1.检索数据

select语句

检索单个列

```
1  select keyword from tablename;
```

检索多个列

使用逗号间隔列名，最后一个列名后无需加逗号

```
1  select keyword1,keyword2,... from tablename;
```

检索所有列

使用通配符，检索所有的列会影响性能和降低检索速度

```
1  select * from tablename;
```

检索不同的值（去掉重复的值）

使用distinct关键字，放在列名前

```
1  select distinct keyword from tablename;
```

Distinct关键字作用于所有列，不仅仅是跟在其后的一列，且该关键字必须放在列名的前面。

限制输出结果

使用top关键字，限制最多返回多少行

```
1  select top 5 keyword from tablename;
```

返回最多5行

若使用oracle则为

```
1  select keyword from tablename where rownum<=5;
```

若使用mysql、Maria DB、SQLite，需要使用limit子句

```
1  select keyword from tablename limit 5;
```

可添加offset关键字来制定从哪开始以及检索的行数

```
1  select keyword from tablename limit 5 offset 5;
```

该语句即从第5行起的5行数据

使用注释

注释使用 -- 或 #，多行注释同c中的/*... */

2.排序检索数据

使用order by子句对输出进行排序，order by子句的位置为select 语句中最后一条子句，否则会出错，且order by排序顺序为字母顺序

```
1  select keyword from tablename order by keyword;
```

按多个列排序

下面的代码检索3个列，并按其中两个列对结果进行排序，列名之间用逗号分开即可。

```
1  select keyword1,keyword2,keyword3 from tablename order by
   keyword1,keyword2;
```

仅在多个行具有相同的keyword1时才对keyword2进行排序。

按列位置排序

order by同时还支持按相对列位置进行排序

```
1  select keyword1,keyword2,keyword3 from tablename order by 2,3;
```

Order by 2表示按select清单中的第二列keyword2进行排序，order by 2,3表示先按keyword2排序，再按keyword3排序

指定排序方向

数据排序不仅限于升序排序（从A到Z），还可以使用DESC关键字进行降序排序

```
1  select keyword1,keyword2,keyword3 from tablename order by keyword1 desc;
```

且desc关键字只应用到直接位于其前面的的列名，也即对多个列数据进行降序排列时，需要在每个列之后加上desc

3.过滤数据

使用where 子句

在select语句中，数据根据where子句中指定的搜索条件进行过滤。where子句在表名之后给出。

```
1  select keyword1, keyword2 from tablename where keyword1 = ?;
```

只显示keyword1=? 的行

在同时使用order by 和where 子句时，应该让order by 位于where之后否则将会产生错误，也即order by应是最后一个子句，在上文笔记已经提过。

where子句的操作符同大部分编程语言的操作符。

4.高级数据过滤

组合where子句

为了进行更强的过滤控制，SQL允许给出多个where子句，这些子句有两种使用方式，即以AND子句或OR子句的方式使用。

AND操作符

```
1  select keyword1, keyword2,keyword3 from tablename where keyword1=' ?'
   AND keyword2 <= ?;
```

增加过滤条件就需要增加AND关键字。

OR操作符

OR操作符与AND操作符正好相反，它指示DBMS检索匹配任一条件的行，即只要满足其中任意一个条件就会被检索出来。

与大多数高级语言相同，AND操作符和OR操作符也有运算顺序，故同时使用时应加上圆括号以明确分组。

例如

```
1  select keyword1,keyword2 from tablename where (keyword1 = ? OR keyword2
   = ?) AND keyword>=?;
```

IN操作符

IN操作符用来指定条件范围，范围中的每个条件都可以进行匹配。IN取一组由逗号分隔、括在圆括号中的合法值。

```
1  select prod_name,prod_price from Products WHERE vend_id IN
   ('DLL01','BRS01') ORDER BY prod_name;
```

