Lab 1. Retrieving Records

This lab focuses on very basic SELECT statements. It is important to have a solid understanding of the basics as many of the topics covered here are not only present in more difficult recipes but also are found in everyday SQL.

1.1. Retrieving All Rows and Columns from a Table

PROBLEM

You have a table and want to see all of the data in it.

SOLUTION

Use the special **"*"** character and issue a SELECT against the table:

```
select * from emp
```

DISCUSSION

The character **"*"** has special meaning in SQL. Using it will return every column for the table specified. Since there is no WHERE clause specified, every row will be returned as well. The alternative would be to list each column individually:

```
select empno,ename,job,sal,mgr,hiredate,comm,deptno from emp
```

In ad hoc queries that you execute interactively, it's easier to use **SELECT ***. However, when writing program code it's better to specify each column individually. The performance will be the same, but by being explicit you will always know what columns you are returning from the query. Likewise, such queries are easier to understand by people other than yourself (who may or may not know all the columns in the tables in the query).

1.2. Retrieving a Subset of Rows from a Table

PROBLEM

You have a table and want to see only rows that satisfy a specific condition.

SOLUTION

Use the **WHERE** clause to specify which rows to keep. For example, to view all employees assigned to department number 10:

```
select * from emp where deptno = 10
```

DISCUSSION

The WHERE clause allows you to retrieve only rows you are interested in. If the expression in the WHERE clause is true for any row, then that row is returned.
Most vendors support common operators such as: =, <, >, <=, >=, !, <>. Additionally, you may want rows that satisfy multiple conditions; this can be done by specifying AND, OR, and parenthesis, as shown in the next recipe.

1.3. Finding Rows That Satisfy Multiple Conditions

PROBLEM

You want to return rows that satisfy multiple conditions.

SOLUTION

Use the WHERE clause along with the OR and AND clauses. For example, if you would like to find all the employees in department 10, along with any employees who earn a commission, along with any employees in department 20 who earn at most $2000:

```
select *       from emp        where deptno = 10       or comm is not null     or sal <= 2000 and deptno=20
```

DISCUSSION

You can use a combination of AND, OR, and parenthesis to return rows that satisfy multiple conditions. In the solution example, the WHERE clause finds rows such that:

- the DEPTNO is 10, or
- the COMM is not NULL, or
- the salary is $2000 or less for any employee in DEPTNO 20.

The presence of parentheses causes conditions within them to be evaluated together.
For example, consider how the result set changes if the query was written with the parentheses as shown below:

```
select *        from emp       where ( deptno = 10      or comm is not null     or sal <= 2000          )       and deptno=20
```

```
 EMPNO ENAME  JOB     MGR  HIREDATE      SAL       COMM  DEPTNO
 ----- ------ ----- -----  ----------- ----- ----------  ------
  7369 SMITH  CLERK  7902  17-DEC-1980   800                 20
  7876 ADAMS  CLERK  7788  12-JAN-1983  1100                 20
```

1.4. Retrieving a Subset of Columns from a Table

PROBLEM

You have a table and want to see values for specific columns rather than for all the columns.

SOLUTION

Specify the columns you are interested in. For example, to see only name, department number, and salary for employees:

```
select ename,deptno,sal from emp
```

DISCUSSION

By specifying the columns in the SELECT clause, you ensure that no extraneous data is returned. This can be especially important when retrieving data across a network, as it avoids the waste of time inherent in retrieving data that you do not need.

1.5. Providing Meaningful Names for Columns

PROBLEM

You would like to change the names of the columns that are returned by your query so they are more readable and understandable. Consider this query that returns the salaries and commissions for each employee:

```
select sal,comm from emp
```

What's "sal"? Is it short for "sale"? Is it someone's name? What's "comm"? Is it communication? You want the results to have more meaningful labels.

