SQL

1. SQL的执行顺序

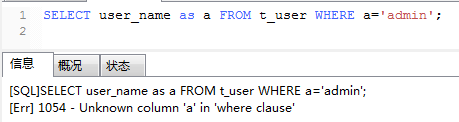
* FROM
* WHERE
* GROUP BY
* HAVING
* SELECT
* DISTINCT
* UNION
* ORDER BY

SELECT 是在大部分语句执行了之后才执行的，严格的说是在 FROM 和 GROUP BY 之后执行的。理解这一点是非常重要的，这就是你不能在 WHERE 中使用在 SELECT 中设定别名的字段作为判断条件的原因。

SELECT A.x + A.y AS z

FROM A

WHERE z = 10 -- z 在此处不可用，因为SELECT是最后执行的语句！



## 2.  SQL 语言的核心是对表的引用（table references）

FROM 语句的“输出”是一张联合表，来自于所有引用的表在某一维度上的联合。上面这句 FROM 语句的输出是一张联合表，联合了表 a 和表 b 。如果 a 表有三个字段， b 表有 5 个字段，那么这个“输出表”就有 8 （ =5+3）个字段。

这个联合表里的数据是 a\*b，即 a 和 b 的笛卡尔积。换句话说，也就是 a 表中的每一条数据都要跟 b 表中的每一条数据配对。如果 a 表有3 条数据， b 表有 5 条数据，那么联合表就会有 15 （ =5\*3）条数据。

FROM 输出的结果被 WHERE 语句筛选后要经过 GROUP BY 语句处理，从而形成新的输出结果。

## 3. SQL 语句中不同的连接操作

SQL 语句中，表连接的方式从根本上分为五种：

* EQUI JOIN
* SEMI JOIN
* ANTI JOIN
* CROSS JOIN
* DIVISION

**EQUI JOIN**

这是一种最普通的 JOIN 操作，它包含两种连接方式：

* INNER JOIN（或者是 JOIN ）
* OUTER JOIN（包括： LEFT 、 RIGHT、 FULL OUTER JOIN）

left join(左联接) 返回包括左表中的所有记录和右表中联结字段相等的记录   
right join(右联接) 返回包括右表中的所有记录和左表中联结字段相等的记录  
inner join(等值连接) 只返回两个表中联结字段相等的行

举例如下：   
--------------------------------------------  
表A记录如下：  
aID　　　　　aNum  
1　　　　　a20050111  
2　　　　　a20050112  
3　　　　　a20050113  
4　　　　　a20050114  
5　　　　　a20050115

表B记录如下:  
bID　　　　　bName  
1　　　　　2006032401  
2　　　　　2006032402  
3　　　　　2006032403  
4　　　　　2006032404  
8　　　　　2006032408

--------------------------------------------  
**1.left join**  
 sql语句如下:   
 select \* from A left join B on A.aID = B.bID

结果如下:

aID　　　　　aNum　　　　　bID　　　　　bName  
1　　　　　a20050111　　　　1　　　　　2006032401  
2　　　　　a20050112　　　　2　　　　　2006032402  
3　　　　　a20050113　　　　3　　　　　2006032403  
4　　　　　a20050114　　　　4　　　　　2006032404  
5　　　　　a20050115　　　　NULL　　　　　NULL

（所影响的行数为 5 行）  
结果说明:  
left join是以A表的记录为基础的,A可以看成左表,B可以看成右表,left join是以左表为准的.  
换句话说,左表(A)的记录将会全部表示出来,而右表(B)只会显示符合搜索条件的记录(例子中为: A.aID = B.bID).  
B表记录不足的地方均为NULL.  
--------------------------------------------  
**2.right join**  
 sql语句如下:   
 select \* from A right join B on A.aID = B.bID

结果如下:  
aID　　　　　aNum　　　　　bID　　　　　bName  
1　　　　　a20050111　　　　1　　　　　2006032401  
2　　　　　a20050112　　　　2　　　　　2006032402  
3　　　　　a20050113　　　　3　　　　　2006032403  
4　　　　　a20050114　　　　4　　　　　2006032404  
NULL　　　　　NULL　　　　　8　　　　　2006032408

（所影响的行数为 5 行）  
结果说明:  
仔细观察一下,就会发现,和left join的结果刚好相反,这次是以右表(B)为基础的,A表不足的地方用NULL填充.  
--------------------------------------------  
**3.inner join**  
 sql语句如下:   
select \* from A innerjoin B on A.aID = B.bID

结果如下:  
aID　　　　　aNum　　　　　bID　　　　　bName  
1　　　　　a20050111　　　　1　　　　　2006032401  
2　　　　　a20050112　　　　2　　　　　2006032402  
3　　　　　a20050113　　　　3　　　　　2006032403  
4　　　　　a20050114　　　　4　　　　　2006032404

结果说明:  
很明显,这里只显示出了 A.aID = B.bID的记录.这说明inner join并不以谁为基础,它只显示符合条件的记录.