此语句检索由DLL01和BRS01的产品。IN操作符后跟由逗号分隔的合法值，这些值 **必须在圆括号**中

NOT操作符

WHERE子句中的NOT操作符有且只有一个功能，那就是**否定其后所跟的任何条件**。

5.用通配符进行过滤

LIKE操作符

通配符（wildcard）即用来匹配值的一部分的特殊字符

在搜索子句中使用通配符，必须使用LIKE操作符。

百分号%通配符

在搜索串中，%表示任何字符出现任意次数。例如为了找出所有以Fish起头的产品，可写以下语句

```
1  SELECT prod_id , prod_name FROM Products WHERE prod_name LIKE 'Fish%';
```

%通配符根据DBMS的不同及其配置，搜索是可以区分大小写的。

通配符可在搜索模式中的任意位置使用，并且可以使用多个通配符。

Note：%通配符看起来可以匹配任何东西，但有个例外，就是NULL。子句 WHERE prod_name LIKE '%'不会匹配产品名称为NULL的行。

下划线(_)通配符

下划线的用途与%一样，但它只匹配单个字符，而不是多个字符。

```
1  SELECT prod_id,prod_name FROM Product WHERE pord_name LIKE'__ inch teddy
   bear';
```

此处使用两个下划线通配符，所以不会显示单个数字的行

方括号（[ ]）通配符

方括号用来指定一个字符集，它必须匹配指定位置的一个字符。

Note：并不总是支持集合，并不是所有DBMS都支持用来创建集合的[]。微软的SQL Server支持集合，但是MySQL，Oracle，DB2，SQLite都不支持。

使用通配符的技巧

- 不要过度使用通配符。如果其他操作符能达到相同目的，应该使用其他操作符
- 在确实需要使用通配符时，也尽量不要吧它们用在搜索模式的开头出。把通配符置于开始处，搜索起来是最慢的
- 仔细注意通配符的位置。如果放错位置，可能不会反悔想要的数据

6.创建计算字段

计算字段

存储在数据库表中的数据一般不是应用程序所需要的格式。

计算字段并不是急存在于数据库表中。计算字段是运行时在SELECT语句内创建的

在SQL中的SELECT语句中，可使用一个特殊的操作符来拼接两个列。此操作符可用加号（+）或两个竖杠（||）表示。在MySQL和MariaDB中必须使用特殊的函数。

```
1  SELECT vend_name + '(' + vend_country + ')' FROM Vendors ORDER BY
   vend_name;
```

||可替换加号实现同样效果，具体看DBMS所支持的语法。

下面是使用MySQL或MariaDB时需要使用的语句：

```
1  SELECT Concat(vend_name , '(',vend_country,')') FROM Vendors ORDER BY
   vend_name;
```

说明：TRIM函数

大多数DBMS都支持RTRIM()、LTRIM()以及TRIM()，来去掉对应位置的空格

使用别名

别名是一个字段或值的替换名。别名用AS关键字赋予。

```
SELECT RTIM(vend_name) + '(' + RTRIM(vend_country) + ')'
AS vend_title
FROM Vendors
ORDER BY vend_name;
```

会输出列名及行

7.使用函数处理数据

函数	说明
LEFT()	返回字符串左边的字符
LENGTH()	返回字符串的长度
LOWER()	将字符串转换为小写
LTRIM()	去掉字符串左边的空格
RIGHT()	返回字符串右边的字符
RTRIM()	去掉字符串右边的空格
SUBSTR()或SUBSTRING()	提取字符串的组成部分
SOUNDEX()	返回字符串的SOUNDEX值
UPPER()	将字符串转换为大写

SOUNDEX是一个将任何文本串转换为描述其语音表示的字母数字模式的算法

日期和时间处理函数

DATEPART()此函数返回日期的某一部分

数值处理函数

函数	说明
ABS()	返回一个数的绝对值
COS()	返回一个角度的余弦
EXP()	返回一个数的指数值
PI()	返回圆周率 π的值
SIN()	返回一个角度的正弦
SQRT()	返回一个数的平方根
TAN()	返回一个角度的正切