SOLUTION

To change the names of your query results use the AS keyword in the form: original_name AS new_name. Some databases do not require AS, but all accept it:

```
select sal as salary, comm as commission from emp
```

```
 SALARY   COMMISSION
 -------  ----------
     800
    1600         300
    1250         500
    2975
    1250        1400
    2850
    2450
    3000
    5000
    1500           0
    1100
     950
    3000
    1300
```

DISCUSSION

Using the AS keyword to give new names to columns returned by your query is known as aliasing those columns. The new names that you give are known as aliases. Creating good aliases can go a long way toward making a query and its results understandable to others.

1.6. Referencing an Aliased Column in the WHERE Clause

PROBLEM

You have used aliases to provide more meaningful column names for your result set and would like to exclude some of the rows using the WHERE clause. However, your attempt to reference alias names in the WHERE clause fails:

```
select sal as salary, comm as commission        from emp        where salary < 5000
```

SOLUTION

By wrapping your query as an inline view you can reference the aliased columns:

```
select *       from (  select sal as salary, comm as commission        from emp        ) x     where salary < 5000
```

DISCUSSION

In this simple example, you can avoid the inline view and reference COMM or SAL directly in the WHERE clause to achieve the same result. This solution introduces you to what you would need to do when attempting to reference any of the following in a WHERE clause:

- Aggregate functions
- Scalar subqueries
- Windowing functions
- Aliases
  Placing your query, the one giving aliases, in an inline view gives you the ability to reference the aliased columns in your outer query. Why do you need to do this? The WHERE clause is evaluated before the SELECT, thus, SALARY and COMMISSION do not yet exist when the “Problem” query’s WHERE clause is evaluated. Those aliases are not applied until after the WHERE clause processing is complete. However, the FROM clause is evaluated before the WHERE. By placing the original query in a FROM clause, the results from that query are generated before the outermost WHERE clause, and your outermost WHERE clause “sees” the alias names. This technique is particularly useful when the columns in a table are not named particularly well.
  TIP
  The inline view in this solution is aliased X. Not all databases require an inline view to be explicitly aliased, but some do. All of them accept it.

1.7. Concatenating Column Values

PROBLEM

You want to return values in multiple columns as one column. For example, you would like to produce this result set from a query against the EMP table:
CLARK WORKS AS A MANAGER
KING WORKS AS A PRESIDENT
MILLER WORKS AS A CLERK
However, the data that you need to generate this result set comes from two different columns, the ENAME and JOB columns in the EMP table:

```
select ename, job       from emp        where deptno = 10


  ENAME      JOB
  ---------- ---------
  CLARK      MANAGER
  KING       PRESIDENT
  MILLER     CLERK
```

SOLUTION

Find and use the built-in function provided by your DBMS to concatenate values from multiple columns.
DB2, Oracle, PostgreSQL
These databases use the double vertical bar as the concatenation operator:

```
select ename||' WORKS AS A '||job as msg from emp where deptno=10
```

MySQL
This database supports a function called CONCAT:

```
select concat(ename, ' WORKS AS A ',job) as msg        from emp        where deptno=10
```

SQL Server

Use the "+” operator for concatenation:

```
select ename + ' WORKS AS A ' + job as msg      from emp        where deptno=10
```

DISCUSSION

Use the CONCAT function to concatenate values from multiple columns. The || is a shortcut for the CONCAT function in DB2, Oracle, and PostgreSQL, while + is the shortcut for SQL Server.

1.8. Using Conditional Logic in a SELECT Statement

PROBLEM

You want to perform IF-ELSE operations on values in your SELECT statement. For example, you would like to produce a result set such that, if an employee is paid $2000orless, amessageof“UNDERPAID”isreturned, ifanemployeeispaid$4000 or more, a message of “OVERPAID” is returned, if they make somewhere in between, then “OK” is returned. The result set should look like this:

```
ENAME SAL STATUS
---------- ---------- ---------
SMITH 800 UNDERPAID
ALLEN 1600 UNDERPAID
WARD 1250 UNDERPAID
JONES 2975 OK
MARTIN 1250 UNDERPAID
BLAKE 2850 OK
CLARK 2450 OK
SCOTT 3000 OK
KING 5000 OVERPAID
TURNER 1500 UNDERPAID
ADAMS 1100 UNDERPAID
JAMES 950 UNDERPAID
FORD 3000 OK
MILLER 1300 UNDERPAID
```

