

## 16: SQL Data Manipulation Language

The SQL data manipulation language (DML) is used to query and modify database data. In this chapter, we will describe how to use the SELECT, INSERT, UPDATE, and DELETE SQL DML command statements, defined below.

- *SELECT* – to query data in the database
- *INSERT* – to insert data into a table
- *UPDATE* – to update data in a table
- *DELETE* – to delete data from a table

In the SQL DML statement:

- Each clause in a statement should begin on a new line.
- The beginning of each clause should line up with the beginning of other clauses.
- If a clause has several parts, they should appear on separate lines and be indented under the start of the clause to show the relationship.
- Upper case letters are used to represent reserved words.
- Lower case letters are used to represent user-defined words.

### SELECT Statement

The SELECT statement, or command, allows the user to extract data from tables, based on specific criteria. It is processed according to the following sequence:

```
SELECT DISTINCT item(s)
FROM table(s)
WHERE predicate
GROUP BY field(s)
ORDER BY fields
```

We can use the SELECT statement to generate an employee phone list from the Employees table as follows:

```
SELECT FirstName, LastName, phone
FROM Employees
ORDER BY LastName
```

This action will display employee's last name, first name, and phone number from the Employees table, seen in Table 16.1.

Last Name	First Name	Phone Number
Hagans	Jim	604-232-3232
Wong	Bruce	604-244-2322

Table 16.1. Employees table.

In this next example, we will use a Publishers table (Table 16.2). (You will notice that Canada is misspelled in the *Publisher Country* field for Example Publishing and ABC Publishing. To correct misspelling, use the UPDATE statement to standardize the country field to Canada – see UPDATE statement later in this chapter.)

Publisher Name	Publisher City	Publisher Province	Publisher Country
Acme Publishing	Vancouver	BC	Canada
Example Publishing	Edmonton	AB	Cnada
ABC Publishing	Toronto	ON	Canda

Table 16.2. Publishers table.

If you add the publisher's name and city, you would use the SELECT statement followed by the fields name separated by a comma:

```
SELECT PubName, city
FROM Publishers
```

This action will display the publisher's name and city from the Publishers table.

If you just want the publisher's name under the display name city, you would use the SELECT statement with *no comma* separating pub\_name and city:

```
SELECT PubName city
FROM Publishers
```

Performing this action will display only the pub\_name from the Publishers table with a "city" heading. If you do not include the comma, SQL Server assumes you want a new column name for pub\_name.

### SELECT statement with WHERE criteria

Sometimes you might want to focus on a portion of the Publishers table, such as only publishers that are in Vancouver. In this situation, you would use the SELECT statement with the WHERE criterion, i.e., WHERE city = 'Vancouver'.

These first two examples illustrate how to limit record selection with the WHERE criterion using BETWEEN. Each of these examples give the same results for store items with between 20 and 50 items in stock.

Example #1 uses the quantity, qty *BETWEEN 20 and 50*.

```
SELECT StorID, qty, TitleID
FROM Sales
WHERE qty BETWEEN 20 and 50 (includes the 20 and 50)
```

Example #2, on the other hand, uses *qty >=20 and qty <=50*.

```
SELECT StorID, qty, TitleID
FROM Sales
WHERE qty >= 20 and qty <= 50
```

Example #3 illustrates how to limit record selection with the WHERE criterion using NOT BETWEEN.

```
SELECT StorID, qty, TitleID
FROM Sales
WHERE qty NOT BETWEEN 20 and 50
```

The next two examples show two different ways to limit record selection with the WHERE criterion using IN, with each yielding the same results.

Example #4 shows how to select records using *province=* as part of the WHERE statement.

```
SELECT *
FROM Publishers
WHERE province = 'BC' OR province = 'AB' OR province = 'ON'
```

Example #5 select records using *province IN* as part of the WHERE statement.

```
SELECT *
FROM Publishers
WHERE province IN ('BC', 'AB', 'ON')
```

The final two examples illustrate how NULL and NOT NULL can be used to select records. For these examples, a Books table (not shown) would be used that contains fields called Title, Quantity, and Price (of book). Each publisher has a Books table that lists all of its books.