**4.full outer join**  
全外连接返回参与连接的两个数据集合中的全部数据，无论它们是否具有与之相匹配的行。在功能上，它等价于对这两个数据集合分别进行左外连接和右外连接，然后再使用消去重复行的并操作将上述两个结果集合并为一个结果集。

**Semi join**

所谓的semi-join是指semi-join子查询。 当一张表在另一张表找到匹配的记录之后，半连接（semi-jion）返回第一张表中的记录。与条件连接相反，即使在右节点中找到几条匹配的记录，左节点 的表也只会返回一条记录。另外，右节点的表一条记录也不会返回。半连接通常使用IN  或 EXISTS 作为连接条件。 该子查询具有如下结构：

SELECT ... FROM outer\_tables WHERE expr IN (SELECT ... FROM inner\_tables ...) AND ...

即在where条件的“IN”中的那个子查询。  
这种查询的特点是我们只关心outer\_table中与semi-join相匹配的记录。  
换句话说，最后的结果集是在outer\_tables中的，而semi-join的作用只是对outer\_tables中的记录进行筛选。这也是我们进行 semi-join优化的基础，即我们只需要从semi-join中获取到最少量的足以对outer\_tables记录进行筛选的信息就足够了。

select \* from Country

where

Country.Code in

(select City.country

from City

where City.Population>**1**\***1000**\***1000**);

**当中的semi-join：**

select City.country

from City

where City.Population>**1**\***1000**\***1000**

**ANTI JOIN**

这种连接的关系跟 SEMI JOIN 刚好相反。在 IN 或者 EXISTS 前加一个 NOT 关键字就能使用这种连接。

**CROSS JOIN**

这个连接过程就是两个连接的表的乘积：即将第一张表的每一条数据分别对应第二张表的每条数据。我们之前见过，这就是逗号在 FROM 语句中的用法。在实际的应用中，很少有地方能用到 CROSS JOIN，但是一旦用上了，你就可以用这样的 SQL语句表达：

SELECT \* FROM A CROSS JOIN B;== SELECT \* FROM A, B;

## 4.SQL 中如同变量的派生表

所谓的派生表就是在括号之中的子查询：FROM (SELECT \* FROM author)

有些时候我们可以给派生表定义一个相关名（即我们所说的别名）。

FROM (SELECT \* FROM author) a

注意：Every derived table must have its own alias

这句话的意思是说每个派生出来的表都必须有一个自己的别名

一般在多表查询时，会出现此错误。

因为，进行嵌套查询的时候子查询出来的的结果是作为一个派生表来进行上一级的查询的，所以子查询的结果必须要有一个别名

把MySQL语句改成：select count(\*) from (select \* from ……) as total;

问题就解决了，虽然只加了一个没有任何作用的别名total，但这个别名是必须的

## 5.SQL 语句中 GROUP BY 是对表的引用进行的操作

**GROUP BY，是在表的引用上进行了操作，将其转换为一种新的引用方式。**

**在MYSQL中使用GROUP BY对表中的数据进行分组时，**  
**GROUP BY X意思是将所有具有相同X字段值的记录放到一个分组里，**  
**GROUP BY X, Y意思是将所有具有相同X字段值和Y字段值的记录放到一个分组里。**

## 6. SQL 语句中的 SELECT 实质上是对关系的映射

通过 SELECT语句，你能对每一个字段进行操作，通过复杂的表达式生成所需要的数据。

SELECT 语句有很多特殊的规则，至少你应该熟悉以下几条：

你仅能够使用那些能通过表引用而得来的字段；

.如果你有 GROUP BY 语句，你只能够使用 GROUP BY 语句后面的字段或者聚合函数；

.当你的语句中没有 GROUP BY 的时候，可以使用开窗函数代替聚合函数；

.当你的语句中没有 GROUP BY 的时候，你不能同时使用聚合函数和其它函数；

.有一些方法可以将普通函数封装在聚合函数中；

其他语句的作用其实就是对表的不同形式的引用。而 SELECT 语句则把这些引用整合在了一起，通过逻辑规则将源表映射到目标表，而且这个过程是可逆的，我们可以清楚的知道目标表的数据是怎么来的。