具体DBMS所支持的算术处理函数，请参阅相应的文档

8.汇总数据

聚集函数

对某些行运行的函数，计算并返回一个值

函数	说明
AVG()	返回某列的平均值
COUNT()	返回某列的行数
MAX()	返回某列的最大值
MIN()	返回某列的最小值
SUM()	返回某列值之和

AVG()函数

AVG()通过对表中行数计数并计算其列值之和，求得该列的平均值。

Note：AVG函数只能用于单列，且忽略NULL的行

COUNT()函数

COUNT函数用来进行计数。可利用COUNT()确定表中行的树木或符合特定条件的行的数目

COUNT函数有两种使用方式

- 使用COUNT(*)对表中的数目进行计数，不管表列中包含的是空值还是非空值
- 使用COUNT(column)对特定列具有值的行进行计数，忽略NULL值

```
1  SELECT COUNT(*) AS num_cst FROM Customers;
```

其他函数同理

聚集不同的值

以上五个函数都可以如下使用

- 对所有行执行计算，指定ALL参数或不指定参数
- 只包含不同的值，指定DISTINCT参数
- ALL为默认，不需要指定

例如

```
SELECT AVG(DISTINCT pord_price) AS avg_price
FROM Products
WHERE vend_id = 'DLL01';
```

NOTE:

DISTINCT不能用于COUNT(*)

组合聚集函数

```
SELECT COUNT(*) AS num_items,
       MIN(prod_price) AS price_min,
       MAX(prod_price) AS price_max,
       AVG(prod_price) AS price_avg
FROM Products;
```

9.分组数据

GROUP BY 子句和HAVING子句

数据分组

```
SELECT vend_id,COUNT(*) AS num_prods
FROM Products
GROUP BY vend_id;
```

- GROUP BY子句可以包含任意数目的列，因而可以对分组进行嵌套，更细致地进行数据分组
- 如果在GROUP BY子句中嵌套了分组，数据将在最后指定的分组上进行汇总。即在建立分组时，指定的所有列都一起计算

- GROUP BY子句列出的每一列都必须是检索列或有效的表达式（不能是聚集函数）。若在SELECT中使用表达式，则必须在GROUP BY子句中指定相同的表达式，不能使用别名
- 大多数SQL实现不允许GROUP BY列带有长度可变的数据类型
- 除聚集计算语句外，SELECT语句中的每一列都必须在GROUP BY子句中给出
- 如果分组列中包含具有NULL值的行，则NULL将作为一个分组返回。如果列中有多行NULL值，它们将分为一组
- GROUP BY子句必须出现在WHERE子句之后，ORDER BY子句之前

过滤分组

WHERE子句过滤指定的是行而不是分组，且WHERE没有分组的概念。SQL为过滤分组提供了另一个子句——HAVING子句

HAVING非常类似于WHERE，并且目前为止所学过的WHERE子句都可以用HAVING代替。唯一的差别是WHERE过滤行，而HAVING过滤分组

Note：HAVING支持所有的WHERE操作符，句法是相同的，只是关键字有差别。

例如

```
SELECT cust_id,COUNT(*) AS orders
FROM Orders
GROUP BY cust_id
HAVING COUNT(*) >=2;
```

此处HAVING子句过滤COUNT(*)>=2的那些分组。

说明：HAVING和WHERE的差别

另一种理解方法，WHERE在数据分组前进行过滤，HAVING在数据分组后进行过滤。这是一个重要的区别，WHERE排除的行不包括在分组中。而WHERE和HAVING是可以同时使用的

例如：列出具有两个以上产品且其价格大于等于4的供应商

```
SELECT vend_id,COUNT(*) AS num_prods
FROM Products
WHERE prod_price >= 4
GROUP BY vend_id
HAVING COUNT(*) >= 2;
```

WHERE子句过滤所有prod_price >= 4的行，然后按vend_id分组，HAVING子句过滤分组中计数 >=2 的分组。

也即先过滤行数据，然后将数据分组后在过滤。

分组和排序

GROUP BY 和 ORDER BY 经常完成相同的工作，但这两者非常不同。

ORDER BY	GROUP BY
对产生的输出排序	对行分组，但输出可能不是分组的顺序
任意列都可以使用	只可能使用选择列货表达式列，而且必须使用每个选择列表达式
不一定需要	如果与聚集函数一起使用列，则必须使用

