# 11 | SQL99是如何使用连接的，与SQL92的区别是什么?

上节课我们讲解了 SQL92 标准，在它之后又提出了 SQL99 标准。现在各大 DBMS 中对 SQL99 标准的支持度更好。你一定听说过 LEFT JOIN、RIGHT JOIN 这样的操作符，这实际上就是 SQL99 的标准，在 SQL92 中它们是用（+）代替的。SQL92 和 SQL99 标准原理类似，只是 SQL99 标准的可读性更强。

今天我就来讲解一下 SQL99 标准中的连接查询，在今天的课程中你需要重点掌握以下几方面的内容：

1. SQL99 标准下的连接查询是如何操作的?
2. SQL99 与 SQL92 的区别是什么?
3. 在不同的 DBMS 中，使用连接需要注意什么?

## SQL99 标准中的连接查询

上一篇文章中，我用 NBA 球员的数据表进行了举例，包括了三张数据表 player、team 和 height_grades。

其中 player 表为球员表，一共有 37 个球员，如下所示：

| player_id | team_id | player_name | height |
|---|---|---|---|
| 10001 | 1001 | 韦恩·艾灵顿 | 1.93 |
| 10002 | 1001 | 雷吉·杰克逊 | 1.91 |
| 10003 | 1001 | 安德烈·德拉蒙德 | 2.11 |
| 10004 | 1001 | 索恩·马克 | 2.16 |
| …… | …… | …… | …… |
| 10037 | 1002 | 伊凯·阿尼博古 | 2.08 |

team 表为球队表，一共有 3 支球队，如下所示：

| team_id | team_name |
| --- | --- |
| 1001 | 底特律活塞 |
| 1002 | 印第安纳步行者 |
| 1003 | 亚特兰大老鹰 |

height_grades 表为身高等级表，如下所示：

| height_level | height_lowest | height_highest |
| --- | --- | --- |
| A | 2.00 | 2.50 |
| B | 1.90 | 1.99 |
| C | 1.80 | 1.89 |
| D | 1.60 | 1.79 |

接下来我们看下在 SQL99 标准中，是如何进行连接查询的？

### 交叉连接

交叉连接实际上就是 SQL92 中的笛卡尔乘积，只是这里我们采用的是 CROSS JOIN。

我们可以通过下面这行代码得到 player 和 team 这两张表的笛卡尔积的结果：

复制代码

```
SQL: SELECT * FROM player CROSS JOIN team
```

运行结果（一共 37*3=111 条记录）：

| player_id | team_id | player_name | height | team_id(1) | team_name |
|---|---|---|---|---|---|
| 10001 | 1001 | 韦恩-艾灵顿 | 1.93 | 1001 | 底特律活塞 |
| 10001 | 1001 | 韦恩-艾灵顿 | 1.93 | 1002 | 印第安纳步行者 |
| 10001 | 1001 | 韦恩-艾灵顿 | 1.93 | 1003 | 亚特兰大老鹰 |
| …… | …… | …… | …… | …… | …… |
| 10037 | 1002 | 伊凯·阿尼博古 | 2.08 | 1003 | 亚特兰大老鹰 |

如果多张表进行交叉连接，比如表 t1，表 t2，表 t3 进行交叉连接，可以写成下面这样：

复制代码

```
SQL: SELECT * FROM t1 CROSS JOIN t2 CROSS JOIN t3
```

## 自然连接

你可以把自然连接理解为 SQL92 中的等值连接。它会帮你自动查询两张连接表中所有相同的字段，然后进行等值连接。

如果我们想把 player 表和 team 表进行等值连接，相同的字段是 team_id。还记得在 SQL92 标准中，是如何编写的么？

复制代码

```
SELECT player_id, a.team_id, player_name, height, team_name FROM player as a, team as b
```

在 SQL99 中你可以写成：

复制代码

```
SELECT player_id, team_id, player_name, height, team_name FROM player NATURAL JOIN team
```

实际上，在 SQL99 中用 NATURAL JOIN 替代了 `WHERE player.team_id = team.team_id`。

## ON 连接

ON 连接用来指定我们想要的连接条件，针对上面的例子，它同样可以帮助我们实现自然连接的功能：

复制代码

```
SELECT player_id, player.team_id, player_name, height, team_name FROM player JOIN team (
```

