Mater Dei College
Tubigon, Bohol
2nd Semester, A.Y. 2022-2023
CC 202 – IM 101 (INFORMATION MANAGEMENT 1)

Topic 12
Study Guide
SQL QUERY STATEMENT (Retrieving Data from a Table)

RELATIONAL OPERATION

A relational operation is an operation that extracts the desired data from a table. There are three basic relational operations.
Project---- extract the specified item from the table
Select-----extract the specified record from the table
Join-----extract data that combines two or more tables by means of a certain item with the same value

PROJECTION AND SELECTION

Below is an example showing the result of extracting data by projection and selection using a SELECT statement on the table EMPLOYEE_TBL. We can also combine selection and projection.

EMPLOYEE_TBL

Name	Department	Home country	Age
Jimmy	Sales	Japan	28
Frank	Human Resources	USA	22
Billy	Sales	France	35
Harry	Sales	Italy	43
Josh	General Affairs	Germany	48
Randy	Human Resources	USA	36
Steve	sales	UK	31

Select name, department from employee_tbl;

Projection (extracting names and departments)

Name	Department
Jimmy	Sales
Frank	Human Resources
Billy	Sales
Harry	Sales
Josh	General Affairs
Randy	Human Resources
Steve	sales

Selection & Projection

Name	Department
Billy	Sales
Harry	Sales
Josh	General affairs
Randy	Human resources

Select name, department
From employee_tbl
Where age>= 35;

select * from employee_tbl where age >= 35;
Selection(extracting rows where the age is 35 or above

Name	Department	Home country	Age
Billy	Sales	France	35
Harry	Sales	Italy	43
Josh	General affairs	Germany	48
Randy	Human resources	USA	36

CLAUSES IN SQL STATEMENT:

Select –retrieves rows, columns, and derived values from one more tables.
From – one of the two most important clauses in SQL statement. This refers to the table where data is to be extracted or retrieved.
Where – filtering unwanted rows from the result.
Order by – sort rows in ascending or descending order.
Group by – consolidates all rows that have the same value in a certain column.

Creating Column Aliases with AS

Column alias is an alternative name (identifier) that you specify to control how column headings are displayed in a result. Use column aliases if column names are cryptic, hard to type, too long or too short.

- Must be specified in select clause of a select statement
- Follow a column name with the AS keyword and an alias enclosed in single or double quotes with the ff. variations
  - Omit the quotes if the alias is a single word that contains only letters, digits, or underscore.
  - Use quotes if the alias contains spaces, punctuation or special characters

Ex. Select Cust_id as “customer id”, Lname as Lastname, address from customer;

Eliminating duplicate rows with DISTINCT

Columns often contain duplicate values, and it’s common to want a result tjat list each duplicate only once.

Ex. Select DISTINCT address from customer;

Sorting rows with ORDER BY

Rows in query result are unordered. Use the order by clause to sort rows by a specified columns or columns in ascending (asc) or descending (desc).

Ex. Select * from customer order by age asc;

Filtering rows with WHERE

Use the WHERE clause to filter unwanted rows from the result. WHERE clause- specifies a search condition that comprises one or more conditions that need to be satisfied by the rows of a table. A **condition**, or predicate, is a logical expression that evaluates to true, false or unknown. The unknown result arises from nulls. Rows for which the condition is true are included in the result; rows for which the condition is false or unknown are excluded.

SQL provides operators that express different types of conditions. Operators are symbols or keywords that specify actions to be performed in values or other elements. You can combine and negate conditions by using the logical operators AND, OR,NOT

Types of Condition and Their corresponding operators

Comparison – with the following operators =, <>, <, <=, >, >=
Pattern Matching – LIKE
Range Filtering – BETWEEN
List Filtering – IN
Null Test – IS NULL

Comparison
EXAMPLE

1. select * from employee_tbl where age >= 35;
2. Select name, department From employee_tbl Where age>= 35;

Pattern Matching

Use LIKE to retrieve rows based on partial information. Like is useful if you don’t know an exact value. Like works with only characters strings, it uses pattern that values are matched against. A **pattern** is a quoted string that contains the literals characters to match and any combination of wildcards. **Wildcards** are special characters used to match parts of a value. You can negate like condition with not like, can combine like conditions and other conditions with AND and OR.