Note：一般在使用GROUP BY 子句时，应该也给出ORDER BY子句。这是保证数据正确排序的唯一方法。不要依赖GROUP BY排序数据

回顾SELECT子句顺序

子句	说明	是否必须使用
SELECT	要返回的列或表达式	是
FROM	从中检索数据的表	仅在从表选择数据时使用
WHERE	行级过滤	否
GROUP BY	分组说明	仅在按组计算聚集时使用
HAVING	组级过滤	否
ORDER BY	输出排序顺序	否

10.使用子查询

SQL允许创建子查询，即嵌套在其他查询中的查询。

利用子查询进行过滤

```
SELECT cust_id
FROM Orders
WHERE order_num IN (SELECT order_num
                    FROM OrderItems
                    WHERE prod_id = 'RGAN01');
```

在SELECT吃语句中，子查询总是从内向外处理。在处理上面的SELECT语句时，DBMS实际上执行了两个操作

首先执行圆括号中的查询，之后在圆括号中的结果中查询cust_id

Note：作为子查询的SELECT语句只能查询单个列，若企图检索多个列将返回错误。

子查询并不总是执行着泪数据检索的最有效办法，即性能会有所损失。

作为计算字段使用子查询

11.联结表

什么是联结

SQL最强大的功能之一就是能在数据查询的执行中联结（join）表

关系表

关系表的设计就是要把信息分解成多个表，一类数据一个表。各表通过某些共同的值互相关联。

为什么使用联结

联结是一种机制，用来在一条SELECT语句中关联表，因此成为联结。使用特殊的语法，可以联结多个表返回一组输出，联结在运行时关联表中正确的行。

创建联结

例

```
1  SELECT vend_name,prod_name,prod_price
2  FROM Vendors, Products
3  WHERE Vendors.vend_id = Products.vend_id;
```

NOTE：由没有联结条件的表关系返回的结果是笛卡尔积。检索出的行的数目将是第一个表中的行数乘以第二个表中的行数。

12.创建高级联结

使用表别名

```
SELECT RTRIM(vend_name) + '(' +RTRIM(vend_country) +')'
        AS vend_title
FROM Vendors
ORDER BY vend_name;
```

使用不同类型的联结

以上描述的联结只是内联结或等值联结的简单联结。还另有其他三种联结：自联结（self join）、自然联结（natural join）和外联结（outer join）

自联结

```
SELECT cust_id,cust_name,cust_contact
FROM Customers
WHERE cust_name =(SELECT cust_name
                  FROM Customers
                  WHERE cust_contact = 'Jim Jones');
```

```
SELECT c1.cust_id,c1.cust_name,c1.cust_contact
FROM Customers AS c1, Customers AS c2
WHERE c1.cust_name = c2.cust_name
AND c2.cust_contact = 'Jim Jones';
```

以上两段查询的结果都是相同的，第一段查询是使用子查询作为计算字段查询，第二段查询是使用表别名作自联结（Oracle 没有AS关键字）。

Note：用自联结而不用子查询，自联结通常作为外部语句，用来替代从相同表中检索数据的使用子查询语句。虽然最终结果相同，但许多DBMS处理联结的速度远比子查询的速度快很多

自然联结

自然联结要求你只能选择那些唯一的列，一般通过对一个表使用*通配符而对其他表的列使用明确的子集完成。

例

```
SELECT C.*,O.order_num,O.order_date,OI.prod_id,OI.quantity,OI.item_price
FROM Customers AS C, Orders AS O,
    OrderItems AS OI
WHERE C.cust_id =O.cust_id
AND OI.order_num = O.order_num
AND prod_id = 'RGAN01';
```

