CL203: Database Systems lab

Lab#05

Contents

Aggregate Functions

SQL can perform various mathematical summaries for you, such as counting the number of rows that contain a specified condition, finding the minimum or maximum values for a specified attribute, summing the values in a specified column, and averaging the values in a specified column. Those aggregate functions are shown in the table listed below:

Function	**Output**
COUNT	The number of rows containing "non null" values.
MIN	The minimum attribute value encountered in a given column.
MAX	The maximum attribute value encountered in a given column.
SUM	The sum of all values for a given column.
AVG	The arithmetic mean (average) of the specified column.

Grouping Data

In the previous examples, the aggregate functions summarized data across all rows in the given tables. Sometimes, however, you do not want to treat the entire table as a single collection of data for summarizing. Rows can be grouped into smaller collections quickly and easily using the GROUP BY clause within the SELECT statement. The aggregate functions will then summarize the data within each smaller collection.

The syntax is:

```
SELECT columnlist

FROM tablelist

[WHERE conditionlist ]

[GROUP BY columnlist ]

[HAVING conditionlist ]

[ORDER BY columnlist [ASC | DESC] ] ;
```

The GROUP BY clause is generally used when you have attribute columns combined with aggregate functions in the SELECT statement.

SELECT COUNT(*),PARK_CODE FROM EMPLOYEE GROUP BY PARK_CODE;

The GROUP BY Feature's HAVING Clause

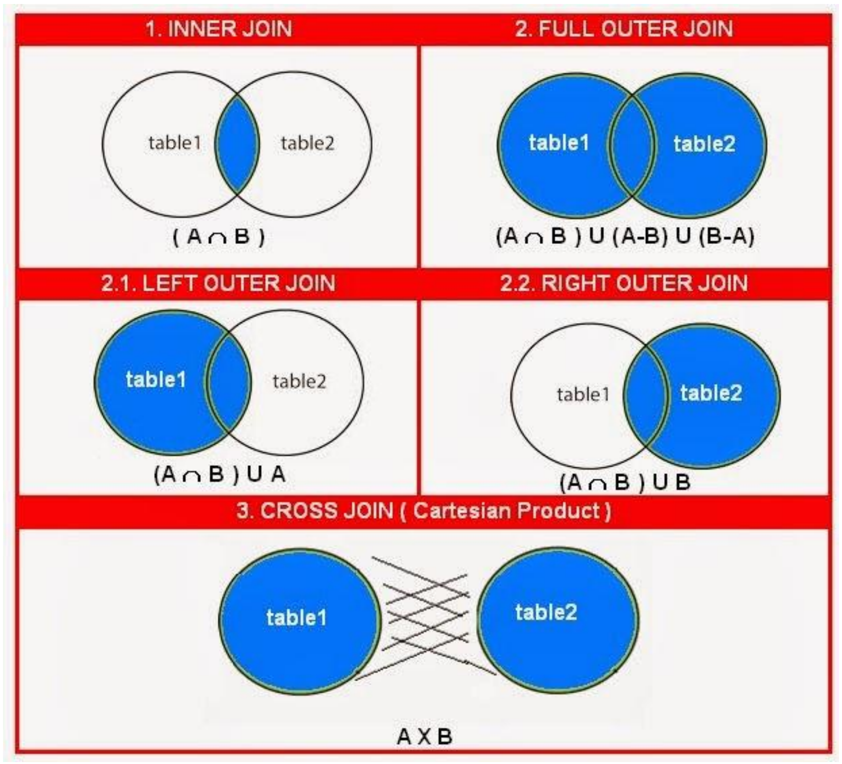
A particularly useful extension of the GROUP BY feature is the HAVING clause. The HAVING clause operates very much like the WHERE clause in the SELECT statement. However, the WHERE clause applies to columns and expressions for individual rows, while the HAVING clause is applied to the output of a GROUP BY operation.

```
SELECT AVG(TICKET_PRICE), PARK_CODE FROM TICKET GROUP BY PARK_CODE HAVING
AVG(TICKET_PRICE) < 20;
```

Joins:

The relational join operation merges rows from two tables and returns the rows with one of the following conditions:

- Have common values in common columns (natural join).
- Meet a given join condition (equality or in-equality).
- Have common values in common columns or have no matching values (outer join).



Cross Join

A cross join performs a relational product (also known as the Cartesian product) of two tables. The cross join syntax is:

SELECT *column-list* FROM *table1* CROSS JOIN *table2*

INNER JOINS

Natural Join

Natural join returns all rows with matching values in the matching columns and eliminates duplicate columns. This style of query is used when the tables share one or more common attributes with common names.

The natural join syntax is:

SELECT *column-list* FROM *table1* NATURAL JOIN *table2*

The natural join will perform the following tasks:

- Determine the common attribute(s) by looking for attributes with identical names and compatible data types.
- Select only the rows with common values in the common attribute.
- If there are no common attributes, return the relational product of the two tables.

JOIN USING CLAUSE

A second way to express a join is through the USING keyword. The query returns only the rows with matching values in the column indicated in the USING clause—and that column must exist in both tables. The syntax is:

SELECT *column-list* FROM *table1* JOIN *table2* USING (*common-column*)

OUTER JOINS

An outer join returns not only the rows matching the join condition (that is, rows with matching values in the common columns), it returns the rows with unmatched values. The ANSI standard defines three types of outer joins: left, right, and full. The left and right designations reflect the order in which the tables are processed by the DBMS. Remember that join operations take place two tables at a time. The first table named in the FROM clause will be the left side, and the second table named will be the right side. If three or more tables are being joined, the result of joining the first two tables becomes the left side, and the third table becomes the right side.

Left Outer Join

The left outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column); it returns the rows in the left table with unmatched values in the right table. The syntax is:

SELECT *column-list* FROM *table1* LEFT [OUTER] JOIN *table2* ON *join-condition*

Right Outer Join

The right outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column), it returns the rows in the right table with unmatched values in the left table. The syntax is:

SELECT *column-list* FROM *table1* RIGHT [OUTER] JOIN *table2* ON *join-condition*

Full Outer Join

The full outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column), it returns all of the rows with unmatched values in the table on either side. The syntax is:

SELECT *column-list* FROM *table1* FULL [OUTER] JOIN *table2* ON *join-condition*

Reference;

https://www.mysqltutorial.org/mysql-functions.aspx

https://stackoverflow.com/questions/17946221/sql-join-and-different-types-of-joins

http://www.sql-join.com/sql-join-types

https://dev.mysql.com/doc/refman/5.6/en/functions.html