IS Course 8 - SQL Part 2

1 - Introduction

- The SQL Data Manipulation Language (DML) is used to query and modify data in a database.
- We will look at the SQL DML commands:
  - SELECT: to query the 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.

2- SELECT Statement:

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

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

- The use of DISTINCT is optional:
  - it is used to return only distinct (different) values.
- You can use the SELECT statement to return the list of all columns and all rows of the "employee" table:

```
select * from employee;
```

idEmployee	First Name	Last Name	Phone_Number	Birthdate	Salary
1000	Alice	Smith	123-456-7890	1990-01-01	50000
1001	Bob	Johnson	987-654-3210	1985-05-15	60000
1002	Charlie	Brown	555-555-5555	1978-08-20	55000
1003	David	Lee	111-222-3333	1992-12-10	48000
1004	Emily	Davis	NULL	1989-03-25	52000
1005	Alicia	Smith	222-333-4444	1995-07-12	45000

idEmployee	First Name	Last Name	Phone_Number	Birthdate	Salary
1006	Robert	Johnson	888-777-6666	1982-11-28	65000
1007	Alice	Johnson	333-444-5555	1988-02-15	58000
1008	Robert	Johnson	999-888-7777	1980-09-10	62000

You can use the SELECT statement to return a list of employee phone numbers from the Employees table as follows:

```
USE sw;

SELECT FirstName, LastName, Phone_Number
FROM Employee;
```

First Name	Last Name	Phone_Number
Alice	Smith	123-456-7890
Bob	Johnson	987-654-3210
Charlie	Brown	555-555-5555
David	Lee	111-222-3333
Emily	Davis	NULL
Alicia	Smith	222-333-4444
Robert	Johnson	888-777-6666
Alice	Johnson	333-444-5555
Robert	Johnson	999-888-7777

- The SELECT command can be used to return the list of all first names (FirstName field), but in association with the "FirstName" field, the alias "The First Name" is used.

```
SELECT FirstName as "The First Name"
FROM Employee;
```

The First Name
Alice
Charlie
David
Emily

The First Name
Alicia
Robert
Alice
Robert

- The following statement returns the list of unique first names:

```
SELECT distinct FirstName as "The First Name"
FROM Employee;
```

The First Name
Alice
Alicia
Bob
Charlie
David
Emily
Robert

- The following statement returns the list of unique last names:

```
SELECT distinct LastName
FROM Employee;
```

Last Name
Brown
Davis
Johnson
Lee
Smith

The following instruction returns unique first and last names:

```
SELECT DISTINCT Firstname, LastName FROM Employee;
```

First Name	Last Name
Alice	Johnson
Alice	Smith
Alicia	Smith
Bob	Johnson
Charlie	Brown
David	Lee
Emily	Davis
Robert	Johnson

2-1 WHERE Criteria in the SELECT Clause:

- The *WHERE* clause is used to **limit** the selection of records using *BETWEEN*, *IN*, *NULL*, and *NOT NULL*, as well as the use of wildcards in the *LIKE* clause.

```
select FirstName from Employee
where idEmployee=1003
```

First Name
David

- *Example 1* uses BETWEEN to specify: idEmployee between 1002 and 1005

```
SELECT FirstName, LastName
FROM Employee
WHERE idEmployee BETWEEN 1002 and 1005;
```

First Name	Last Name
Charlie	Brown
David	Lee
Emily	Davis
Alicia	Smith

- *Example 2*

```
SELECT FirstName, LastName
FROM Employee
WHERE idEmployee >=1002 and idEmployee<=1005;
```

First Name	Last Name
Charlie	Brown
David	Lee
Emily	Davis
Alicia	Smith

- *Example 3*

```
SELECT FirstName, LastName
FROM Employee
WHERE idEmployee >=1002 and salary<=50000;
```

First Name	Last Name
David	Lee
Alicia	Smith

- *Example 4* illustrates how to limit the selection of records using the WHERE clause by using NOT BETWEEN.

```
SELECT FirstName, LastName
FROM Employee
WHERE idEmployee NOT BETWEEN 1002 AND 1007;
```

First Name	Last Name
Alice	Smith
Bob	Johnson
Robert	Johnson

- *Example 5*

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName = 'Alice' OR FirstName = 'Alicia';
```

First Name	Last Name
Alice	Smith
Alicia	Smith
Alice	Johnson

- *Example 6* selects records using FirstName IN in the WHERE clause.

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName IN ('Alice', 'Alicia');
```