这里我们指定了连接条件是`ON player.team_id = team.team_id`，相当于是用 ON 进行了 team_id 字段的等值连接。

当然你也可以 ON 连接进行非等值连接，比如我们想要查询球员的身高等级，需要用 player 和 height_grades 两张表：

复制代码

```
SQL99：SELECT p.player_name, p.height, h.height_level
FROM player as p JOIN height_grades as h
ON height BETWEEN h.height_lowest AND h.height_highest

```

这个语句的运行结果和我们之前采用 SQL92 标准的查询结果一样。

复制代码

```
SQL92：SELECT p.player_name, p.height, h.height_level
FROM player AS p, height_grades AS h
WHERE p.height BETWEEN h.height_lowest AND h.height_highest
```

一般来说在 SQL99 中，我们需要连接的表会采用 JOIN 进行连接，ON 指定了连接条件，后面可以是等值连接，也可以采用非等值连接。

## USING 连接

当我们进行连接的时候，可以用 USING 指定数据表里的同名字段进行等值连接。比如：

复制代码

```
SELECT player_id, team_id, player_name, height, team_name FROM player JOIN team USING(te
```

你能看出与自然连接 NATURAL JOIN 不同的是，USING 指定了具体的相同的字段名称，你需要在 USING 的括号 () 中填入要指定的同名字段。同时使用 JOIN USING 可以简化 JOIN ON 的等值连接，它与下面的 SQL 查询结果是相同的：

复制代码

```
SELECT player_id, player.team_id, player_name, height, team_name FROM player JOIN team (
```

## 外连接

SQL99 的外连接包括了三种形式：

1. 左外连接：LEFT JOIN 或 LEFT OUTER JOIN
2. 右外连接：RIGHT JOIN 或 RIGHT OUTER JOIN
3. 全外连接：FULL JOIN 或 FULL OUTER JOIN

我们在 SQL92 中讲解了左外连接、右外连接，在 SQL99 中还有全外连接。全外连接实际上就是左外连接和右外连接的结合。在这三种外连接中，我们一般省略 OUTER 不写。

1. 左外连接

### SQL92

复制代码

```
SELECT * FROM player, team where player.team_id = team.team_id(+)
```

### SQL99

复制代码

```
SELECT * FROM player LEFT JOIN team ON player.team_id = team.team_id
```

2. 右外连接

### SQL92

复制代码

```
SELECT * FROM player, team where player.team_id(+) = team.team_id
```

**SQL99**

复制代码

```
SELECT * FROM player RIGHT JOIN team ON player.team_id = team.team_id
```

3. 全外连接

**SQL99**

复制代码

```
SELECT * FROM player FULL JOIN team ON player.team_id = team.team_id
```

需要注意的是 MySQL 不支持全外连接，否则的话全外连接会返回左表和右表中的所有行。当表之间有匹配的行，会显示内连接的结果。当某行在另一个表中没有匹配时，那么会把另一个表中选择的列显示为空值。

也就是说，全外连接的结果 = 左右表匹配的数据 + 左表没有匹配到的数据 + 右表没有匹配到的数据。

## 自连接

自连接的原理在 SQL92 和 SQL99 中都是一样的，只是表述方式不同。

比如我们想要查看比布雷克·格里芬身高高的球员都有哪些，在两个 SQL 标准下的查询如下。

**SQL92**

复制代码

```
SELECT b.player_name, b.height FROM player as a , player as b WHERE a.player_name = '布
```

**SQL99**

复制代码

```
SELECT b.player_name, b.height FROM player as a JOIN player as b ON a.player_name = '布
```

运行结果（6 条记录）：

| player_name | height |
| --- | --- |
| 安德烈·德拉蒙德 | 2.11 |
| 索恩·马克 | 2.16 |
| 扎扎·帕楚里亚 | 2.11 |
| 亨利·埃伦森 | 2.11 |
| 多曼塔斯·萨博尼斯 | 2.11 |
| 迈尔斯·特纳 | 2.11 |

## SQL99 和 SQL92 的区别

至此我们讲解完了 SQL92 和 SQL99 标准下的连接查询，它们都对连接进行了定义，只是操作的方式略有不同。我们再来回顾下，这些连接操作基本上可以分成三种情况：