Example #6 uses NULL.

```
SELECT price, title
FROM Books
WHERE price IS NULL
```

Example #7 uses NOT NULL.

```
SELECT price, title
FROM Books
WHERE price IS NOT NULL
```

## Using wildcards in the LIKE clause

The LIKE keyword selects rows containing fields that match specified portions of character strings. LIKE is used with char, varchar, text, datetime and smalldatetime data. A *wildcard* allows the user to match fields that contain certain letters. For example, the wildcard province = 'N%' would give all provinces that start with the letter 'N'. Table 16.3 shows four ways to specify wildcards in the SELECT statement in regular express format.

%	Any string of zero or more characters
–	Any single character
[ ]	Any single character within the specified range (e.g., [a-f]) or set (e.g., [abcdef])
[^]	Any single character not within the specified range (e.g., [^a – f]) or set (e.g., [^abcdef])

Table 16.3. How to specify wildcards in the SELECT statement.

In example #1, LIKE 'Mc%' searches for all last names that begin with the letters "Mc" (e.g., McBadden).

```
SELECT LastName
FROM Employees
WHERE LastName LIKE 'Mc%'
```

For example #2: LIKE '%inger' searches for all last names that end with the letters "inger" (e.g., Ringer, Stringer).

```
SELECT LastName
FROM Employees
WHERE LastName LIKE '%inger'
```

In, example #3: LIKE '%en%' searches for all last names that have the letters "en" (e.g., Bennett, Green, McBadden).

```
SELECT LastName
FROM Employees
WHERE LastName LIKE '%en%'
```

## SELECT statement with ORDER BY clause

You use the ORDER BY clause to sort the records in the resulting list. Use ASC to sort the results in ascending order and DESC to sort the results in descending order.

For example, with ASC:

```
SELECT *
FROM Employees
ORDER BY HireDate ASC
```

And with DESC:

```
SELECT *
FROM Books
ORDER BY type, price DESC
```

## SELECT statement with GROUP BY clause

The GROUP BY clause is used to create one output row per each group and produces summary values for the selected columns, as shown below.

```
SELECT type
FROM Books
GROUP BY type
```

Here is an example using the above statement.

```
SELECT type AS 'Type', MIN(price) AS 'Minimum Price'
FROM Books
```

```
WHERE royalty > 10
GROUP BY type
```

If the SELECT statement includes a WHERE criterion where *price is not null*,

```
SELECT type, price
FROM Books
WHERE price is not null
```

then a statement with the GROUP BY clause would look like this:

```
SELECT type AS 'Type', MIN(price) AS 'Minimum Price'
FROM Books
WHERE price is not null
GROUP BY type
```

### Using COUNT with GROUP BY

We can use COUNT to tally how many items are in a container. However, if we want to count different items into separate groups, such as marbles of varying colours, then we would use the COUNT function with the GROUP BY command.

The below SELECT statement illustrates how to count groups of data using the COUNT function with the GROUP BY clause.

```
SELECT COUNT(*)
FROM Books
GROUP BY type
```

### Using AVG and SUM with GROUP BY

We can use the AVG function to give us the average of any group, and SUM to give the total.

Example #1 uses the AVG FUNCTION with the GROUP BY type.

```
SELECT AVG(qty)
FROM Books
GROUP BY type
```

Example #2 uses the SUM function with the GROUP BY type.

```
SELECT SUM(qty)
FROM Books
GROUP BY type
```

Example #3 uses both the AVG and SUM functions with the GROUP BY type in the SELECT statement.

```
SELECT 'Total Sales' = SUM(qty), 'Average Sales' = AVG(qty), stor_id
FROM Sales
GROUP BY StorID ORDER BY 'Total Sales'
```

### Restricting rows with HAVING

The HAVING clause can be used to restrict rows. It is similar to the WHERE condition except HAVING can include the aggregate function; the WHERE cannot do this.

