如何编写SQL查询

了解如何使用 SELECT、FROM、JOIN、WHERE、GROUP BY、HAVING、ORDER BY、OFFSET 和 FETCH 使用 SQL 检索数据。

> 译自[How to Write SQL Queries](http://cxyroad.com/”https://thenewstack.io/how-to-write-sql-queries/”)，作者 Gerald Venzl。

SQL 是一种类似英语的声明式领域语言，用于查询、分析和操作数据。SQL 起源于[关系数据库](http://cxyroad.com/ ”https://thenewstack.io/json-and-relational-tables-how-to-get-the-best-of-both/”)，但此后已在其他地方被广泛采用。SQL 被认为是一种声明式语言，这意味着用户声明他们想要_什么_结果，而不是_如何_获得这些结果（后者是命令式编程语言的方法，例如 C、Java 和[Python](http://cxyroad.com/”https://thenewstack.io/python/”)）。正因为如此，以及几乎可以将 SQL 语句读作英语句子，因此 SQL 通常被视为用于分析数据的最佳高级声明式编程语言之一，因为它具有[易于学习的语法](http://cxyroad.com/”https://thenewstack.io/how-to-make-sql-easier-to-understand-test-and-maintain/”)。

SQL 具有不同的语言元素，在高级别上可以分为**查询**和**数据操作**。SQL 查询使用`SELECT`语句，而用于数据操作的 SQL 使用`INSERT`、`UPDATE`、`DELETE`和`MERGE`语句。数据操作语句统称为**数据操作语言**或 DML。

本文将分解 SQL 查询语言的结构，而本系列的第二部分将描述 DML。

SQL 查询可能是 SQL 中最常用的操作，因为它们允许用户从一个或多个表中检索和分析数据。SQL 查询语句包括以下元素：

* `SELECT`和`FROM`
* 不带`FROM`的`SELECT`
* `JOIN`
* `WHERE`
* `GROUP BY`
* `HAVING`

* `ORDER BY`
* `OFFSET`
* `FETCH`
* `OFFSET`和`FETCH`

`SELECT`语句包含几个元素，但只有前两个是必需的：`SELECT`和`FROM`。但是，包括[Oracle 数据库](http://cxyroad.com/”https://www.oracle.com/database/?source=:ex:pw:::::TheNewStack_A&SC=:ex:pw:::::TheNewStack_A&pcode=”)和 MySQL 在内的某些数据库使`FROM`子句可选，如果`SELECT`仅引用自包含表达式，例如`SELECT 1;SELECT sysdate;`和`SELECT my_function();`。在这些情况下，数据不是从表中派生的，因此`FROM`不是必需的。

可选组件通过在它们周围放置`[]`来表示。

```
SELECT <expressions>
  FROM <table or sub-query>
    [ JOIN <to other table> ON <join condition> ]
  [ WHERE <predicates> ]
  [ GROUP BY <expressions>
    [ HAVING <predicates> ] ]
  [ ORDER BY <expressions> ]
  [ OFFSET ]
  [ FETCH ]
```

一个常见的误解是，这些组件按照它们在查询中出现的顺序执行。事实并非如此，因为`SELECT`组件在`HAVING`子句之后处理。以下列出了子句的处理顺序及其目的：

1. **FROM：** 指示从哪些表检索数据。**FROM** 子句确定正在检索的数据的工作集。这通常是指一个表，但也可以包括一个子查询（另一个`SELECT`查询，充当当前查询的输入源）。
2. **JOIN：** 指定连接多个表的规则。**JOIN** 子句是`FROM`子句的一部分，并将来自多个表的数据合并到一个数据集中。它是关系模型的基本运算符之一，用于将不同的关系合并到一个集合中。**JOIN**子句允许连接条件，以确保只有逻辑上属于一起的行才连接（具有匹配主键 -> 外键关系的行）。可以指定多个`JOIN`子句以将多个表连接到数据集中。因为`JOIN`子句是`FROM`子句的一部分，所以不能在查询中指定它而没有前面的`FROM`语句。
3. **WHERE：** 过滤查询返回的行。**WHERE** 子句根据提供的_谓词_或筛选条件筛选数据集，并丢弃所有不匹配它们的行的。它缩小了结果范围

