OPFLAG_SEEKEQ时，此操作码后面必须跟一个具有相同参数的IdxLE操作码。如果此操作码成功，则IdxGE操作码将被跳过，但IdxGE操作码将用于后续循环迭代。OPFLAG_SEEKEQ标志是对btree层的一个提示，表明这是一个相等搜索。 另请参阅：Found、NotFound、SeekGt、SeekGe、SeekLt
SeekLT	如果游标P1引用SQL表（使用整数键的B树），则使用寄存器P3中的值作为键。如果游标P1引用SQL索引，则P3是用作解压缩索引键的P4寄存器数组中的第一个。 重新定位光标P1，使其指向小于键值的最大条目。如果没有小于键的记录，并且P2不为零，则跳到P2。 此操作码将光标配置为以相反的顺序从末尾向开头移动。换句话说，光标被配置为使用Prev，而不是Next。 另请参阅：Found、NotFound、SeekGt、SeekGe、SeekLe
SeekRowid	P1是在SQL表btree上打开的游标的索引（带有整数键）。如果寄存器P3不包含整数或P1不包含rowid为P3的记录，则立即跳转到P2。如果P2为0，则引发SQLITE_CORRUPT错误。如果P1确实包含rowid为P3的记录，则将光标保持指向该记录，并继续执行下一条指令。 Notrests操作码执行相同的操作，但使用Notrests时，必须保证P3寄存器包含整数值。使用此操作码，寄存器P3可能不包含整数。 NotFound操作码对索引btree（具有任意多值键）执行相同的操作。 此操作码使游标处于无法向任何方向前进的状态。换句话说，Next和Prev操作码在此操作码之后将不起作用。 参见：Found、NotFound、NoConflict、SeekRowid

SeekScan

此操作码是SeekGE的前缀操作码。换句话说，这个操作码必须紧跟在SeekGE之后。此约束由assert（）语句检查。
此操作码使用后续SeekGE的P1到P4操作数。在下文中，后续SeekGE操作码的操作数表示为SeekOP.P1至SeekOP.P4。只有P1，P2和P5操作数也被使用，并被称为This.P1，This.P2和This.P5。

此操作码有助于优化多列索引上的IN运算符，其中IN运算符位于索引的后面项上，方法是避免在btree上进行不必要的查找，而是将步骤替换到b-tree的下一行。如果这个操作码被省略或者是空操作，则获得正确答案。

SeekGE.P3和SeekGE.P4操作数标识一个未打包的键，它是我们希望光标SeekGE.P1指向的所需条目。将SeekGE.P3/P4行称为“目标”。

如果SeekGE.P1游标当前未指向有效行，则此操作码为空操作，控制将传递到SeekGE。

如果SeekGE.P1游标指向有效行，则该行可能是目标行，也可能靠近目标行并略早于目标行，或者在目标行之后。如果游标当前位于目标行之前，则此操作码尝试通过在游标上调用sqlite3 BtreeStep（）1到This.P1次，将游标定位在目标行之上或之后。

This.P5参数是一个标志，它指示当光标最终指向超过目标行的有效行时应执行的操作。如果This.P5为false（0），则跳转到SeekGE.P2。如果This.P5为真（非零），则跳转到This.P2。P5==0的情况发生在IN约束右侧没有不等式约束时。跳转到SeekGE.P2结束循环。P5！当IN运算符右侧存在不等式约束时，将发生0情况。在这种情况下，This.P2将直接指向或指向检查循环终止的IdxGT或IdxGE操作码之前的设置代码。

此操作码的可能结果：

1. 如果游标最初没有指向任何有效的行，那么就进入后续的SeekGE操作码。
2. 如果光标指向目标行之前的一行，即使在对sqlite3 BtreeNext（）进行了多达This.P1的调用之后，也会落入SeekGE。
3. 如果游标指向目标行，或者是因为它位于要开始的目标行，或者是因为一个或多个sqlite3 BtreeNext（）调用将游标移动到目标行，则跳转到This.P2..，
4. 如果光标在目标行之前开始，并且调用to sqlite3 BtreeNext（）将光标移到了索引的末尾（表示目标行肯定不存在于btree中），则跳转到SeekGE.P2，结束循环。