外联结

联结包含了那些在相关表中没有关联行的行。这种联结称为外联结

下面的SELECT语句给出了一个简单的内联结。它检索所有顾客机器订单：

```
SELECT Customers.cust_id , Orders.order_num
FROM Customers
INNER JOIN Orders ON Customers.cust_id = Orders.cust_id;
```

而外联结语法类似。要检索包括没有订单顾客在内

```
SELECT Customers.cust_id , Orders.order_num
FROM Customers
LEFT OUTER JOIN Orders ON Customers.cust_id = Orders.cust_id;
```

与内联结关联两个表中的行不同的是，外联结还包括没有关联行的行。在使用outer join语法时，必须使用RIGHT或LEFT关键字指定包括其所有行的表RIGHT和LEFT分别指outer join 右边和左边的表，上述例子中left outer join指出的是 outer join 从 from子句左边的表中选择所有的行。

13.插入数据

INSERT关键字用来将行插入到数据库表

其中几种方式

- 插入完整的行
- 插入行的一部分
- 插入某些查询的结果

插入完整的行

```
INSERT INTO Customers
VALUES(10000000006,
       'Tony Land',
       '123 Any Street',
       'New York',
       'NY',
       '11111',
       'USA',
       NULL,
       NULL);
```

此例将一个新顾客插入到Customers表中。

Note：不能插入同一条记录两次

必须给每一列都提供一个列值，否则会产生一定错误信息。且插入的列的字段得符合创建表的字段要求。

插入部分行

即使用INSERT 语句指定某些列，提供这些列的列值，不插入的列不指出。要求该列定义允许NULL值（无值或空值）

插入检索出的数据

即在INSERT语句中使用SELECT语句进行查询

```
1  INSERT INTO Customers(列名)
2  SELECT 列名 FROM CustNew;
```

从一个表复制到另一个表

使用CREATE SELECT进行嵌套

```
1  CREATE TABLE CustCopy AS SELECT * FROM Customers;
```

14.更新和删除数据

更新数据

更新或修改表中的数据，可以使用UPDATE语句。有两种使用UPDATE的方式

- 更新表中的特定行
- 更新表中的所有行

Note：不要省略WHERE子句在使用UPDATE时要细心，稍不注意就会更新表中的其他行甚至所有行，且在修改时需要有足够的权限

```
1  UPDATE Customers
2  SET cust_email = 'kim@thetoystore.com'
3  WHERE cust_id = 10000005;
```

UPDATE语句总是要以要更新的表名开始，SET命令用来将新值赋给被更新的列。

而更新多个列的语法略有不同

```
UPDATE Customers
SET cust_contact = 'Sam Roberts',
    cust_email = 'sam@toyland.com'
WHERE cust_id = 10000000006;
```

删除数据

从一个表中删除数据，使用DELETE语句。有两种使用DELETE语句的方式：

- 从表中删除特定的行
- 从表中删除所有行

Note：同理不要省略WHERE子句，否则可能会遇到问题，且有一定的权限。

```
DELETE FROM Customers
WHERE cust_id = 10000000006;
```

DELETE删除的是表的内容而不是表本身，删除表需要使用DROP关键字

Note：更快的删除表中所有行，可以使用TRUNCATE TABLE语句。

15.创建和操纵表

创建表

利用CREATE TABLE 创建表，必须给出下列信息：

- 新表的名字，在关键字CREATE TABLE之后给出；
- 表列的名字和定义，用逗号分隔；
- 有的DBMS还要求指定表的位置。

```
CREATE TABLE Products (
  prod_id CHAR(10) NOT NULL,
  vend_id CHAR(10) NOT NULL,
  prod_name CHAR(254) NOT NULL,
  prod_price DECIMAL(8,2) NOT NULL,
  prod_desc VARCHAR(1000) NULL
);
```

表名紧跟CREATE TABLE 关键字

更新表

使用ALTER TABLE 语句

- 理想情况下，不要再表中包含数据时对其进行更新。应该在表的设计过程中充分考虑未来可能的需求，避免今后对表的结构做大的改动
- 所有的DBMS都允许给现有的表增加列，不过对所增加列的数据类型有所限制
- 许多DBMS不允许删除或更改表中的列