First Name	Last Name
Alice	Smith
Alicia	Smith
Alice	Johnson

- The last two examples illustrate how NULL and NOT NULL can be used to select records
- *Example 7*: uses NULL.

```
select FirstName, LastName from employee
where phone_number is null;
```

First Name	Last Name	
Emily	Davis	

- *Example 8*: uses NOT NULL.

```
select FirstName, LastName from employee
where phone_number is not null;
```

First Name	Last Name
Alice	Smith
Bob	Johnson

First Name	Last Name
Charlie	Brown
David	Lee
Alicia	Smith
Robert	Johnson
Alice	Johnson
Robert	Johnson

2-2 Using Wildcards in the LIKE Clause

- The keyword *LIKE* selects rows containing fields that match specified portions of character strings.
- *LIKE* is used with data of type *char*, *varchar*, *text*, *datetime*, and *smalldatetime*. A wildcard allows the user to match fields containing certain letters.
- The following table shows four ways to specify wildcards in the SELECT statement in regular expression format.

Character	Description
%	Any string of zero or more characters
_	Any single character
[ ]	Any single character in the specified range (e.g., [a-f]) or set (e.g., [abcdef])
[^]	Any single character that is not in the specified range (for example, [^a – f] or the set (for example, [^abcdef])

- *Example 1*: *LIKE 'Sm%'* searches for all last names beginning with the letters "Sm".

```
SELECT FirstName, LastName FROM Employee
WHERE LastName LIKE 'Sm%';
```

First Name	Last Name
Alice	Smith
Alicia	Smith

- *Example 2*: *LIKE '%on'* searches for all last names ending with the letters "on".

```
SELECT FirstName, LastName FROM Employee
WHERE LastName LIKE '%on';
```

First Name	Last Name
Bob	Johnson
Robert	Johnson
Alice	Johnson
Robert	Johnson

- *example 3*: *LIKE '%s%'* searches for all last names containing the letter "s"

```
SELECT FirstName, LastName FROM Employee
WHERE LastName LIKE '%s%';
```

- Other Examples:

First Name	Last Name
Alice	Smith
Bob	Johnson
Emily	Davis
Alicia	Smith
Robert	Johnson
Alice	Johnson
Robert	Johnson

```
SELECT FirstName, LastName FROM Employee
WHERE LastName not LIKE '%s%';
```

First Name	Last Name
Charlie	Brown
David	Lee

```
SELECT FirstName, LastName FROM Employee
WHERE LastName LIKE '_%s%';
```

First Name	Last Name
Bob	Johnson

First Name	Last Name
Emily	Davis
Robert	Johnson
Alice	Johnson
Robert	Johnson

```
SELECT FirstName, LastName FROM Employee
WHERE LastName not LIKE '_%s%';
```

First Name	Last Name
Alice	Smith
Charlie	Brown
David	Lee
Alicia	Smith

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName LIKE '[cb]%';
```

First Name	Last Name
Bob	Johnson
Charlie	Brown

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName LIKE '[b-d]%';
```

First Name	Last Name
Bob	Johnson
Charlie	Brown
David	Lee

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName LIKE '[^a-f]%';
```

First Name	Last Name
Robert	Johnson
Robert	Johnson

```
SELECT FirstName, LastName FROM Employee
WHERE FirstName LIKE '[^abers]%';
```

2-3 The SELECT statement with the ORDER BY clause:

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 idemployee from employee
where idemployee>1005
order by birhdate asc;
```

idEmployee
1008
1006
1007

And with DESC:

```
select idemployee from employee
where idemployee>1005
order by birhdate desc;
```

idEmployee
1007
1006
1008

```
select idemployee from employee
where firstname like '%b%'
```

```
order by firstname asc, birthdate asc;
```

idEmployee
1001
1008
1006

2-4 SELECT Statement with GROUP BY Clause:

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

```
SELECT lastname
FROM employee WHERE idemployee > 1004
GROUP BY lastname;
```

lastname
Johnson
Smith

2-4-1 Using COUNT with GROUP BY:

COUNT can be used to count the number of items in a container. If you want to count different items in separate groups, such as marbles of different colors, you would use the COUNT function with the GROUP BY clause.

```
SELECT COUNT(*) as "Number of Employees" FROM employee;
```