，例如，检索 Europe 大陆的所有`countries`，而不是世界上的所有国家。
4. **GROUP BY：** 将具有指定列中公共值的行的聚合（或分组）到一行中。**GROUP BY**子句将具有公共值的行的聚合到一行中，因此行数将与唯一值的数量一样多。对于未在`GROUP BY`中指定的列的值，`SELECT`子句中的聚合函数需要按组聚合这些值。
5. **HAVING：** 过滤由**GROUP BY**子句生成的行。因此，它是**GROUP BY**的一部分，不能在查询中指定它而没有前面的**GROUP BY**语句。
6. **SELECT：** 定义查询结果输出中显示的列和表达式的列表。SELECT 子句计算任何表达式，并定义要返回或作为查询结果投影的列的列表。
7. **ORDER BY：** 标识用于对结果数据排序的列，以及对它们进行排序的方向（升序或降序）。如果省略 ORDER BY，则 SQL 查询返回的行顺序是未定义的。
8. **OFFSET：** 指定在返回数据之前在结果集中跳过的行数。
9. **FETCH：** 指定从结果返回的行数。

使用 SQL 查询

现在您已经熟悉了各种 SQL 查询子句的含义，就可以开始使用它们了。您可以使用我的[GitHub 存储库](http://cxyroad.com/”https://github.com/gvenzl/sample-data/tree/main/countries-cities-currencies”)中的数据模型来完成这些练习。

SELECT 和 FROM

在最简单的形式中，SQL 查询由 SELECT 和 FROM 子句组成：

```
SQL> SELECT *
  2*  FROM regions;

REGION_ID NAME
_____ _____
AF        Africa
AN        Antarctica
AS        Asia
EU        Europe
NA        North America
OC        Oceania
SA        South America
```

7 rows selected.

```

此查询从名为 regions 的表中选择所有行和所有列（如 SELECT 后面的 \* 所示，它表示“所有列”）。如果您想返回给定的列列表，则可以具体地调用它们：

```
SQL> SELECT name
  2* FROM regions;

NAME
____
Africa
Antarctica
Asia
Europe
North America
Oceania
South America

7 rows selected.
```

### 不带 FROM 的 SELECT

该 SELECT 语句还可以计算表达式，例如，1+2。从技术上讲，常量 1 和常量 2 都不来自任何表，但 ISO SQL 标准仍然需要 FROM 子句。许多数据库都有“虚拟”表来启用此类查询，例如 Oracle Database 中的 dual 表。

```
SQL> SELECT 1+2
2* FROM dual;
1+2
_____
3
```

但是，包括 Oracle Database 在内的许多数据库已经放宽了 SQL 标准中的此限制，并允许查询在这种情况下省略 FROM 子句：
```

```
SQL> SELECT 1+2;
1+2

_____
3

```

JOIN

关系模型完全是关于规范化数据，即把独立数据放入单独的表中，并在这些表之间定义_关系_。要重新组合规范化数据，可以使用_联接_将这些表重新联接在一起。

以下示例有两个表：先前查询的 regions 表和新的 countries 表。要编写一个将两个表联接到一个结果中的查询，请使用 JOIN 子句。如果没有 JOIN 子句，如果您在 FROM 子句中指定两个表，则 regions 表中的每一行都将乘以 countries 表中的每一行。这通常称为_笛卡尔积_，是 SQL 初学者常犯的一个错误。例如：

```
SQL> SELECT r.name, c.name
  2*  FROM regions r, countries c;

NAME    NAME
_______ ____________________________________
Africa Kosovo
Africa Yemen
Africa South Africa
Africa Zambia
Africa Zimbabwe
Africa Andorra
Africa United Arab Emirates
Africa Afghanistan
Africa Antigua and Barbuda
Africa Albania
Africa Armenia
Africa Angola
Africa Argentina
Africa Austria
Africa Australia
```

...
...
...
South America Uzbekistan
South America Vatican City
South America Saint Vincent and the Grenadines
South America Venezuela
South America Vietnam
South America Vanuatu
South America Samoa