	5. 如果游标在一个有效的行上结束，而该行已经超过了目标行（表示目标行在btree中不存在），那么如果This.P5==0，则跳转到SeekOP.P2;如果This.P5>0，则跳转到This.P2。
Sequence	查找游标P1的下一个可用序列号。将序列号写入寄存器P2。在此指令之后，游标上的序列号递增。
SequenceTest	P1是一个排序器游标。如果序列计数器当前为零，则跳转到P2。无论是否进行跳转，递增序列值。
SetCookie	将整数值P3写入数据库P1的cookie编号P2。P2==1是模式版本。P2==2是数据库格式。P2==3是推荐的寻呼机缓存大小，依此类推。P1==0是主数据库文件，P1==1是用于存储临时表的数据库文件。 必须在执行此操作码之前启动事务。 如果P2是SCHEMA_VERSION cookie（cookie编号1），则内部模式版本设置为P3-P5。“PRAGMA schema_version=N”语句将P5设置为1，因此内部模式版本将不同于数据库模式版本，从而导致模式重置。
ShiftLeft	将寄存器P2中的整数值向左移位寄存器P1中整数指定的位数。将结果存储在寄存器P3中。如果任何一个输入为NULL，则结果为NULL。
ShiftRight	将寄存器P2中的整数值向右移位寄存器P1中整数指定的位数。将结果存储在寄存器P3中。如果任何一个输入为NULL，则结果为NULL。
SoftNull	将寄存器P1设置为由MakeRecord指令看到的值NULL，但不释放与寄存器相关联的任何字符串或blob内存，以便如果该值是先前使用SCopy复制的字符串或blob，则副本将继续有效。
Sort	这个操作码与Rewind做的事情完全相同，除了它增加了一个用于测试的未记录的全局变量。 排序是通过将记录写入排序索引，然后倒回该索引并从头到尾播放来完成的。我们使用Sort操作码而不是Rewind来执行倒回，这样全局变量将递增，回归测试可以确定优化器是否正确地优化了排序。

SorterCompare	P1是一个排序器游标。该指令将寄存器P3中的记录blob的前缀与分类器光标当前指向的条目的前缀进行比较。只比较r[P3]的前P4字段和排序器记录。 如果P3或排序器(sorter)在它们的有效字段之一中包含NULL（不包括在末尾被忽略的P4字段），则假设比较相等。 如果两个记录比较相等，则执行下一条指令。如果它们不同，则跳转到P2。
SorterData	将排序器(sorter)游标P1的当前排序器(sorter)数据写入寄存器P2。然后清除游标P3上的列标题缓存。 此操作码通常用于将记录移出排序器(sorter)并移入寄存器，该寄存器是使用OpenPseudo创建的伪表游标的源。该伪表游标是由参数P3标识的游标。作为此操作码的一部分，清除P3列缓存使我们不必发出单独的NullRow指令来清除该缓存。
SorterInsert	寄存器P2保存使用MakeRecord指令生成的SQL索引键。这个操作码将该键写入排序器P1。条目的数据为nil。
SorterNext	此操作码的工作方式与Next类似，不同之处在于P1必须是已为其调用SorterSort操作码的排序器(sorter)对象。此操作码将光标推进到下一个排序记录，或者如果没有更多的排序记录，则跳转到P2。
SorterOpen	此操作码的工作方式类似于OpenEphemeral，只是它打开了一个临时索引，该索引专门用于使用外部合并排序算法对大型表进行排序。 如果参数P3为非零，则表明排序器(sorter)可以假设考虑每个关键字的前P3个字段的稳定排序足以产生所需的结果。
SorterSort	在所有记录都插入到由P1标识的Sorter对象中之后，调用此操作码来实际执行排序。如果没有要排序的记录，则跳转到P2。 此操作码是用于Sorter对象的Sort and Rewind的别名。
SqlExec	运行P4字符串中指定的SQL语句。如果P1为true，则在这些语句运行时禁用Auth和Trace回调。