Number of Employees
9

```
SELECT COUNT(Lastname) as "Number of Employees" FROM employee;
```

Number of Employees
9

```
SELECT COUNT(distinct LastName) as "Number of Unique Lastnames " FROM employee;
```

Number of Unique Lastnames
5

```
SELECT LastName, COUNT(Lastname) as "Occurrences"
FROM employee
where idEmployee>1004
group by LastName;
```

LastName	Occurrences
Johnson	3
Smith	1

2-4-2 Using AVG and SUM with GROUP BY:

AVG gives the average of a group, and *SUM* gives the total. Here are examples using these functions with the GROUP BY clause.

```
SELECT LastName, sum(salary) as "Sum Salary"
FROM employee
where idEmployee>1004
group by LastName;
```

LastName	Sum Salary
Johnson	185000
Smith	45000

```
SELECT LastName, avg(salary) as "Avg Salary"
FROM employee
where idEmployee>1004
group by LastName;
```

LastName	Avg Salary
Johnson	61666.6666666667

LastName	Avg Salary
Smith	45000

```
SELECT LastName, sum(salary)/3 as "Sum Salary by 3"
FROM employee
where idEmployee>1004
group by LastName;
```

LastName	Sum Salary by 3
Johnson	61666.6666666667
Smith	15000

2-4-3 Restricting Rows with HAVING:

The *HAVING* clause can be used to restrict rows, similar to the WHERE condition, but applicable to groups.

```
SELECT LastName, sum(salary) as 'Sum Salary'
from Employee
where idEmployee>1004
group by LastName
having sum(salary)> 50000;
```

LastName	Sum Salary
Johnson	185000

```
SELECT LastName, sum(salary) as 'Sum Salary'
from Employee
where idEmployee>1004
group by LastName
having LastName like 'S%';
```

LastName	Sum Salary
Smith	45000

3 - 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 into another
3. Date: Displays date and time information
4. Mathematical: Performs operations on numeric data
5. String: Performs operations on strings, binary data, or expressions
6. System: Returns special information from the database
7. Text and Image: Performs operations on text and image data

3-1 Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single summary value.

FUNCTION	DESCRIPTION
AVG	Returns the average of all values, or only 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-zero 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 sort sequence. MAX ignores 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 lowest value in the sort sequence. MIN ignores null values.
SUM	Returns the sum of all values, or only DISTINCT values, in the expression. SUM can only be used with numeric columns.

Below are examples of each of the aggregate functions listed in the table:

Example 1: AVG

```
SELECT AVG(Salary) AS 'Average Salary'
FROM employee;
```

Average Salary
55000

Example 2: COUNT

```
SELECT count(phone_number) as "Number of Phone Numbers"
FROM employee;
```

Number of Phone Numbers
8

Example 3:

```
SELECT count(lastname) as "Number of Last Names"

FROM employee;
```

Number of Last Names
9

```
SELECT count(distinct lastname) as "Number Last names"
FROM employee;
```

Number Last names
5

Example 4: MAX

```
SELECT max(salary) as 'Max Salary'
FROM employee;
```

Max Salary
65000

```
SELECT max(birthDate) as 'Max Birth Date'
FROM employee;
```

Max Birth Date
1995-07-12

```
SELECT max(birthDate) as 'Max Birth Date'
FROM employee
where idemployee >1008;
```

Max Birth Date
NULL

```
SELECT max(lastname) as 'Max Last Name'
FROM employee;
```

Max Last Name
Smith

Example 5: MIN

```
SELECT min(salary) as 'Min Salary'
FROM employee;
```

Minimum Salary
45000

```
SELECT min(birthDate) as 'Min Birth Date'
FROM employee;
```

Min Birth Date
1978-08-20

```
SELECT min(birthDate) as 'Min Birth Date'
FROM employee
where idemployee >1008;
```

Min Birth Date
NULL

```
select min(phone_number) as "min phone num"
from employee;
```

min phone number
111-222-3333

```
SELECT min(lastname) as 'Min Last Name'
FROM employee;
```

Min Last Name
Brown

Example 6: SUM

```
SELECT SUM(salary) AS 'Sum Salary' FROM Employee;
```

Sum Salary
495000

```
SELECT employee
FROM Employee
WHERE phone_number = (SELECT min(phone_number) FROM employee)
```

idemployee
1003

3-2 Conversion Functions

Conversion functions transform one data type into another.