1,372 rows selected.

```

此查询的输出显然不正确。既没有 1,372 个国家，奥地利也不位于非洲。我们真正想要的是将 countries 表中的所有行与 regions 表**中的行联接起来，**region\_id**相同的地方**。这通常称为\_联接条件\_，可以在 JOIN 子句的一部分 ON 子句中指定：

```
SQL> SELECT r.name, c.name
  2   FROM regions r
  3 JOIN countries c
  4* ON (r.region_id=c.region_id);

NAME   NAME
_________ ____________________________________
Africa South Africa
Africa Zambia
Africa Zimbabwe
Africa Angola
Africa Burkina Faso
Africa Burundi
Africa Benin
...
...
...
South America Ecuador
South America Guyana
South America Peru
South America Paraguay
South America Suriname
South America Uruguay
South America Venezuela
```
```

196 rows selected.

```

这更接近我们想要的结果！

还有一件事需要注意：上面的查询指定 SELECT r.name, c.name 并将字母 r 和 c 放在表名旁边。这些是\_表别名\_，数据库需要它们来告诉您想要哪个表列。如果该语句只说 SELECT name, name，则不清楚该查询是指 regions 表列 name 还是 countries 表列 name。

### WHERE

该 WHERE 子句筛选由 FROM 子句生成的行。到目前为止，您始终会得到表中的所有行。如果您只想返回南美洲的所有国家，这就需要**WHERE 子句**。WHERE 子句用于匹配所有 regions.name 列为 'South America' 的行:

```
SQL> SELECT r.name, c.name
  2   FROM regions r
  3 JOIN countries c
  4 ON (r.region_id=c.region_id)
  5* WHERE r.name = 'South America';

NAME   NAME
_______________ _____________
South America Argentina
South America Bolivia
South America Brazil
South America Chile
South America Colombia
South America Ecuador
South America Guyana
South America Peru
South America Paraguay
South America Suriname
South America Uruguay
South America Venezuela

12 rows selected.

```
```

GROUP BY

GROUP BY 子句用于将多行聚合到一个组中，本质上将多行合并为一行。例如，countries 表包含一个名为 population 的列，但 regions 表不包含：

```
SQL> SELECT r.name, c.name, c.population
  2   FROM regions r
  3 JOIN countries c
  4 ON (r.region_id=c.region_id)
  5*  WHERE r.name = 'South America';

NAME          NAME          POPULATION
_____________ _____________ ____________
South America Argentina      44694000
South America Bolivia        11306000
South America Brazil        208847000
South America Chile          17925000
South America Colombia       48169000
South America Ecuador        16291000
South America Guyana           741000
South America Peru           31331000
South America Paraguay        7026000
South America Suriname         598000
South America Uruguay         3369000
South America Venezuela      31689000

12 rows selected.
```

一个常见的业务问题可能是：“每个地区的总人口是多少？”鉴于 regions 表没有包含该信息的列，答案只能通过计算每个地区每个国家/地区的 population 列的总和来提供。因此，您需要一种机制，将 countries 表的 196 行根据其地区放入七个组或存储区（因为 regions 表中有七行）。但是，该查询不能仅仅将 196 行放入七行；它需要根据属于该地区的国家/地区的人口计算每个地区的总人口。

这可以通过对 population 列应用 SUM() 聚合函数来完成：

```
SQL> SELECT r.name, SUM(c.population)
```

```
  2   FROM regions r
  3 JOIN countries c
  4 ON (r.region_id=c.region_id)
  5*  GROUP BY r.name;

NAME                 SUM(C.POPULATION)
_____________________ _____________________
Africa                1263685000
Asia                  4439011000
Europe                 748985000
North America          575767000
Oceania                 37556000
South America          421986000

6 rows selected.
```

此查询显示了其他一些有趣的内容。尽管在 regions 表中包含七个地区，但此查询只产生了六行。这是因为存在一个地区“南极洲”，但在 countries 表中没有该 region_id 的国家。因此，JOIN 子句会将该地区过滤掉（因为在 countries 表中没有符合 ON 子句所指定的 matching region_id）。