String	长度为P1（字节）的字符串值P4存储在寄存器P2中。 如果P3不为零并且寄存器P3的内容等于P5，则寄存器P2的数据类型被转换为BLOB。内容是相同的字节序列，它只是被解释为一个BLOB而不是一个字符串，就像它被CAST一样。换句话说： 如果（P3！0 and reg[P3]==P5）reg[P2]：= CAST（reg[P2] as BLOB）
String8	P4指向以nul结尾的UTF-8字符串。这个操作码在第一次执行之前被转换成一个字符串操作码。在此转换过程中，计算字符串P4的长度并将其存储为P1参数。
Subtract	从寄存器P2中的值减去寄存器P1中的值，并将结果存储在寄存器P3中。如果任何一个输入为NULL，则结果为NULL。
TableLock	获取特定表上的锁。此指令仅在启用共享缓存功能时使用。 P1是获取锁的数据库在sqlite3.aDb[]中的索引。如果P3==0，则获得读锁定;如果P3==1，则获得写锁定。 P2包含要锁定的表的根页。 P4包含一个指向被锁定的表名的指针。这仅用于在无法获得锁时生成错误消息。
Trace	如果启用了语句跟踪，则在语句跟踪输出上写入P4。 操作数P1必须为0x7fffffff，P2必须为正数。

Transaction	如果事务尚未处于活动状态，则开始数据库P1上的事务。如果P2为非零，则开始写事务，或者如果读事务已经是活动的，则将其升级为写事务。如果P2为零，则开始读事务。如果P2等于或大于2，则启动独占事务。 P1是启动事务的数据库文件的索引。索引0是主数据库文件，索引1是用于临时表的文件。索引为2或更大用于附加的数据库。 如果开始写事务并且Vdbe.usesStmtJournal标志为真（如果Vdbe可以修改多个行并且可以抛出ABORT异常，则设置该标志），则还可以打开语句事务。更具体地说，如果数据库连接当前未处于自动提交模式，或者存在其他活动语句，则打开语句事务。语句事务允许在发生错误后回滚此VDBE所做的更改，而不必回滚整个事务。如果没有遇到错误，语句事务将在VDBE停止时自动提交。 如果P5！= 0，则此操作码还将对照P3检查模式cookie，并对照P4检查模式生成计数器。每当数据库架构发生更改时，Cookie都会更改其值。此操作用于检测cookie何时发生更改以及当前进程何时需要重新读取架构。如果P3中的模式cookie与数据库头中的模式cookie不同，或者P4中的模式生成计数器与当前生成计数器不同，则会引发SQLITE_SCHEMA错误并停止执行。然后，sqlite3_step（）包装器函数可能重新准备语句并从头开始重新编译。
TypeCheck	从P1开始，对P2寄存器范围应用亲和性。从P4中的Table对象中获取亲和度。如果任何值不能强制转换为正确的类型，则引发错误。此操作码类似于Affinity，只是此操作码强制寄存器类型为Table列类型。这是用来实现“严格亲和性”的。 生成始终作为.仅当P3为零时才检查STATIC列。当P3为非零时，不对静态生成的列进行类型检查。虚拟列是在查询时计算的，因此从不检查它们。 前提条件： • P2应该是P4表中的非虚拟列数。 • 表P4应该是一个STRICT表。 如果任何前提条件为假，则发生断言错误。
Vacuum	清空整个数据库P1。对于“main”，P1为0;对于附加数据库，P1为2或更大。“临时”数据库可能无法清空。 如果P2不为零，则它是一个寄存器，保存一个字符串，该字符串是真空结果应写入的文件。当P2为零时，真空覆盖原始数据库。