The HAVING clause behaves like the WHERE clause, but is applicable to groups. In this example, we use the HAVING clause to exclude the groups with the province 'BC'.

```
SELECT au_fname AS 'Author's First Name', province as 'Province'
FROM Authors
GROUP BY au_fname, province
HAVING province <> 'BC'
```

### INSERT statement

The *INSERT statement* adds rows to a table. In addition,

- INSERT specifies the table or view that data will be inserted into.
- Column\_list lists columns that will be affected by the INSERT.
- If a column is omitted, each value must be provided.

- If you are including columns, they can be listed in any order.
- VALUES specifies the data that you want to insert into the table. VALUES is required.
- Columns with the IDENTITY property should not be explicitly listed in the column\_list or values\_clause.

The syntax for the INSERT statement is:

```
INSERT [INTO] Table_name | view name [column_list]
DEFAULT VALUES | values_list | select statement
```

When inserting rows with the INSERT statement, these rules apply:

- Inserting an empty string ( ' ' ) into a varchar or text column inserts a single space.
- All char columns are right-padded to the defined length.
- All trailing spaces are removed from data inserted into varchar columns, except in strings that contain only spaces. These strings are truncated to a single space.
- If an INSERT statement violates a constraint, default or rule, or if it is the wrong data type, the statement fails and SQL Server displays an error message.

When you specify values for only some of the columns in the column\_list, one of three things can happen to the columns that have no values:

1. A default value is entered if the column has a DEFAULT constraint, if a default is bound to the column, or if a default is bound to the underlying user-defined data type.
2. NULL is entered if the column allows NULLs and no default value exists for the column.
3. An error message is displayed and the row is rejected if the column is defined as NOT NULL and no default exists.

This example uses INSERT to add a record to the publisher's Authors table.

```
INSERT INTO Authors
VALUES('555-093-467', 'Martin', 'April', '281 555-5673', '816 Market St.,', 'Vancouver', 'BC', 'V7G3P4', 0)
```

This following example illustrates how to insert a partial row into the Publishers table with a column list. The country column had a default value of Canada so it does not require that you include it in your values.

```
INSERT INTO Publishers (PubID, PubName, city, province)
VALUES ('9900', 'Acme Publishing', 'Vancouver', 'BC')
```

To insert rows into a table with an IDENTITY column, follow the below example. Do not supply the value for the IDENTITY nor the name of the column in the column list.

```
INSERT INTO jobs
VALUES ('DBA', 100, 175)
```

### Inserting specific values into an IDENTITY column

By default, data cannot be inserted directly into an IDENTITY column; however, if a row is accidentally deleted, or there are gaps in the IDENTITY column values, you can insert a row and specify the IDENTITY column value.

IDENTITY\_INSERT option

To allow an insert with a specific identity value, the IDENTITY\_INSERT option can be used as follows.

```
SET IDENTITY_INSERT jobs ON
INSERT INTO jobs (job_id, job_desc, min_lvl, max_lvl)
VALUES (19, 'DBA2', 100, 175)
SET IDENTITY_INSERT jobs OFF
```

### Inserting rows with a SELECT statement

We can sometimes create a small temporary table from a large table. For this, we can insert rows with a SELECT statement. When using this command, there is no validation for uniqueness. Consequently, there may be many rows with the same pub\_id in the example below.

This example creates a smaller temporary Publishers table using the CREATE TABLE statement. Then the INSERT with a SELECT statement is used to add records to this temporary Publishers table from the publis table.

```
CREATE TABLE dbo.tmpPublishers (  
  PubID char (4) NOT NULL ,  
  PubName varchar (40) NULL ,  
  city varchar (20) NULL ,  
  province char (2) NULL ,  
  country varchar (30) NULL DEFAULT ('Canada')  
)  
INSERT tmpPublishers  
SELECT * FROM Publishers
```

In this example, we're copying a subset of data.

```
INSERT tmpPublishers (pub_id, pub_name)  
SELECT PubID, PubName  
FROM Publishers
```

In this example, the publishers' data are copied to the tmpPublishers table and the country column is set to Canada.

```
INSERT tmpPublishers (PubID, PubName, city, province, country)  
SELECT PubID, PubName, city, province, 'Canada'  
FROM Publishers
```

### UPDATE statement

The *UPDATE statement* changes data in existing rows either by adding new data or modifying existing data.