GROUP BY 子句并不需要任何 JOIN 子句；您可以在一个表中创建组。例如，“有多少个国家以相同字母开头？”也可以通过一个 GROUP BY 来回答。要执行此操作，请根据所有行的唯一第一个字母值创建与组一样多的组，方法是使用 SUBSTR() 函数，然后计算属于该组或类别中的行：

```
SQL> SELECT SUBSTR(name,1,1), COUNT(*)
  2   FROM countries
  3*  GROUP BY SUBSTR(name,1,1);

SUBSTR(NAME,1,1)   COUNT(*)
_____________________ _____________
K                     6
Y                     1
S                     26
Z                     2
A                     11
U                     7
B                     17
C                     17
D                     5
G                     11
```

```
E                              8
F                              3
M                               18
H                              3
I                             8
J                              3
N                               11
L                              9
O                               1
P                              9
Q                               1
R                              3
T                              12
V                              4

24 rows selected.
```

HAVING

HAVING 子句根据提供的谓词过滤 GROUP BY 子句产生的行。例如，如果您只想返回人口超过 5 亿的人口，则无法在 WHERE 子句中指定，因为 WHERE 子句在 GROUP BY 子句之前处理。因此，WHERE 子句没有地区人口的概念。这就是 HAVING 子句的用武之地。从逻辑角度来看，它的行为与 WHERE 子句相同，但它在不同的处理阶段进行过滤：

```
SQL> SELECT r.name, SUM(c.population)
  2   FROM regions r
  3 JOIN countries c
  4 ON (r.region_id=c.region_id)
  5   GROUP BY r.name
  6*   HAVING SUM(c.population) > (500 * 1000 * 1000);

NAME                 SUM(C.POPULATION)
_____________________ _____________________
Africa                1263685000
Asia                 4439011000
Europe                 748985000
North America             575767000
```

ORDER BY

ORDER BY 子句对结果数据进行排序。到目前为止，未定义的行排序已经奏效，除了“每个第一个字母的国家/地区”之外。ORDER BY 子句可用于按字母顺序返回行：

```
SQL> SELECT SUBSTR(name,1,1), COUNT(*)
  2   FROM countries
  3   GROUP BY SUBSTR(name,1,1)
  4*  ORDER BY SUBSTR(name,1,1);

SUBSTR(NAME,1,1)   COUNT(*)
_____________________ _____________
A                      11
B                      17
C                      17
D                      5
E                      8
F                      3
G                      11
H                      3
I                     8
J                      3
K                      6
L                      9
M                      18
N                      11
O                      1
P                      9
Q                      1
R                      3
S                      26
T                      12
U                      7
V                      4
Y                      1
Z                      2

24 rows selected.
```

默认情况下，行以升序排列，但你可以使用 DESC (DESCENDING) 关键字颠倒该顺序：

```
SQL> SELECT SUBSTR(name,1,1), COUNT(*)
  2   FROM countries
  3   GROUP BY SUBSTR(name,1,1)
  4*  ORDER BY SUBSTR(name,1,1) DESC;

SUBSTR(NAME,1,1)   COUNT(*)
_____________________ _____________
Z                     2
Y                     1
V                     4
U                     7
T                     12
S                     26
R                     3
Q                     1
P                     9
O                     1
N                     11
M                     18
L                     9
K                     6
J                     3
I                     8
H                     3
G                     11
F                     3
E                     8
D                     5
C                     17
B                     17
A                     11

24 rows selected.
```

OFFSET

OFFSET 子句指定在开始返回数据之前要跳过的行数。此子句是其他需要分析查询或子查询的简写。例如，询问“给我南美洲所有按平方公里排序的国家，但不要前三个”可以用以下方式回答：

```
SQL> SELECT c.name, c.area_sq_km
  2   FROM countries c
  3 JOIN regions r
  4 ON (c.region_id=r.region_id)
  5   WHERE r.name = 'South America'
  6   ORDER BY area_sq_km DESC
  7*  OFFSET 3 ROWS;

NAME          AREA_SQ_KM
_____________ _____________
Colombia        1138910
Bolivia         1098581
Venezuela        912050
Chile           756102
Paraguay         406752
Ecuador          283561
Guyana           214969
Uruguay          176215
Suriname         163820

9 rows selected.
```