**DISTINCT**：mysql提供有distinct这个关键字来过滤掉多余的重复记录只保留一条。

select distinct name, id from A实际上是根据“name+id”来去重，distinct同时作用在了name和id上

select id, distinct name from A; --会提示错误，因为distinct必须放在开头

① select \* from table limit 2,1;

//含义是跳过2条取出1条数据，limit后面是从第2条开始读，读取1条信息，即读取第3条数据

② select \* from table limit 2 offset 1;

//含义是从第1条（不包括）数据开始取出2条数据，limit后面跟的是2条数据，offset后面是从第1条开始读取，即读取第2,3条

ORDER BY 语句

ORDER BY 语句用于根据指定的列对结果集进行排序。

ORDER BY 语句默认按照升序对记录进行排序。

如果您希望按照降序对记录进行排序，可以使用 DESC 关键字。

以逆字母顺序显示公司名称，并以数字顺序显示顺序号：

SELECT Company, OrderNumber FROM Orders ORDER BY Company DESC, OrderNumber ASC

UNION 和 UNION ALL 操作符

UNION 操作符用于合并两个或多个 SELECT 语句的结果集。

请注意，UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。

注释：默认地，UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。

SQL 分为两个部分：数据操作语言 (DML) 和 数据定义语言 (DDL)。

SQL (结构化查询语言)是用于执行查询的语法。但是 SQL 语言也包含用于更新、插入和删除记录的语法。

查询和更新指令构成了 SQL 的 DML 部分：

SELECT - 从数据库表中获取数据

UPDATE - 更新数据库表中的数据

DELETE - 从数据库表中删除数据

INSERT INTO - 向数据库表中插入数据

SQL 的数据定义语言 (DDL) 部分使我们有能力创建或删除表格。我们也可以定义索引（键），规定表之间的链接，以及施加表间的约束。

SQL 中最重要的 DDL 语句:

CREATE DATABASE - 创建新数据库

ALTER DATABASE - 修改数据库

CREATE TABLE - 创建新表

ALTER TABLE - 变更（改变）数据库表

DROP TABLE - 删除表

CREATE INDEX - 创建索引（搜索键）

DROP INDEX - 删除索引

INSERT INTO table\_name (列1, 列2,...) VALUES (值1, 值2,....)

UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值

DELETE FROM 表名称 WHERE 列名称 = 值

SELECT TOP 2 \* FROM Persons --返回前两条数据

SELECT TOP 50 PERCENT \* FROM Persons –返回50%的数据

SELECT column\_name(s)列名

FROM table\_name表名

WHERE column\_name LIKE pattern（匹配的模式） eg：%hello%

|  |  |
| --- | --- |
| **通配符** | **描述** |
| % | 替代一个或多个字符 |
| \_ | 仅替代一个字符 |
| [charlist] | 字符列中的任何单一字符 |
| [^charlist]  或者  [!charlist] | 不在字符列中的任何单一字符 |

BETWEEN ... AND 会选取介于两个值之间的数据范围。这些值可以是数值、文本或者日期

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函数**

**1.** **AVG() 函数返回数值列的平均值。NULL 值不包括在计算中。**

**2.** **COUNT() 函数返回匹配指定条件的行数。**

**3.** **FIRST() 函数返回指定的字段中第一个记录的值。**

**4.** **LAST() 函数返回指定的字段中最后一个记录的值。**

**5.** **MAX() 函数返回一列中的最大值。NULL 值不包括在计算中。**

**6.** **MIN ()函数返回一列中的最小值。NULL 值不包括在计算中。**

**7.** **SUM ()函数返回数值列的总数（总额）。**

**8.** **在 SQL 中增加 HAVING 子句原因是，WHERE 关键字无法与合计函数一起使用。**

**9.** **UCASE ()函数把字段的值转换为大写。**

**10.** **LCASE ()函数把字段的值转换为小写。**

**11.** **LEN ()函数返回文本字段中值的长度。**

**12.** **ROUND 函数用于把数值字段舍入为指定的小数位数。**

SELECT ROUND(column\_name,decimals) FROM table\_name

|  |  |
| --- | --- |
| **参数** | **描述** |
| column\_name | 必需。要舍入的字段。 |
| decimals | 必需。规定要返回的小数位数。 |

**13.** **NOW ()函数返回当前的日期和时间。**

**14.** **FORMAT 函数用于对字段的显示进行格式化。**

**SELECT FORMAT(column\_name,format) FROM table\_name**

|  |  |
| --- | --- |
| **参数** | **描述** |
| column\_name | 必需。要格式化的字段。 |
| format | 必需。规定格式。 |