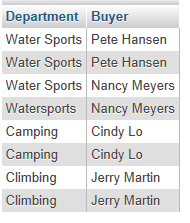
**STRUCTURED QUERY LANGUAGE**

**These examples are based on the database.sql file on the Canvas in the Chapter 2 module.**

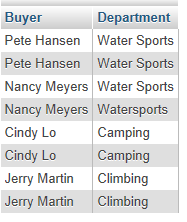
**Reading specified columns from a single table:** We begin very simply. Suppose we want to obtain the values of the Department and Buyer columns of the SKU\_Data table. A SQL statement to read that data is the following:

**Select Department, Buyer from sku\_data;**

****

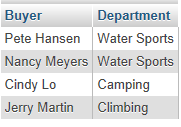
The order of the column names in the select phrase determines the order of the columns in the result.

**Select Buyer, Department from sku\_data;**



Notice that some rows are duplicated in these results. The data in the first and second row, for example, is identical. We can eliminate duplicates by using the DISTINCT keyword as follows:

**Select DISTINCT Buyer, Department from sku\_data;**



Suppose that we want to view all the columns of the SKU\_DATA table. To do so, we can name each column in the SELECT statement as follows:

**Select SKU, SKU\_Description, Department, Buyer from sku\_data;**

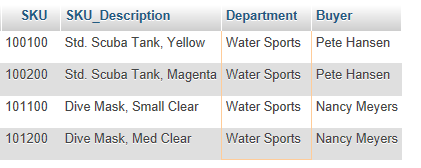
However, SQL provides a shorthand notation for querying all of the columns of a table. The shorthand is to use an asterisk:

**Select \* From sku\_data;**

****

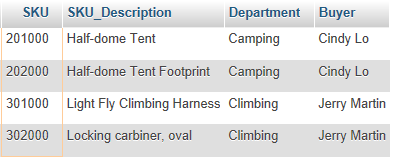
**Reading Specified Rows from a Single Table:** Suppose we want all of the columns of the SKU\_Data table, but we want only the rows for the Water Sports department. We can obtain that result by using the WHERE clause as follows:

**Select \* from sku\_data where Department = “Water Sports”;**

****

In a WHERE clause, if the column contains text or date data, the comparison values must be enclosed in quotation marks. If the column contains numeric data, however, the comparison values need not be in quotes. Thus, to find all of the SKU\_rows with a value greater than 200,000, we would code:

**Select \* From sku\_data Where SKU>200000;**

****

**Reading Specified Columns and Specified Rows from a Single Table:** So far, we have selected certain columns and all rows and we have selected all columns and certain rows. We can combine these operations to select certain columns and certain rows by naming the columns we want and using the WHERE clause. For example, to obtain the SKU\_Description and Department of all products in the climbing department, we specify:

**Select SKU\_Description, Department from sku\_data Where Department = “Climbing”;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

**Sorting the Results:** The order of the rows produced by a SQL statement is arbitrary. If you want the DBMS to display the rows in a particular order, you can use the ORDER BY phrase.

**Select \* from order\_item order by OrderNumber;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

We can sort by two columns by adding a second column name. For example, to sort first by OrderNumber and then By Price within OrderNumber, code the following:

**Select \* from order\_item order by OrderNumber, Price;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

By default, rows are sorted in ascending order. To sort in descending order, add the keyword DESC after the column name. Thus, to sort first by Price in descending order and then by OrderNumber in ascending order, we can specify:

**Select \* from order\_item Order By Price DESC, OrderNumber ASC;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

**WHERE Clause Options**

SQL includes a number of WHERE clause options that expand SQL’s power and utility. In this section, we consider three options: compound clauses, ranges, and wildcards.

**Compound Where Clauses:** SQL WHERE clauses can include multiple conditions by using the AND, OR, IN, and NOT IN operators. For example, to find all the rows in SKU\_Data that have a Department named Water Sports and a Buyer named Nancy Meyers, we can code:

**Select \* from sku\_data where Department = "Water Sports" AND Buyer ="Nancy Meyers";**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

Similarly, to find all of the rows of SKU\_Data for either the Camping or Climbing departments, we can code:

**Select \* from sku\_data where Department = "Camping" OR Department ="Climbing";**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

Three or more AND and OR conditions can be combines, but in such cases the IN and NOT IN operators are easier to use. For example, suppose we want to obtain all of the rows in SKU\_DATA for buyers Nancy Meyers, Cindy Lo, and Jerry Martin. We could construct a WHERE clause with two ANDs, but an easier way to do this is to use the IN keyword as follows:

**Select \* from sku\_data Where Buyer IN(“Nancy Meyers”, “Cindy Lo”, “Jerry Martin”);**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

Similarly, if we want to find rows of SKU\_DATA for which the buyer is someone other than Nancy Meyers, or Cindy Lo, or Jerry Martin, we would code:

**Select \* From sku\_data Where Buyer Not In (“Nancy Meyers”, “Cindy Lo”, “Jerry Martin”);**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

**Ranges in WHERE Clauses:** SQL WHERE clauses can specify ranges keyword. For example, the following SQL statement:

**Select \* from order\_item Where ExtendedPrice Between 100 and 200;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

Notice that both the ends of the range, 100 and 200, are included in the resulting table. The preceding SQL statement is equivalent to:

**Select \* from order\_item Where ExtendedPrice >=100 And ExtendedPrice <=200;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

Note, too, that the ORDER BY keyword can be combined with any WHERE clause:

**Select \* from order\_item Where ExtendedPrice Between 100 and 200 Order By OrderNumber DESC;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

**Wildcards in Where Clauses:** The keyword LIKE can be used in WHERE clauses to specify matches on portions of column values. For example, suppose we want to find the rows in the SKU\_Data table for all buyers whose first name is Pete. To find such rows, we use the keyword LIKE with the wildcard character % as follows:

**Select \* From sku\_data Where Buyer LIKE "Pete%";**

**Take a screenshot of your output and insert it here!**

Suppose we want to find the rows in SKU\_DATA for which the SKU\_Description includes the word Tent somewhere in the description. Because the word Tent could be at the front, the end, or in the middle, we need to place a wildcard on both ends of the LIKE phrase as follows:

**Select \* From sku\_data Where SKU\_Description LIKE "%Tent%";**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

**Performing Calculations in SQL Queries:** SQL provides five built-in functions for performing arithmetic on table columns: SUM, AVG, MIN, MAX, and COUNT.

Suppose we want to know the sum of OrderTotal for all of the orders in Retail\_Order. We can obtain that sum as follows:

**Select Sum(OrderTotal) From retail\_order;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

The OrderTotal sum is not a colum in a table, so DBMS has no column name to provide. The result is confusing. We would prefer to have a meaningful column name, and SQL allows us to assign one using the AS keyword. If we specify:

**Select Sum(OrderTotal) as OrderSum From retail\_order;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

The built-in functions can be mixed and matched in a single statement. For example, we can code:

**Select sum(ExtendedPRice) as OrderItemSUM, avg(ExtendedPrice) As OrderItemAvg, min(extendedprice) as orderitemmin, max(extendedprice) as orderitemmax from order\_item;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

The function count counts the number of rows. Suppose we want to count the number of departments in the SKU\_Data table.

**Select count(Department) as DepartmentCount from sku\_data;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

If we want to count the unique values of departments, we need to use the DISTINCT keyword as follows:

**Select count(distinct department) as DepartmentCount from sku\_data;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

**Arithmetic in Select Statements:** It is possible to do basic arithmetic in SQL statements. For example, suppose we want to compute the values of extended price, perhaps we want to verify the accuracy of the data in the ORDER\_ITEM table. To compute the extended price, we code:

**Select quantity \* Price as ExtendedPriceVerification from order\_item;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

**String Manipulation:** Another use for expressions in SQL statements is to perform string manipulation. Suppose we want to combine the Buyer and Department columns into a single column named Sponsor.

**Select Concat(Buyer, ' in ',Department) As Sponsor from sku\_data;**

**Take a screenshot of your output and insert it here!**

**Grouping:** In SQL, rows can be grouped according to common values using the GROUP BY keyword. For example, if you specify GROUP BY Department in a SELECT statement on the SKU\_Data table, the DBMS will first sort all rows by Department and then combine all of the rows having the same value into a group for that department.

**Select Department, Count(\*) as Dept\_SKU\_Count from sku\_data group by Department;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

**Querying Two or More Tables with SQL:** We can query multiple tables via several approaches.

This approach is called “querying multiple tables with joins”

**Select Buyer, ExtendedPrice from sku\_data, Order\_Item Where SKU\_Data.SKU = Order\_ITEM.SKU;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

This approach is called “querying multiple tables with subqueries”

Business Question: What is the sum of extendedprice for items managed by the water sports department?

**Select sum(extendedprice) as revenue from order\_item where sku in (select sku from sku\_data where department ="Water Sports")**

**Important: In this approach subquery gets executed first!**

**MySQL Output**

**Take a screenshot of your output and insert it here!**

We can extend this syntax to join three or more tables. For example, suppose want to obtain the Buyer, ExtendedPrice, and OrderMonth for all purchases of items managed by each buyer. To retrieve that data, we need to join all three tables together as follows:

**Select sku\_data.Buyer, order\_item.ExtendedPrice, retail\_order.OrderMonth from sku\_data, order\_item, retail\_order where sku\_data.SKU = order\_item.SKU and order\_item.OrderNumber = retail\_order.OrderNumber;**

**MySQL Output**

**Take a screenshot of your output and insert it here!**