SOLUTION

Use the CASE expression to perform conditional logic directly in your SELECT statement:

```
select ename,sal,      case when sal <= 2000 then 'UNDERPAID'  when sal >= 4000 then 'OVERPAID'        else 'OK'       end as status
```

DISCUSSION

The CASE expression allows you to perform condition logic on values returned by a query. You can provide an alias for a CASE expression to return a more readable result set. In the solution, you’ll see the alias STATUS given to the result of the CASE expression. The ELSE clause is optional. Omit the ELSE, and the CASE expression will return NULL for any row that does not satisfy the test condition.

1.9. Limiting the Number of Rows Returned

PROBLEM

You want to limit the number of rows returned in your query. You are not concerned with order; any n rows will do.

SOLUTION

Use the built-in function provided by your database to control the number of rows returned.

DB2

In DB2 use the FETCH FIRST clause:

```
select *       from emp fetch first 5 rows only
```

MySQL and PostgreSQL

Do the same thing in MySQL and PostgreSQL using LIMIT:

```
select * from emp limit 5
```

Oracle

In Oracle, place a restriction on the number of rows returned by restricting ROWNUM in the WHERE clause:

```
select *       from emp        where rownum <= 5
```

SQL Server

Use the TOP keyword to restrict the number of rows returned:

```
select top 5 * from emp
```

DISCUSSION

Many vendors provide clauses such as FETCH FIRST and LIMIT that let you specify the number of rows to be returned from a query. Oracle is different, in that you must make use of a function called ROWNUM that returns a number for each row returned (an increasing value starting from 1).
Here is what happens when you use ROWNUM <= 5 to return the first five rows:

1. Oracle executes your query.
2. Oracle fetches the first row and calls it row number 1.
3. Have we gotten past row number 5 yet? If no, then Oracle returns the row, because it meets the criteria of being numbered less than or equal to 5. If yes, then Oracle does not return the row.
4. Oracle fetches the next row and advances the row number (to 2, and then to 3, and then to 4, and so forth).
5. Go to step 3.
   As this process shows, values from Oracle's ROWNUM are assigned after each row is fetched. This is a very important and key point. Many Oracle developers attempt to return only, say, the fifth row returned by a query by specifying ROWNUM = 5.
   Using an equality condition in conjunction with ROWNUM is a bad idea. Here is what happens when you try to return, say, the fifth row using ROWNUM = 5:
6. Oracle executes your query.
7. Oracle fetches the first row and calls it row number 1.
8. Have we gotten to row number 5 yet? If no, then Oracle discards the row, because it doesn't meet the criteria. If yes, then Oracle returns the row. But the answer will never be yes!
9. Oracle fetches the next row and calls it row number 1. This is because the first row to be returned from the query must be numbered as 1.
10. Go to step 3.
    Study this process closely, and you can see why the use of ROWNUM = 5 to return the fifth row fails. You can't have a fifth row if you don't first return rows one through four!
    You may notice that ROWNUM = 1 does, in fact, work to return the first row, which may seem to contradict the explanation thus far. The reason ROWNUM = 1 works to return the first row is that, to determine whether or not there are any rows in the table, Oracle has to attempt to fetch at least once. Read the preceding process carefully, substituting 1 for 5, and you'll understand why it's OK to specify ROWNUM = 1 as a condition (for returning one row).

1.10. Returning n Random Records from a Table

PROBLEM

You want to return a specific number of random records from a table. You want to modify the following statement such that successive executions will produce a different set of five rows:

```
select ename, job       from emp
```

SOLUTION

Take any built-in function supported by your DBMS for returning random values. Use that function in an ORDER BY clause to sort rows randomly. Then, use the previous recipe's technique to limit the number of randomly sorted rows to return.