Wildcards operators
% a percent sign matches any string of zero or more characters
_ an underscore matches any one character.

Ex. Select fname, lname from customer where fname like ‘%Ke’;
Select fname, lname from customer where fname not like ‘Ke%’;

Examples of % and _ patterns

PATTERN	MATCHES
‘a%’	-matches a string value of length >= 1 that begins with ***a,*** including single letter A.
‘%s’	- matches a string value of length >= 1 that ends with ***s,*** including single letter s.
‘%in%	-matches a string value of length >= 1 that contains ***in*** anywhere.
‘____’	-matches any four characters string value.
‘Qua__’	-matches any five characters string value that begins with ***Qua.***
‘_re_’	-matches any four characters string value that has ***re*** as its second and third characters.

'_re%' -matches a string values of length >= 3 that begins with any character and has ***re*** as its second and third characters.

'%re_' - matches a string value of length >=3 that has ***re*** as the second and third characters from its end, and ends with any character.

Range Filtering with BETWEEN

Use between to determine whether a given value falls within a specified range.

Important characteristics of between conditions are:

- Between works with character strings, numbers, and datetimes.
- The between range contains a low value and a high value, separated by AND. The low value must be less than of equal to the high value.
- Between is a convenient shorthand clause, you can replicate its behavior by using AND;
    - Where test_column BETWEEN low_value and high_value
      Is equivalent to
      Where (test_column>= low_value) and (test_column<= high_value)
- Between specifies an inclusive range, in which the high value and low value are included in the result/search. To specify an exclusisve range, which excludes endpoints, use > and < comparisons operator instead of between operator.
- You can negate a between conditions with not between operator.
- You can combine BETWEEN conditions and other conditions with AND and OR.

List Filtering with IN

Use IN to determine whether a given value matches any value in a specified list.

- IN works with character strings, numbers, and datetimes
- The IN list is a parenthesized listing of one or more comma-separated values.
- IN is a convenient, shorthand clause. You can replicate its behavior by using OR:
  Where test_columnIN(value1, value2, value3)
  Is equivalent to
  Where (test_column = value1) OR (test_column = value2) OR (test_column = value3)
- You can negate an IN condition with NOT IN.
- You can combine IN conditions and other conditions with AND and OR.

Testing for Nulls with IS NULL

Null represent missing or unknown values. This situation causes a problem: LIKE, BETWEEN, IN, and other WHERE clause options can't find nulls because unknown values don't satisfy specific conditions. Nulls match no value. IS NULL determines whether a given value is null.

- Works for columns of any data type
- You can negate an IS NULL condition with IS NOT NULL
- You can combine IS NULL conditions and other conditions with AND and OR.
  EX. SELECT columns from table where test_column IS [NOT] NULL;

OPERATORS AND FUNCTIONS

Operators and functions permit you to calculate results derived from column data, system-determined values, and constant expressions. Where in you can perform the following:

- Arithmetic operators like cut everyone's salary by 10 percent
- String operators like concatenate personal information into a mailing address
- Datetime operations – find out what time your DBMS thinks it is.

Operator is a symbol or keyword indicating an operation that acts on one or more elements. The elements, called operands, are SQL expressions which is any legal combination of symbols and tokens that evaluates to a single value (or null). For example price * 2, * is the operator, and price and 2 is the operands.

Function is a built-in, named routine that performs a specialized task. Most functions take parenthesized arguments, which are values you pass to the function then uses to perform its task. Arguments can be column names, constants, nested functions, or more complex expressions. For example UPPER (last_name). Upper is the function name, last_name is the argument.

Creating derived columns

You can use operators and functions to create columns. A **derived column** is the result of a calculation and is created with the select clause expression that is something other than a simple reference to a column. Derived columns don't become a permanent columns in a table; they're for display purposes only. The values in derived colun often are computed from values in existing columns

Ex.

Select au_id, salary*2 from authors;
Select brand, description, price, price + (0.5 * Price);
Select brand, description, price as "old price", price * 1.5;
Select brand, description, price as "old price", price * 0.5 as increase, price* 1.5 as "new price";
Select brand, description, price as "old price", price * 0.5 as discount, price – (.5* price) as "new price";

Determining the order of evaluation

Precedence determines the priority of various operators when more than one operator is used in an expression. Operators with higher precedence are evaluated first. Arithmetic operators (+,-,*,/) have higher precedence than comparison operators(<,>,<>,<+,>=,=), which have higher precedence than logical operators(NOT,AND,OR)

Concatenating string with CONCAT()

Use the concat operator to combine, or concatenate strings.