- Syntax of the Cast function:

```
CAST ( expression AS data_type [ ( length ) ] )
```

- Syntax of the Convert function:

```
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

- Examples:

```
select idemployee
from employee
where convert(char(4),idEmployee) like '%3%';
```

```
select idemployee
from employee
where cast(idEmployee as char(4)) like '%3%';
```

idEmployee
1003

- The following query gives the same result:

```
select idEmployee
from employee
where idEmployee like '%3%';
```

idEmployee
1003

3-3 Date Functions

- Examples of date manipulation functions:
- DATEADD(datepart, number, date): Returns a new datetime value by adding a duration interval (number * specified datepart) to the specified date.
- DATEDIFF(datepart, startdate, enddate): Returns the number of date or datepart endpoints that have passed between two specified dates.

- GETEDATE(): Returns a datetime value containing the date and time of the computer on which the SQL Server instance is running. The returned value does not include time zone offsets.
- ISDATE(): Determines if an input expression of type datetime or smalldatetime has a valid date or time value.
- DAY(date): Returns an integer representing the day portion of the specified date.
- MONTH(date): Returns an integer representing the month portion of a specified date.
- YEAR(date): Returns an integer representing the year portion of a specified date.
- Examples of Use

```
SELECT DATEADD(MONTH, 3, birthDate) as "Add 3 months"
FROM Employee
where idEmployee=1000;
```

Add 3 months
1990-04-01

- Possible values for the datepart parameter:

DATEPART	Note
Year	
Quarter	
Month	
DAY	
WEEK	
HOUR	type of the third parameter DATETIME or TIME with dateadd
MINUTE	type of the third parameter DATETIME or TIME with dateadd
SECOND	type of the third parameter DATETIME or TIME with dateadd
MILLISECOND	type of the third parameter DATETIME or TIME with dateadd

```
SELECT DATEDIFF(YEAR, birthdate, '2024-11-17' ) as 'Age'
FROM Employee
where idEmployee=1000;
```

Age
35

```
SELECT GETDATE() as 'Today';
```

Today
2024-11-17 09:35:19.337

```
SELECT DATEDIFF(YEAR, birthdate,getdate()) as 'Age'
FROM Employee
where idEmployee=1000;
```

Age
34

```
SELECT day(birthdate) as "day"
FROM Employee
where idEmployee=1001;
```

Day
15

```
SELECT month(birthdate) as "Month"
FROM Employee
where idEmployee=1001;
```

Month
5

```
SELECT year(birthdate) as "Year"
FROM Employee
where idEmployee=1001;
```

Year
1985

```
select isdate('20:30') as 'validation';
```

```
select isdate('2024-01-01') as 'validation';
```

```
select isdate('2024-01-01 20:30') as 'validation';
```

validation
1

3-4 Mathematical Functions

- Mathematical functions perform operations on numerical data.

```
SELECT idemployee, salary, (salary * 1.1) AS 'New Salary'
from Employee
where idemployee = '1000';
```

idemployee	salary	New Salary
1000	50000	55000

```
SELECT SQRT(81) as 'Square root';
```

Square root
9

```
SELECT LastName, round(avg(salary),2) as "Avg Salary"
FROM employee
where idEmployee>1004
group by LastName;
```

LastName	Avg Salary
Johnson	61666.67
Smith	45000

```
use sw;
select idEmployee, round(sqrt(salary),2) as "square root"
from employee
where idEmployee>1006;
```

idEmployee	Square root
1007	240.83
1008	249

```
SELECT ROUND(4567.2376,2) as "rounded number";
```

Rounded Number
4567.2400

```
SELECT floor(4567.2376) as "floor int";
```

floor int
4567

```
SELECT idEmployee, floor(4567.2376) as "floor int"
from employee where idemployee >1006;
```

idEmployee	floor int
1007	4567
1008	4567

```
select ceiling(4567.2376) as 'ceiling int';
```

ceiling int
4568

- Other mathematical functions:
  - ABS(): Returns the absolute value
  - PI(): Returns the value of PI
  - LOG(): Returns the natural logarithm of the specified floating-point expression in SQL Server.
  - COS(): Returns the trigonometric cosine of the specified angle - measured in radians - in the specified expression.
  - EXP(): Returns the exponential value of the specified floating-point expression.
- Examples