This example uses the UPDATE statement to standardize the country field to be Canada for all records in the Publishers table.

```
UPDATE Publishers  
SET country = 'Canada'
```

This example increases the royalty amount by 10% for those royalty amounts between 10 and 20.

```
UPDATE roysched  
SET royalty = royalty + (royalty * .10)  
WHERE royalty BETWEEN 10 and 20
```

### Including subqueries in an UPDATE statement

The employees from the Employees table who were hired by the publisher in 2010 are given a promotion to the highest job level for their job type. This is what the UPDATE statement would look like.

```
UPDATE Employees  
SET job_lvl =  
(SELECT max_lvl FROM jobs  
WHERE employee.job_id = jobs.job_id)  
WHERE DATEPART(year, employee.hire_date) = 2010
```

### DELETE statement

The *DELETE statement* removes rows from a record set. DELETE names the table or view that holds the rows that will be deleted and only one table or row may be listed at a time. WHERE is a standard WHERE clause that limits the deletion to select records.

The DELETE syntax looks like this.

```
DELETE [FROM] {table_name | view_name }  
[WHERE clause]
```

The rules for the DELETE statement are:

1. If you omit a WHERE clause, all rows in the table are removed (except for indexes, the table, constraints).
2. DELETE cannot be used with a view that has a FROM clause naming more than one table. (Delete can affect only one base table at a time.)

What follows are three different DELETE statements that can be used.

1. Deleting all rows from a table.

```
DELETE  
FROM Discounts
```

2. Deleting selected rows:

```
DELETE  
FROM Sales  
WHERE stor_id = '6380'
```

3. Deleting rows based on a value in a subquery:

```
DELETE FROM Sales  
WHERE title_id IN  
(SELECT title_id FROM Books WHERE type = 'mod_cook')
```

## Built-in Functions

There are many built-in functions in SQL Server such as:

1. *Aggregate*: returns summary values
2. *Conversion*: transforms one data type to another
3. *Date*: displays information about dates and times
4. *Mathematical*: performs operations on numeric data
5. *String*: performs operations on character strings, binary data or expressions
6. *System*: returns a special piece of information from the database
7. *Text and image*: performs operations on text and image data

Below you will find detailed descriptions and examples for the first four functions.

## Aggregate functions

Aggregate functions perform a calculation on a set of values and return a single, or summary, value. Table 16.4 lists these functions.

FUNCTION	DESCRIPTION
AVG	Returns the average of all the values, or only the DISTINCT values, in the expression.
COUNT	Returns the number of non-null values in the expression. When DISTINCT is specified, COUNT finds the number of unique non-null values.
COUNT(*)	Returns the number of rows. COUNT(*) takes no parameters and cannot be used with DISTINCT.
MAX	Returns the maximum value in the expression. MAX can be used with numeric, character and datetime columns, but not with bit columns. With character columns, MAX finds the highest value in the collating sequence. MAX ignores any null values.
MIN	Returns the minimum value in the expression. MIN can be used with numeric, character and datetime columns, but not with bit columns. With character columns, MIN finds the value that is lowest in the sort sequence. MIN ignores any null values.
SUM	Returns the sum of all the values, or only the DISTINCT values, in the expression. SUM can be used with numeric columns only.

Table 16.4 A list of aggregate functions and descriptions.

Below are examples of each of the aggregate functions listed in Table 16.4.