1. 内连接：将多个表之间满足连接条件的数据行查询出来。它包括了等值连接、非等值连接和自连接。
2. 外连接：会返回一个表中的所有记录，以及另一个表中匹配的行。它包括了左外连接、右外连接和全连接。
3. 交叉连接：也称为笛卡尔积，返回左表中每一行与右表中每一行的组合。在 SQL99 中使用的 CROSS JOIN。

不过 SQL92 在这三种连接操作中，和 SQL99 还存在着明显的区别。

首先我们看下 SQL92 中的 WHERE 和 SQL99 中的 JOIN。

你能看出在 SQL92 中进行查询时，会把所有需要连接的表都放到 FROM 之后，然后在 WHERE 中写明连接的条件。而 SQL99 在这方面更灵活，它不需要一次性把所有需要连接的表都放到 FROM 之后，而是采用 JOIN 的方式，每次连接一张表，可以多次使用 JOIN 进行连接。

另外，我建议多表连接使用 SQL99 标准，因为层次性更强，可读性更强，比如：

复制代码

```
SELECT ...
FROM table1
    JOIN table2 ON table1 和 table2 的连接条件
        JOIN table3 ON table2 和 table3 的连接条件
```

它的嵌套逻辑类似我们使用的 FOR 循环：

复制代码

```
for t1 in table1:
    for t2 in table2:
       if condition1:
           for t3 in table3:
              if condition2:
                  output t1 + t2 + t3
```

SQL99 采用的这种嵌套结构非常清爽，即使再多的表进行连接也都清晰可见。如果你采用 SQL92，可读性就会大打折扣。

最后一点就是，SQL99 在 SQL92 的基础上提供了一些特殊语法，比如 NATURAL JOIN 和 JOIN USING。它们在实际中是比较常用的，省略了 ON 后面的等值条件判断，让 SQL 语句更加简洁。

## 不同 DBMS 中使用连接需要注意的地方

SQL 连接具有通用性，但是不同的 DBMS 在使用规范上会存在差异，在标准支持上也存在不同。在实际工作中，你需要参考你正在使用的 DBMS 文档，这里我整理了一些需要注意

的常见的问题。

**1. 不是所有的 DBMS 都支持全外连接**

虽然 SQL99 标准提供了全外连接，但不是所有的 DBMS 都支持。不仅 MySQL 不支持，Access、SQLite、MariaDB 等数据库软件也不支持。不过在 Oracle、DB2、SQL Server 中是支持的。

**2.Oracle 没有表别名 AS**

为了让 SQL 查询语句更简洁，我们经常会使用表别名 AS，不过在 Oracle 中是不存在 AS 的，使用表别名的时候，直接在表名后面写上表别名即可，比如 player p，而不是 player AS p。

**3.SQLite 的外连接只有左连接**

SQLite 是一款轻量级的数据库软件，在外连接上只支持左连接，不支持右连接，不过如果你想使用右连接的方式，比如`table1 RIGHT JOIN table2`，在 SQLite 你可以写成`table2 LEFT JOIN table1`，这样就可以得到相同的效果。

除了一些常见的语法问题，还有一些关于连接的性能问题需要你注意：

**1. 控制连接表的数量**

多表连接就相当于嵌套 for 循环一样，非常消耗资源，会让 SQL 查询性能下降得很严重，因此不要连接不必要的表。在许多 DBMS 中，也都会有最大连接表的限制。