Variable	将绑定参数P1的值转移到寄存器P2中 如果参数已命名，则其名称将出现在P4中。P4值由sqlite3_bind_parameter_name（）使用。
VBegin	P4可以是指向sqlite3_vtab结构的指针。如果是，则调用该表的xBegin方法。 此外，无论是否设置了P4，都要检查是否在对虚拟表xSync（）方法的回调中调用了它。如果是，则错误代码将设置为SQLITE_LOCKED。
VCheck	P4是指向Table对象的指针，该对象是模式P1中支持xIntegrity（）方法的虚拟表。此操作码使用P3作为整数参数，为该虚拟表运行xIntegrity（）方法。如果返回了错误报告，则表名将被添加到错误消息之前，并且该消息将存储在P2中。如果没有发现错误，则寄存器P2被设置为NULL。
VColumn	在寄存器P3中存储游标P1的虚拟表的当前行的第P2列的值。 如果在UPDATE操作期间使用VColumn操作码获取不变列的值，则P5值为OPFLAG_NOCHNG。这将导致sqlite3_vtab_nocchange（）函数在虚表实现的xColumn方法中返回true。P5列还可能包含其他位（OPFLAG_LENGTHARG或OPFLAG_TYPEOFARG），但VColumn不使用这些位。
VCreate	P2是保持数据库P1中的虚拟表的名称的寄存器。调用该表的xCreate方法。
VDestroy	P4是数据库P1中虚拟表的名称。调用该表的xDestroy方法。
VFilter	P1是使用VOpen打开的游标。P2是一个在过滤结果集为空时跳转到的地址。 P4为NULL或由模块的xBestIndex方法生成的字符串。P4字符串的解释留给模块实现。 这个操作码在P1指定的虚拟表上调用xFilter方法。xFilter的整数查询计划参数存储在寄存器P3中。寄存器P3+1存储要传递给xFilter方法的argc参数。寄存器P3+2.. P3+1+argc是作为argv传递给xFilter的附加参数。寄存器P3+2在传递到xFilter时变为argv[0]。 如果过滤后的结果集为空，则跳转到P2。

VInitIn	用缓存寄存器P3和输出寄存器P3+1将寄存器P2设置为指向游标P1的ValueList对象的指针。此ValueList对象可用作sqlite3_vtab_in_first（）和sqlite3_vtab_in_next（）的第一个参数，以提取P1游标中存储的所有值。寄存器P3用于保存sqlite3_vtab_in_first（）和sqlite3_vtab_in_next（）返回的值。
VNext	将虚拟表P1前进到其结果集中的下一行，并跳转到指令P2。或者，如果虚拟表已到达其结果集的末尾，则继续执行下一条指令。
VOpen	P4是一个指向虚拟表对象的指针，一个sqlite3_vtab结构。P1是游标编号。这个操作码打开指向虚拟表的游标，并将该游标存储在P1中。
VRename	P4是一个指向虚拟表对象的指针，一个sqlite3_vtab结构。这个操作码调用相应的xtagem方法。寄存器P1中的值作为zName参数传递给xQuery方法。
VUpdate	P4是一个指向虚拟表对象的指针，一个sqlite3_vtab结构。此操作码调用相应的xUpdate方法。P2值是从P3开始传递到xUpdate调用的连续存储单元。寄存器（P3+P2-1）中的值对应于传递给xUpdate的argv数组的第p2个元素。 xUpdate方法将执行一个重命名或重命名，或者两者都执行。argv[0]元素（其对应于存储器单元P3）是要删除的行的rowid。如果argv[0]为NULL，则不会发生删除。argv[1]元素是新行的rowid。这个值可以为NULL，让虚拟表为自己选择新的rowid。数组中的后续元素是新行中列的值。 如果P2==1，则不执行插入。argv[0]是要删除的行的rowid。 P1是一个布尔标志。如果设置为true并且xUpdate调用成功，则sqlite3_last_insert_rowid（）返回的值将设置为刚插入的行的rowid值。 P5是在插入或更新的约束失败时应用的错误操作（OE_Replace、OE_Fail、OE_Ignore等）。
Yield	将程序计数器与寄存器P1中的值交换。这具有屈服于协程的效果。 如果此指令启动的协程以Yield或Return结束，则继续执行下一条指令。但如果此指令启动的协程以EndCoroutine结束，则跳转到P2，而不是继续下一条指令。 主条目：InitCoroutine

ZeroOrNull	如果寄存器P1和P3都为NOT NULL，则在寄存器P2中存储零。如果寄存器P1或P3为NULL，则将NULL放入寄存器P2。