**Example #1: AVG**

```
SELECT AVG (price) AS 'Average Title Price'
FROM Books
```

**Example #2: COUNT**

```
SELECT COUNT(PubID) AS 'Number of Publishers'
FROM Publishers
```

**Example #3: COUNT**

```
SELECT COUNT(province) AS 'Number of Publishers'
FROM Publishers
```

**Example #3: COUNT (\*)**

```
SELECT COUNT(*)
FROM Employees
WHERE job_lvl = 35
```

**Example #4: MAX**

```
SELECT MAX (HireDate)
FROM Employees
```

**Example #5: MIN**

```
SELECT MIN (price)
FROM Books
```

**Example #6: SUM**

```
SELECT SUM(discount) AS 'Total Discounts'
FROM Discounts
```

**Conversion function**

The conversion function transforms one data type to another.

In the example below, a price that contains two 9s is converted into five characters. The syntax for this statement is SELECT 'The date is ' + CONVERT(varchar(12), getdate()).

```
SELECT CONVERT(int, 10.6496)
SELECT title_id, price
FROM Books
WHERE CONVERT(char(5), price) LIKE '%99%'
```

In this second example, the conversion function changes data to a data type with a different size.

```
SELECT title_id, CONVERT(char(4), ytd_sales) as 'Sales'
FROM Books
WHERE type LIKE '%cook'
```

**Date function**

The date function produces a date by adding an interval to a specified date. The result is a datetime value equal to the date plus the number of date parts. If the date parameter is a smalldatetime value, the result is also a smalldatetime value.

The DATEADD function is used to add and increment date values. The syntax for this function is DATEADD(datepart, number, date).

```
SELECT DATEADD(day, 3, hire_date)
FROM Employees
```

In this example, the function DATEDIFF(datepart, date1, date2) is used.

This command returns the number of datepart “boundaries” crossed between two specified dates. The method of counting crossed boundaries makes the result given by DATEDIFF consistent across all data types such as minutes, seconds, and milliseconds.

```
SELECT DATEDIFF(day, HireDate, 'Nov 30 1995')
FROM Employees
```



For any particular date, we can examine any part of that date from the year to the millisecond.

The date parts (DATEPART) and abbreviations recognized by SQL Server, and the acceptable values are listed in Table 16.5.

DATE PART	ABBREVIATION	VALUES
Year	yy	1753-9999
Quarter	qq	1-4
Month	mm	1-12
Day of year	dy	1-366
Day	dd	1-31
Week	wk	1-53
Weekday	dw	1-7 (Sun.-Sat.)
Hour	hh	0-23
Minute	mi	0-59
Second	ss	0-59
Millisecond	ms	0-999

Table 16.5. Date part abbreviations and values.

### Mathematical functions

Mathematical functions perform operations on numeric data. The following example lists the current price for each book sold by the publisher and what they would be if all prices increased by 10%.

```
SELECT Price, (price * 1.1) AS 'New Price', title
FROM Books
SELECT 'Square Root' = SQRT(81)
SELECT 'Rounded' = ROUND(4567.9876,2)
SELECT FLOOR (123.45)
```

### Joining Tables

Joining two or more tables is the process of comparing the data in specified columns and using the comparison results to form a new table from the rows that qualify. A join statement:

- Specifies a column from each table
- Compares the values in those columns row by row
- Combines rows with qualifying values into a new row

Although the comparison is usually for equality – values that match exactly – other types of joins can also be specified. All the different joins such as inner, left (outer), right (outer), and cross join will be described below.

#### Inner join

An *inner join* connects two tables on a column with the same data type. Only the rows where the column values match are returned; unmatched rows are discarded.

##### Example #1

```
SELECT jobs.job_id, job_desc
FROM jobs
INNER JOIN Employees ON employee.job_id = jobs.job_id
WHERE jobs.job_id < 7
```

##### Example #2

```
SELECT authors.au_fname, authors.au_lname, books.royalty, title
FROM authors INNER JOIN titleauthor ON authors.au_id=titleauthor.au_id
INNER JOIN books ON titleauthor.title_id=books.title_id
GROUP BY authors.au_lname, authors.au_fname, title, title.royalty
ORDER BY authors.au_lname
```

### Left outer join

A *left outer join* specifies that all left outer rows be returned. All rows from the left table that did not meet the condition specified are included in the results set, and output columns from the other table are set to NULL.