Ex.

- CONCAT('formal','ware')
- CONCAT('formal','','ware')
- SELECT CONCAT(last_name,', ',first_name) AS full_name
  -> FROM table_name ORDER BY full_name;
- SELECT CONCAT(first_name,' ',lastname,'is',age,'years old') as remark from table_name;

Changing String Case with UPPER() and LOWER()

USE THE UPPER() function to return a string with lowercase letters converted to upper case, and use the LOWER() function to return a string with uppercase letters converted to lowercase.

Ex. Select upper(Lname) as upper, lower(Fname) as lower from table_name;

Finding the length of a string with CHARACTER_LENGTH ()

Use the character_length function to return the number of characters in a string.

- Character_length() returns an integer greater than or equal to zero.
- Counts characters not bytes
- The length of empty string is zero
- Can use in select, where, and order by caluses.
- Character_length() and char_length are synonyms
  Ex. Select character_length(fname) from tb_name;
  Select fname, char_length(fname) from tb_name;
  Select fname, char_length(fname) as 'firstname length' from tb_name;

Evaluating Conditional Values with CASE

Case Characteristics are:

- You'll recognize that CASE provides SQL the equivalent of the if-the-else, case, or switch statements used in procedural languages except that CASE is an expression, not a statement.
- CASE is used to evaluate several conditions and return a single value for the first true condition
- CASE allows you to display an alternative value to the actual value in a column. CASE makes no changes to the underlying data.
- A common use of CASE is to replace codes or abbreviations with more-readable values. If the column marital_status contains the integer codes 1,2,3,4 meaning single, married, divorced, or widowed-readers will prefer to see explanatory text rather than cryptic codes. (Database designer prefer to use codes, because it's more efficient to store and manage abbreviated codes than explanatory text).

- CASE has two formats: simple and searched. The simple case expression compares an expression to a set of simple expressions to determine the result. The searched case expression evaluates a set of logical (Boolean) expressions to determine the result
- CASE returns an optional ELSE result as the default value if no test condition is true
- Ex. **Simple case expression**

```
Select title_id, type, price,
        Case type
        When 'history' then price * 1.10
        When 'psychology' then price * 1.20
        Else price
        End as "new price" from book;
```

Ex: **Searched CASE expression**

```
Select title_id, case when sales is null then "unknown'
        When sales <= 1000 then "sale is low"
        When sales <= 50000 then "sale is high"
        Else 'sale is high' end as 'sales category'
        From book
        Order by sales asc;
```

SUMMARIZING AND GROUPING DATA

The preceding chapter described scalar functions, which operate on individual row values. This topic introduces SQL's aggregate functions, or set functions, which operate on a group of values to produce a single, summarizing value. You apply an aggregate to a set of rows, which may be:

- All the rows in a table
- Only those rows specified by a where clause
- those rows created by a group by clause

No matter how many rows the set contains, an aggregate function returns a single statistic: a sum, minimum, or average.

USING AGGREGATE FUNCTIONS

Function	Returns
MIN(expr)	Minimum value in expr
MAX(expr)	Maximum value in expr
SUM(expr)	Sum of the values in expr
AVG(expr)	Average(arithmetic mean) of the values in expr
COUNT(expr)	The number of non-null values in expr
COUNT(*)	The number of rows in a table or set

The characteristics of the aggregate functions are:

- Expression often is a column name.
- SUM () and AVG() work with only numeric data type. MIN() AND MAX() work with characters, numeric, and date time data types. COUNT(expr) and COUNT(*) work with all data types.
- All aggregate functions except COUNT(*) ignore nulls
- Often use with the group by clause
- Use WHERE clause to restrict the rows used in aggregate calculations
- Default column headings for aggregate vary by DBMS use: use column alias to name the result column

```
Ex. Select sum(sales), min(price), max(price) from item group by description;
        Select count(*) as 'total number of items' from item;
    Select count(description) as 'total' from item group by description order by description asc;
```