删除表

```
DROP TABLE CustCopy;
```

重命名表

这个操作不存在严格的标准，需要查阅对应DBMS的相关文档以获得帮助。

16.使用视图

视图

视图是虚拟的表，与包含数据的表不一样，视图只包含使用时动态检索数据的查询

SQLite仅支持只读视图

为什么使用视图

视图的常见应用

- 重用SQL语句
- 简化复杂的SQL操作。在编写查询后，可以方便地重用它而不必知道其基本查询细节
- 使用表的一部分而不是整个表
- 保护数据。可以授予用户访问表的特定部分的权限，而不是整个表的访问权限。
- 更改数据格式和表示。视图可返回与底层表的表示和格式不同的数据

视图的规则和限制

- 与表一样，视图必须唯一命名
- 对于可以创建的视图数目没有限制
- 创建视图，必须具有足够的访问权限
- 视图可以嵌套，即可以利用从其他视图中检索数据的查询来构造视图
- 许多DBMS禁止在视图查询中使用ORDER BY子句
- 有些DBMS要求对返回的所有列进行命名，如果列是计算字段，则需要使用别名，=。
- 视图不能索引，也不能有关联的触发器或默认值

创建视图

使用CREATE VIEW语句进行创建，与CREAT TABLE语句相同

且视图可以重命名，删除视图使用DROP TABLE 语句

利用视图简化复杂的联结

```
CREATE VIEW ProductCustomers AS
SELECT cust_name,cust_contact,prod_id
FROM Customers,Orders,OrderItems
WHERE Customers.cust_id = Orders.cust_id
AND OrderItems.order_num = Orders.order_num;
```

分析：此段语句创建了一个名为Product Customers的视图，它联结三个表，返回已订购了任意产品的所有顾客的列表。

17.使用存储过程

存储过程

存储过程就是为以后使用而保存的一条或多条SQL语句，可将其视为批文件，虽然他的作用不仅限于批处理

Note：不适用SQLite

为什么使用存储过程

- 通过把处理封装在一个易用的单元中，可以简化复杂的奥作
- 由于不要求反复建立一系列处理步骤，因而保证了数据的一致性，
- 需要执行的步骤越多，越容易出现错误，存储过程可以在一定程度上防止错误
- 简化对变动的管理
- 因为存储过程通常以编译过的形式存储，所以DBMS处理命令所需的工作量少，提高了性能
- 存在一些只能用在单个请求中的SQL元素和特性，存储过程可以使用它们来编写功能更强更灵活的代码

执行存储过程

使用EXECUTE 关键字

```
EXECUTE AddNewProduct('JTS01',
                      'Stuffed Eiffel Tower',
                      6.49,
                      'Plush stuffed toy with
                      the text La Tour Eiffel in red white and blue');
```

此处执行一个名为AddNewProduct的存储过程，将一个新产品添加到Product表中，其中有四个参数。

创建存储过程

```
CREATE PROCEDURE MailingListCount (
 ListCount OUT INTEGER
 )
 IS
 v_rows INTEGER;
 BEGIN
      SELECT COUNT(*) INTO v_rows
      FROM Customers
      WHERE NOT cust_email IS NULL;
      ListCount := v_rows;
 END;
```

这个存储过程有一个名为ListCount参数，此参数从存储过程返回一个值而不是传递一个值给存储过程。

关键字OUT用来指示这种行为

Oracle支持IN（传递值给存储过程）、OUT（从存储过程返回值）

存储过程的代码括在BEGIN和END语句中，这里执行一条简单的SELECT 语句，它检索具有邮件地址的顾客。然后用检索出的行数设置ListCount

18.事务处理

使用事务处理，通过确保成批的SQL操作要么完全执行，要么完全不执行，来维护数据库的完整性。

专业术语：

- 事物（transaction）指一组SQL语句；
- 回退（rollback）指撤销指定SQL语句的过程；
- 提交（commit）指将未存储的SQL语句结果写入数据库表
- 保留点（savepoint）指事物处理中设置的临时占位符（placeholder），可以对它发布会退