This first example uses the new syntax for a left outer join.

```
SELECT publishers.pub_name, books.title
FROM Publishers
LEFT OUTER JOIN Books ON publishers.pub_id = books.pub_id
```

This is an example of a left outer join using the old syntax.

```
SELECT publishers.pub_name, books.title
FROM Publishers, Books
WHERE publishers.pub_id *= books.pub_id
```

### Right outer join

A *right outer join* includes, in its result set, all rows from the right table that did not meet the condition specified. Output columns that correspond to the other table are set to NULL.

Below is an example using the new syntax for a right outer join.

```
SELECT titleauthor.title_id, authors.au_lname, authors.au_fname
FROM titleauthor
RIGHT OUTER JOIN authors ON titleauthor.au_id = authors.au_id
ORDER BY au_lname
```

This second example show the old syntax used for a right outer join.

```
SELECT titleauthor.title_id, authors.au_lname, authors.au_fname
FROM titleauthor, authors
WHERE titleauthor.au_id =* authors.au_id
ORDER BY au_lname
```

### Full outer join

A *full outer join* specifies that if a row from either table does not match the selection criteria, the row is included in the result set, and its output columns that correspond to the other table are set to NULL.

Here is an example of a full outer join.

```
SELECT books.title, publishers.pub_name, publishers.province
FROM Publishers
FULL OUTER JOIN Books ON books.pub_id = publishers.pub_id
WHERE (publishers.province <> "BC" and publishers.province <> "ON")
ORDER BY books.title_id
```

### Cross join

A *cross join* is a product combining two tables. This join returns the same rows as if no WHERE clause were specified. For example:

```
SELECT au_lname, pub_name,
FROM Authors CROSS JOIN Publishers
```

### Key Terms

**aggregate function:** returns summary values

**ASC:** ascending order

**conversion function:** transforms one data type to another

**cross join:** a product combining two tables

**date function:** displays information about dates and times

**DELETE statement:** removes rows from a record set

**DESC:** descending order

**full outer join:** specifies that if a row from either table does not match the selection criteria

**GROUP BY:** used to create one output row per each group and produces summary values for the selected columns

**inner join:** connects two tables on a column with the same data type

**INSERT statement:** adds rows to a table

**left outer join:** specifies that all left outer rows be returned

**mathematical function:** performs operations on numeric data

**right outer join:** includes all rows from the right table that did not meet the condition specified

**SELECT statement:** used to query data in the database

**string function:** performs operations on character strings, binary data or expressions

**system function:** returns a special piece of information from the database

**text and image functions:** performs operations on text and image data

**UPDATE statement:** changes data in existing rows either by adding new data or modifying existing data

**wildcard:** allows the user to match fields that contain certain letters.

## Exercises

For questions 1 to 18 use the PUBS sample database created by Microsoft. To download the script to generate this database please go to the following site: [www.microsoft.com/en-ca/download...aspx?id=23654](http://www.microsoft.com/en-ca/download...aspx?id=23654).

1. Display a list of publication dates and titles (books) that were published in 2011.
2. Display a list of titles that have been categorized as either traditional or modern cooking. Use the Books table.
3. Display all authors whose first names are five letters long.
4. Display from the Books table: type, price, pub\_id, title about the books put out by each publisher. Rename the column type with "Book Category." Sort by type (descending) and then price (ascending).
5. Display title\_id, pubdate and pubdate plus three days, using the Books table.
6. Using the datediff and getdate function determine how much time has elapsed in months since the books in the Books table were published.
7. List the title IDs and quantity of all books that sold more than 30 copies.
8. Display a list of all last names of the authors who live in Ontario (ON) and the cities where they live.
9. Display all rows that contain a 60 in the payterms field. Use the Sales table.
10. Display all authors whose first names are five letters long, end in O or A, and start with M or P.
11. Display all titles that cost more than \$30 and either begin with T or have a publisher ID of 0877.
12. Display from the Employees table the first name (fname), last name (lname), employee ID(emp\_id) and job level (job\_lvl) columns for those employees with a job level greater than 200; and rename the column headings to: "First Name," "Last Name," "IDENTIFICATION#" and "Job Level."
13. Display the royalty, royalty plus 50% as "royalty plus 50" and title\_id. Use the Roysched table.
14. Using the STUFF function create a string "12xxxx567" from the string "1234567."
15. Display the first 40 characters of each title, along with the average monthly sales for that title to date (ytd\_sales/12). Use the Title table.
16. Show how many books have assigned prices.
17. Display a list of cookbooks with the average cost for all of the books of each type. Use the GROUP BY.