DB2

Use the built-in function RAND in conjunction with ORDER BY and FETCH:

```
select ename,job       from emp        order by rand() fetch first 5 rows only
```

MySQL

Use the built-in RAND function in conjunction with LIMIT and ORDER BY:

```
select ename,job       from emp        order by rand() limit 5
```

PostgreSQL

Use the built-in RANDOM function in conjunction with LIMIT and ORDER BY:

```
select ename,job       from emp        order by random() limit 5
```

Oracle

Use the built-in function VALUE, found in the built-in package DBMS_RANDOM, in conjunction with ORDER BY and the built-in function ROWNUM:

```
select *       from (  select ename, job       from emp        order by dbms_random.value()    )       where rownum <= 5
```

SQL Server

Use the built-in function NEWID in conjunction with TOP and ORDER BY to return a random result set:

```
select top 5 ename,job         from emp        order by newid()
```

DISCUSSION

The ORDER BY clause can accept a function's return value and use it to change the order of the result set. The solution queries all restrict the number of rows to return after the function in the ORDER BY clause is executed. Non-Oracle users may find it helpful to look at the Oracle solution as it shows (conceptually) what is happening under the covers of the other solutions.
It is important that you don't confuse using a function in the ORDER BY clause with using a numeric constant. When specifying a numeric constant in the ORDER BY clause, you are requesting that the sort be done according the column in that ordinal position in the SELECT list. When you specify a function in the ORDER BY clause, the sort is performed on the result from the function as it is evaluated for each row.

1.11. Finding Null Values

PROBLEM

You want to find all rows that are null for a particular column.

SOLUTION

To determine whether a value is null, you must use IS NULL:

```
select *       from emp        where comm is null
```

DISCUSSION

NULL is never equal/not equal to anything, not even itself, therefore you cannot use = or != for testing whether a column is NULL. To determine whether or not a row has NULL values you must use IS NULL. You can also use IS NOT NULL to find rows without a null in a given column.

1.12. Transforming Nulls into Real Values

PROBLEM

You have rows that contain nulls and would like to return non-null values in place of those nulls.

SOLUTION

Use the function COALESCE to substitute real values for nulls:

```
select coalesce(comm,0)        from emp
```

DISCUSSION

The COALESCE function takes one or more values as arguments. The function returns the first non-null value in the list. In the solution, the value of COMM is returned whenever COMM is not null. Otherwise, a zero is returned.
When working with nulls, it's best to take advantage of the built-in functionality provided by your DBMS; in many cases you'll find several functions work equally as well for this task. COALESCE happens to work for all DBMSs. Additionally, CASE can be used for all DBMSs as well:

```
select case     when comm is not null then comm         else 0          end     from emp
```

While you can use CASE to translate nulls into values, you can see that it's much easier and more succinct to use COALESCE.

1.13. Searching for Patterns

PROBLEM

You want to return rows that match a particular substring or pattern. Consider the following query and result set:

```
select ename, job       from emp        where deptno in (10,20)
```

```
ENAME       JOB
----------  ---------
SMITH       CLERK
JONES       MANAGER
CLARK       MANAGER
SCOTT       ANALYST
KING        PRESIDENT
ADAMS       CLERK
FORD        ANALYST
MILLER      CLERK
```

Of the employees in departments 10 and 20, you want to return only those that have either an “I” somewhere in their name or a job title ending with “ER”:

```
ENAME JOB
---------- ---------
SMITH CLERK
JONES MANAGER
CLARK MANAGER
KING PRESIDENT
MILLER CLERK
```

SOLUTION

Use the LIKE operator in conjunction with the SQL wildcard operator (”%”):

```
select ename, job      from emp        where deptno in (10,20)         and (ename like '%I%' or job like '%ER')
```

DISCUSSION

When used in a LIKE pattern-match operation, the percent (“%”) operator matches any sequence of characters. Most SQL implementations also provide the underscore (“_”) operator to match a single character. By enclosing the search pattern “I” with “%” operators, any string that contains an “I” (at any position) will be returned. If you do not enclose the search pattern with “%”, then where you place the operator will affect the results of the query. For example, to find job titles that end in “ER”, prefix the “%” operator to “ER”; if the requirement is to search for all job titles beginning with “ER”, then append the “%” operator to “ER”.