控制事物处理

```
1  BEGIN TRANSACTION
2  ...
3  COMMIT TRANSACTION
```

在MariaDB和MySQL中等同的代码为：

```
1  START TRANSACTION
2  ...
```

Oracle中的用法

```
1  SET TRANSACTION
```

使用ROLLBACK

ROLLBACK命令用来回退撤销SQL语句

```
1  DELETE FROM Orders;
2  ROLLBACK
```

此例子中，用delete操作，然后用rollback语句撤销

使用COMMIT

一般的SQL语句都是针对数据库表直接执行和编写的。这就是所谓的隐式提交，即提交是自动进行的。

```
1  BEGIN TRANSACTION
2  DELETE OrderItems WHERE order_num = 12345
3  DELETE Order WHERE order_num = 12345
4  COMMIT TRANSACTION
```

在这个例子中，从系统中完全删除订单12345.因为涉及更新两个数据库表orders和ordersitems,所以使用事务处理块来保证订单不会被部分删除，最后COMMIT语句仅在不出错时写出更改。

使用保留点

在MariaDB、MySQL和Oracle中创建占位符，可使用SAVEPOINT语句

```
1  SAVEPOINT deletel;
```

19. 使用游标

游标

需要在检索出来的行中前进或后退一行或多行，这就是游标的用途所在。游标cursor是一个存储在DBMS服务器上的数据库查询，它不是一条SELETCT语句，而是被改语句检索出来的结果集。在存储了游标之后，应用程序可以根据需要滚动或浏览其中的数据。

游标常见的特性

- 能够标记游标为只读，使数据能读取，但不能更新和删除。
- 能控制可以执行的定向操作
- 能标记某些列尾可编辑的，某些列为不可编辑的。
- 规定范围，使游标对创建它的特定请求或对所有请求可访问。
- 只是DBMS对检索出的数据进行复制，使数据在游标打开和访问期间不变化

使用游标

- 在使用游标前，必须定义它。这个过程实际上没有检索数据
- 一旦声明，就必须打开游标以供使用。这个过程用前面定义的SELECT语句把数据实际检索出来
- 对于填有数据的游标，根据需要取出各行
- 在结束游标使用时，必须关闭游标，可能的话，释放游标

声明游标后，可根据需要频繁地打开和关闭游标。

创建游标

使用DECLARE语句创建游标，这条语句在不同的DBMS中有所不同。DECLARE命名游标，并定义相应的SELECT语句，根据需要带WHERE和其他子句。

```
1  DECLARE CustCursor CURSOR
2  FOR
3  SELECT * FROM Customers
4  WHERE cust_email IS NULL;
```

SELECT语句定义一个包含没有电子邮件地址的所有顾客的游标。

定义游标之后，就可以打开它了。

使用游标

```
1  OPEN CURSOR CustCursor
```

在处理OPEN CURSOR语句时，执行查询，存储检索出的数据以供浏览和滚动。

现在可以用FETCH语句访问游标数据了。FETCH指出要检索哪些行，从何处检索它们以及将它们放于何处。

```
1  DECLARE TYPE CustCursor IS REF CURSOR
2    RETURN Customer%ROWTYPE;
3  DECLARE CustRecord Customers%ROWTYPE
4  BEGIN
5    OPEN CustCursor;
6    FETCH CustCursor INTO CustRecord;
7    CLOSE CustCursor;
8  END;
```

关闭游标

```
1  CLOSE CutCursor;
```

20.高级SQL特性

约束

管理如何插入或处理数据库数据的规则

主键

- 任意两行的主键值都不相同
- 每行都具有一个主键值
- 包含主键值的列从不修改或更新
- 主键值不能重用。如果从表中删除某一行，其主键值不分配给新行。
- PRIMARY KEY

外键

外键是表中的一列，其值必须列在另一表的主键中。外键是保证因哟个完整性的极其重要部分

```
1  REFERENCES keyword
```