### Advanced Questions (Union, Intersect, and Minus)

1. The relational set operators UNION, INTERSECT and MINUS work properly only if the relations are union-compatible. What does union-compatible mean, and how would you check for this condition?
2. What is the difference between UNION and UNION ALL? Write the syntax for each.
3. Suppose that you have two tables, Employees and Employees\_1. The Employees table contains the records for three employees: Alice Cordoza, John Cretchakov, and Anne McDonald. The Employees\_1 table contains the records for employees: John Cretchakov and Mary Chen. Given that information, what is the query output for the UNION query? List the query output.
4. Given the employee information in question 3, what is the query output for the UNION ALL query? List the query output.
5. Given the employee information in question 3, what is the query output for the INTERSECT query? List the query output.
6. Given the employee information in question 3, what is the query output for the EXCEPT query? List the query output.
7. What is a cross join? Give an example of its syntax.
8. Explain these three join types:
  1. left outer join
  2. right outer join
  3. full outer join
9. What is a subquery, and what are its basic characteristics?
10. What is a correlated subquery? Give an example.
11. Suppose that a Product table contains two attributes, PROD\_CODE and VEND\_CODE. The values for the PROD\_CODE are: ABC, DEF, GHI and JKL. These are matched by the following values for the VEND\_CODE: 125, 124, 124 and 123, respectively (e.g., PROD\_CODE value ABC corresponds to VEND\_CODE value 125). The Vendor table contains a single attribute, VEND\_CODE, with values 123, 124, 125 and 126. (The VEND\_CODE attribute in the Product table is a foreign key to the VEND\_CODE in the Vendor table.)
12. Given the information in question 11, what would be the query output for the following? Show values.
  1. A UNION query based on these two tables
  2. A UNION ALL query based on these two tables
  3. An INTERSECT query based on these two tables
  4. A MINUS query based on these two tables

### Advanced Questions (Using Joins)

1. Display a list of all titles and sales numbers in the Books and Sales tables, including titles that have no sales. Use a join.
2. Display a list of authors' last names and all associated titles that each author has published sorted by the author's last name. Use a join. Save it as a view named: Published Authors.
3. Using a subquery, display all the authors (show last and first name, postal code) who receive a royalty of 100% and live in Alberta. Save it as a view titled: AuthorsView. When creating the view, rename the author's last name and first name as 'Last Name' and 'First Name'.
4. Display the stores that did not sell the title *Is Anger the Enemy*?
5. Display a list of store names for sales after 2013 (Order Date is greater than 2013). Display store name and order date.
6. Display a list of titles for books sold in store name "News & Brews." Display store name, titles and order dates.
7. List total sales (qty) by title. Display total quantity and title columns.
8. List total sales (qty) by type. Display total quantity and type columns.
9. List total sales (qty\*price) by type. Display total dollar value and type columns.
10. Calculate the total number of types of books by publisher. Show publisher name and total count of types of books for each publisher.
11. Show publisher names that do not have any type of book. Display publisher name only.

---

16: SQL Data Manipulation Language is shared under a [CC BY](#) license and was authored, remixed, and/or curated by LibreTexts.

- **1.16: SQL Data Manipulation Language** by Adrienne Watt, Nelson Eng is licensed [CC BY 4.0](#). Original source: <https://opentextbc.ca/dbdesign01/>.