FETCH

FETCH 子句指定从结果中返回的行数。一些数据库称之为 LIMIT 子句。与 OFFSET 子句一样，这也是一个简写，可用于回答诸如“按人口排名前三的国家/地区有哪些？”之类的业务问题。可以用以下方式回答：

```
SQL> SELECT name, population
  2   FROM countries
  3   ORDER BY population DESC
  4*  FETCH FIRST 3 ROWS ONLY;

NAME              POPULATION
_________________ _____________
China              1384689000
India              1296834000
United States       329256000
```

您可能想知道如果两行在第三个位置上相等会发生什么；两行都会返回吗？还是只有第一行？对于这些情况，FETCH 子句提供了 ONLY 和 WITH TIES 关键字。上面只使用了 ONLY，因为两个国家不太可能拥有相同的人口。

但是，按字母对国家进行排名时，重叠的空间更大。例如，在按国家/地区第一个字母的国家/地区示例中，按国家/地区数量进行排名时，很明显一些字母具有相同数量：

```
SQL> SELECT SUBSTR(name,1,1), COUNT(*)
  2   FROM countries
  3   GROUP BY SUBSTR(name,1,1)
  4*  ORDER BY COUNT(*) DESC;

SUBSTR(NAME,1,1)   COUNT(*)
_____________________ _________
S                      26
M                       18
B                      17
C                      17
T                      12
A                      11
N                      11
G                      11
L                      9
P                      9
I                     8
E                      8
U                      7
K                      6
D                      5
V                      4
J                     3
H                      3
F                     3
R                      3
Z                      2
Q                      1
Y                      1
O                      1

24 rows selected.
```

如果您对该查询运行相同的 FETCH 子句，则字母 C 将从结果中省略，尽管它与字母 B 具有完全相同数量的国家/地区：

```
SQL> SELECT SUBSTR(name,1,1), COUNT(*)
  2   FROM countries
  3   GROUP BY SUBSTR(name,1,1)
  4   ORDER BY COUNT(*) DESC
  5*  FETCH FIRST 3 ROWS ONLY;

SUBSTR(NAME,1,1)   COUNT(*)
_____________________ ___________
S                     26
M                      18
B                     17
```

此时，WITH TIES 关键字派上用场，因为它将在结果中包含平局：

```
SQL> SELECT SUBSTR(name,1,1), COUNT(*)
  2   FROM countries
  3   GROUP BY SUBSTR(name,1,1)
  4   ORDER BY COUNT(*) DESC
  5*  FETCH FIRST 3 ROWS WITH TIES;

SUBSTR(NAME,1,1)   COUNT(*)
_____________________ ___________
S                     26
M                      18
B                     17
C                     17
```

OFFSET 和 FETCH

组合 OFFSET 和 FETCH 子句允许另一个简洁的简写，否则需要分析查询或子查询。考虑以下问题：“按平方公里计算，地球上第二小的国家是什么？”可以通过组合 OFFSET 从第二行开始返回结果，以及 FETCH 仅获取第二行来回答此问题：

```
SQL> SELECT name, area_sq_km
  2    FROM countries
  3    ORDER BY area_sq_km
  4    OFFSET 1 ROW
  5*   FETCH FIRST 1 ROW ONLY;

NAME      AREA_SQ_KM
_________ _____________
Monaco              2

```

接下来的内容?

本系列中的第二篇文章将分解 SQL 数据操作语言 (DML) 的结构。您可以在我的[GitHub 存储库中找到本文和第二部分中使用的数据模型](http://cxyroad.com/ "https://github.com/gvenzl/sample-data/tree/main/countries-cities-currencies")。

> 本文在[云云众生](http://cxyroad.com/ "https://yylives.cc/")（[yylives.cc/](http://cxyroad.com/ "https://yylives.cc/")）首发，欢迎大家访问。

原文链接: https://juejin.cn/post/7352075703859331123