IS 441 SQL Handout 1 – Points to Watch in SQL Basics, Version 4

Dr. Yue “Jeff” Zhang, October 23, 2016

This handout does not attempt to systematically cover all the basic points of the SQL syntax and basic usage. Instead, it aims at providing clarification and deeper understanding of several keep concepts/skills that are important and that have appeared to be error-prone in the past.

=========================

\*\*\*\*PLEASE study all four materials:\*\*\*\*

1. This handout; 2. The password-protected handout “SQL\_Handout\_pswd”; 3. PPT slides; 4. Textbook.

=========================

The following are the main topics calling for special attention: 1. Column alias; 2. Caution on parentheses; 3. Sorting with ORDER BY; 4. GROUP BY, with WHERE or HAVING or both.

**1. Using Column Alias**

Column alias can be designated as the column header of the query output. Please note: The column alias is only for the display of the output; it cannot participate in operations in the SQL commands. Examples:

|  |  |
| --- | --- |
| **Right** | **Wrong** |
| SELECT product\_finish, avg(standard\_price) as [Average price]  FROM PRODUCT\_t  GROUP BY product\_finish  HAVING **avg(standard\_price)** > 250; | SELECT product\_finish, avg(standard\_price) as [Average price]  **Not work**  FROM PRODUCT\_t  GROUP BY product\_finish  HAVING **[Average price]** > 250; |
| **avg(standard\_price) is legitimate SQL function; can be used anywhere** | **Average price is only for display but not for Boolean operation** |

**2. Watch Your Parentheses**

The precedence of logical operation is NOT – AND – OR. If you must change that precedence, you need to use parentheses - **correctly**.

Use parentheses if you must break the “natural” precedence of NOT, AND, and OR;

Do NOT use parentheses if adding parentheses would change the logic that you intended.

Example:

|  |  |
| --- | --- |
| SELECT Major, LName, GPA  FROM STUD  WHERE  Major <> “Acct” AND GPA >=3.5 OR GPA <= 2.0 | 1. The non-accounting majors; 2. Non-ACCT majors whose GPA >= 3.5 3. Or: ANYONE (major doesn’t matter) GPA <= 2.0   The logic “flows” in the sequence as each part is mentioned, since they happen to agree w the precedence of NOT-AND-OR |

|  |  |
| --- | --- |
| SELECT Major, LName, GPA  FROM STUD  WHERE  Major <> “Acct” ***AND*** (GPA >=3.5 OR GPA <= 2.0) | 1. The non-accounting majors; 2. Non-ACCT majors **(**whose GPA >= 3.5 or whose GPA <= 2.0**)** – compare point “2” here with “2” and “3” above   Logical sequence altered by the parentheses. |
| SELECT Major, LName, GPA  FROM STUD  WHERE  Major <> “Acct” **OR** GPA >=2.0 **AND** GPA <= 3.5 | 1. The non-accounting majors; 2. ALL majors whose GPA between 2.0 and 3.5 (inclusive); 3. EITHER “1” **OR** “2” will be selected   The blue portion was processed before OR, since they are joined by **AND**. |

When you hesitate on the NOT/AND/OR and with or without parentheses, you may come back to visit these examples; hopefully they can help you to clarify your thinking and make the right judgment.

\*\* Same caution regarding parentheses applies to your mathematical expressions in SQL:

Use parentheses if you must break the “natural” precedence of - power, X or ÷, + or -;

Do NOT use parentheses if adding parentheses would change the logic that you intended.

**3. Sorting: ORDER BY and its use**

Note: Key word is ORDER! There’s not a word “sort”!

Sorting in SQL is accomplished by using

**ORDER** BY *field\_list*

When there are multiple fields that are ordered upon, the first in the list is the primary sort field, and the second the secondary sort field, and so on. SQL will first sort the resulted records by the primary sort field; then in the resulted list (already sorted on the primary sort field) records WITHIN the subgroup (whose members have the same value of the primary sort field) will be further sorted by the secondary sort field (again, that is WITHIN EACH GROUP in the result of the primary sort).

Example:

Members first sorted by Major would be then ***further*** sorted by GPA, \*IF\* they are of the same Major (within the same group of Major); then by name \*IF\* some of them have the same Major & GPA.

SELECT S\_LName, Major, GPA

FROM STUDENT

ORDER BY Major, GPA DESC, S\_LName;

The following is a likely outcome:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. Primary sort: by majors - Acct then BLAW then … Mgmt… SOM | | 2. Secondary sort: “GPA **DESC**” | 3. Tertiary sort: LName |  |
| within Acct:  4.0 🡪 3.9 🡪 3.8,  descending on GPA | **Major** | **GPA** | **S\_LName** |  |
| Acct | 4.0 | Smith | GPA 3.9: by S\_LName – B then C |
| Acct | 3.9 | Baker |
| Acct | 3.9 | Chen |
| Acct | 3.8 | Adams |
| Within Mgmt:  4.0 🡪 3.3 | Mgmt | 4.0 | Miller |  |
| Mgmt | 3.3 | Gonzalez | GPA 3.3: by S\_LName – G then O |
| Mgmt | 3.3 | Owens |
|  | **1st sort** | 2nd sort | 3rd sort |  |

**4. GROUP BY with HAVING, as compared to WHERE**

HAVING imposes conditions on groups, while

WHERE… GROUP BY…

GROUP BY… HAVING…

WHERE imposes conditions on individual records.