**2. 在连接时不要忘记 WHERE 语句**

多表连接的目的不是为了做笛卡尔积，而是筛选符合条件的数据行，因此在多表连接的时候不要忘记了 WHERE 语句，这样可以过滤掉不必要的数据行返回。

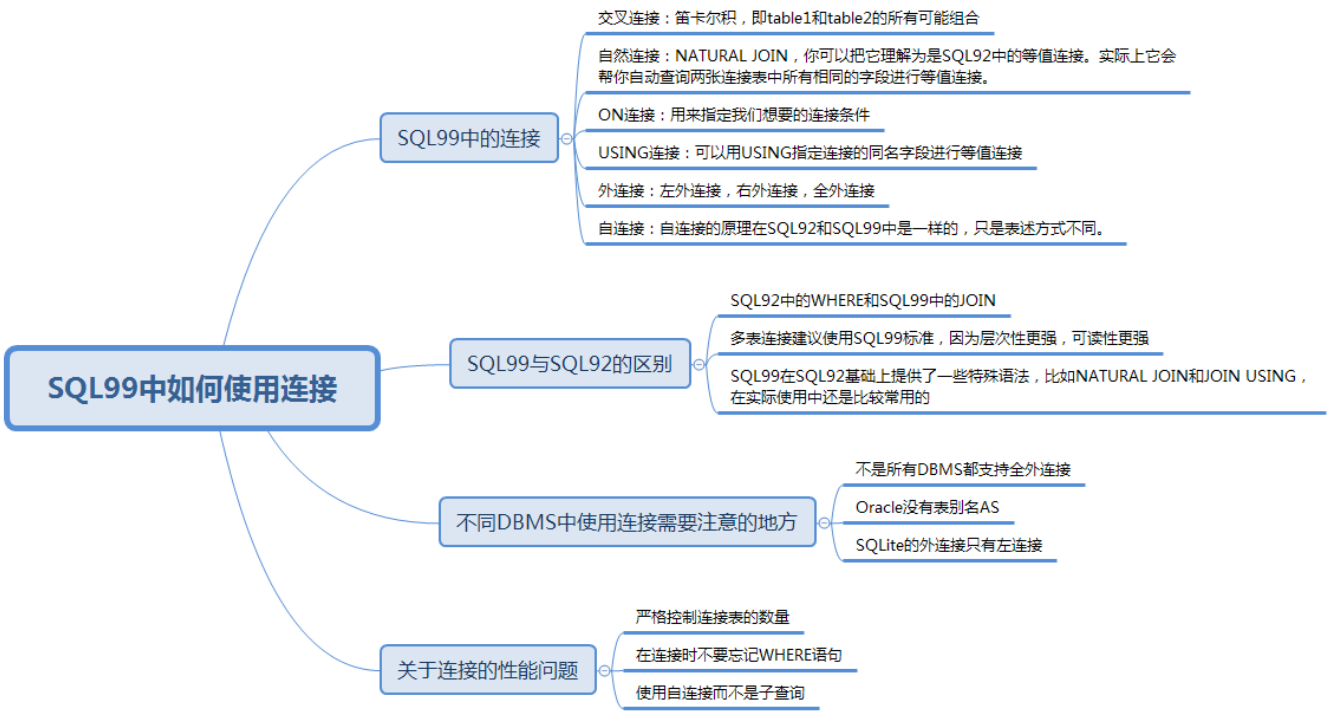
**3. 使用自连接而不是子查询**

我们在查看比布雷克·格里芬高的球员都有谁的时候，可以使用子查询，也可以使用自连接。一般情况建议你使用自连接，因为在许多 DBMS 的处理过程中，对于自连接的处理速度要比子查询快得多。你可以这样理解：子查询实际上是通过未知表进行查询后的条件判断，而自连接是通过已知的自身数据表进行条件判断，因此在大部分 DBMS 中都对自连接处理进行了优化。

## 总结

连接可以说是 SQL 中的核心操作，通过两篇文章的学习，你已经从多个维度对连接进行了了解。同时，我们对 SQL 的两个重要标准 SQL92 和 SQL99 进行了学习，在我们需要进行外连接的时候，建议采用 SQL99 标准，这样更适合阅读。

此外我还想强调一下，我们在进行连接的时候，使用的关系型数据库管理系统，之所以存在关系是因为各种数据表之间存在关联，它们并不是孤立存在的。在实际工作中，尤其是做业务报表的时候，我们会用到 SQL 中的连接操作（JOIN），因此我们需要理解和熟练掌握 SQL 标准中连接的使用，以及不同 DBMS 中对连接的语法规范。剩下要做的，就是通过做练习和实战来增强你的经验了，做的练习多了，也就自然有感觉了。




我今天讲解了 SQL99 的连接操作，不妨请你做一个小练习。请你编写 SQL 查询语句，查询不同身高级别（对应 height_grades 表）对应的球员数量（对应 player 表）。