There are three cases that we need to distinct the two:

A. Both can work and produce the same final outputs;

B. Both can work BUT produce DIFFERENT final outputs;

C. Only HAVING works; WHERE does not work

Use the STUDENT table as an example.

Case A: Both can work and produce the same final outputs

|  |  |  |  |
| --- | --- | --- | --- |
| SELECT Major, AVG(GPA) FROM STUDENT | | | |
| WHERE Major = ‘Acct’  GROUP BY Major | | GROUP BY Major  HAVING Major = ‘Acct’ | |
| **Note: Condition on rows** | | **Note: Condition on groups** | |
| Output: The AVG(GPA) of 1,000+ Acct students, from our 7,000+ students; BUT with –  different processing: | | | |
| First find those whose Major is Acct (due to WHERE-clause) | Got 1,000+ records | First group all 7,000 students by their majors, | got 10 groups (assuming we have 10 majors in the College) |
| Then find their AVG(GPA), | Got 1 AVG(GPA) value | Then find the AVG(GPA) for all groups, | Got 10 AVG(GPA) values |
| Then choose from the 10 groups and 10 GPAs the group whose Major is “Acct”, | Got that 1 AVG(GPA) |
| Display the Major (“Acct”) and the AVG(GPA) value for that major | | | |

Case B: Both can work BUT will produce DIFFERENT final outputs

🡺🡺🡺 🡺🡺🡺

**Explanation next page**

|  |  |  |  |
| --- | --- | --- | --- |
| SELECT Major, AVG(GPA) FROM STUDENT | | | |
| WHERE GPA >= 3.5  GROUP BY Major | | GROUP BY Major  HAVING AVG(GPA) >= 3.5 | |
| Output: The AVG(GPA) from our 10 majors, BUT with different values: | | | |
| First find those students whose GPA is 3.5 or higher – the “high-performer subset” | Got, say, 200+ records - “high-performers” | First group all 7,000 students by their majors – the “population groups” (not “high-performer groups”) | got 10 groups (assuming we have 10 majors in the College) |
| Then group them | Got 10 groups – by Major | Then find the AVG(GPA) for all groups, | Got 10 AVG(GPA) values |
| Then find their AVG(GPA), | Got 10 (ten) AVG(GPA) values | Then choose from the 10 groups those groups whose AVG(GPA) is 3.5 or higher | Got several  (may be <10!!) AVG(GPA) values |
| 1. These AVGs are from those who are already 3.5 or higher – the “high-performer subset”;  2. All majors will appear, because we can reasonably assume that **every major** has students whose GPA is 3.5 or higher, who were selected in the WHERE clause to participate in the AVG stage. | | 1. These AVGs are from the “general population”, so the participating GPAs here can be from 0 to 4;  2. It is likely that not all majors would appear, since there is no guarantee that every major’s average GPA would be 3.5 or higher – here we display ONLY those majors (say Acct) whose AVG(GPA) is 3.5 or higher; | |
| 3. These 10 AVG(GPA)s are higher than those obtained from the right, because they are based on individual GPAs which are already AT LEAST 3.5; so after AVG those AVGs are certainly >3.5 – NOT just “=”, but surly “>”. | | 3. The AVG(GPA)s here are based on participating GPAs from 0 to 4 (compare: those on LEFT were 3.5 to 4). So the AVG(GPA)s here are lower than those on the left, with some possibly only at the threshold 3.5.  AND, very likely the AVG(GPA)s obtain would be <10. | |

Case C. Only HAVING works; WHERE does not work

|  |  |  |
| --- | --- | --- |
| SELECT Major, COUNT(SID) FROM STUDENT | | |
| WHERE does not work, because it only works on individual records but not groups, hence it can NOT handle group characters such as the COUNT of members in groups. | GROUP BY Major  HAVING COUNT(SID) >= 300 | |
| First group all 7,000 students by their majors, | got 10 groups (assuming we have 10 majors in the College) |
| Then COUNT the SID for all groups, | Got ten counts, say 1200, 180, 600, … |
| Then “filter” with the condition “count of the members in the group >=300” | Got, say, six majors that have at least 300 students |
| Display the results:   * The few (not all!!) majors that have at least 300 students (“GROUP BY Major HAVING COUNT(SID) >= 300” gives the number of students in a major), AND – * The number of students (“COUNT”) in each of those majors. * **Note: WHERE does NOT work with COUNT, SUM, AVG, etc.** | |