```
select abs(-305) as 'abs';
```

abs
305

```
select PI() as 'PI';
```

PI
3.14159265358979

```
select Log(205) as 'Log';
```

Log
5.32300997913841

```
select cos(pi()/3) as 'cos';
```


```
select cos(pi()/3) as 'cos';
```

cos
0.5

```
SELECT EXP(LOG(20)) as 'val';
```

val
20

3-5 String Functions

- TRIM(): Removes the space character char(32) or other specified characters from the beginning and end of a string.
- Starting with SQL Server 2022 (16.x), optionally removes the space character char(32) or other specified characters from the beginning, end, or both ends of a string.
- UPPER(): Returns a character expression with lowercase character data converted to uppercase.
- LOWER(): Returns a character expression after converting uppercase character data to lowercase.
- CONCAT(): Returns a string resulting from the concatenation, or joining, of two or more string values in a continuous manner.
- SUBSTRING(): Returns a substring from a character, binary, text, or image expression in SQL Server.
- REPLACE(): Replaces all occurrences of a specified string value with another string value.
- LEN(): Returns the number of characters in the specified string expression, excluding trailing spaces.

```
SELECT  trim('     this is trimming     ') as 'trim';
```

trim
this is trimming

```
SELECT upper(firstname) as 'upper'
from employee
where idEmployee=1008;
```

upper
ROBERT

```
SELECT lower(firstname) as 'lower'
from employee
where idEmployee=1008;
```

lower
robert

```
SELECT  concat(trim(firstname),' ',trim(LastName)) as 'name'
from employee
where idEmployee=1008;
```

concat
Robert Johnson

```
SELECT  SUBSTRING(firstname,1,5) as 'substring'
from employee
where idEmployee=1006;
```

substring
Rober

```
SELECT  replace(FirstName,'R','J')  as 'replace'
from employee
where idEmployee=1006;
```

replace
JobeJt

```
select len('expression') as len;
```

len
10

```
select len(firstname) as len
from Employee
where idEmployee>1005
```

len
6
5
6

4 - Table Joins

Joining two or more tables involves comparing the data in specified columns and using the comparison results to create a new table from the qualifying rows. A *JOIN* operation:

- Specifies one column from each table
- Compares the values in these columns row by row
- Combines the rows with qualifying values into a new row
  While the comparison is generally for equality—values that exactly match—other types of joins can also be specified.
- For the following sections, we will work with these three tables:
- The Book Table:

id_book	title	id_author	publication_Year	Language	id_category
1	The Little Prince	7	1943	French	3
2	1984	2	1949	English	1
3	The Lord of the Rings	9	1954	English	2
4	Moby Dick	4	1851	English	4
5	Les Misérables	3	1862	French	5
6	Pride and Prejudice	6	1813	English	4
7	The Name of the Rose	8	1980	Italian	6
8	Harry Potter and the Philosopher's Stone	1	1997	English	2
9	Dune	5	1965	English	1

- Category Table

id_category	category	subcategory
1	Literature	Science Fiction
2	Literature	Fantasy
3	Literature	Children's Literature
4	Literature	Novel
5	Literature	Historical Novel
6	Literature	Mystery
7	Science	Physics
8	Science	Chemistry
9	History	Antiquity
10	History	Middle Ages

- Author Table

id_author	last_name	first_name
1	Rowling	J.K.
2	Orwell	George
3	Hugo	Victor
4	Melville	Herman
5	Herbert	Frank
6	Austen	Jane
7	Saint-Exupéry	Antoine
8	Eco	Umberto
9	Tolkien	J.R.R.

- The tables belong to the "Library" database, created using the following query:

```
create database Library;
```

- The tables were created using the following queries:
- category:

```
use Library;
create table category(
id_category int not null primary key,
```

```
category char(50) not null,
subcategory char(50) not null,

)
```

- author:

```
create table author(
id_author int not null primary key,
last_name char(50) not null,
first_name char(50) not null,
)
```

- book

```
create table book(
id_book int not null primary key,
title char(100) not null,
id_author int not null,
publication_year int not null,
language char(50),
id_category int not null,
constraint fk_author foreign key(id_author) references Author(id_author),
constraint fk_category foreign key (id_category) references Category
(id_category)

)
```

4-1 Inner Join

An inner join connects two tables based on a column of the same data type. Only rows where the column values match are returned; unmatched rows are ignored.

- Example
- For example, if you want to display the corresponding book title for each subcategory with an ID greater than or equal to 5, only subcategories with a title should be included. Titles without a corresponding category are also not displayed.
- Here's how to do it with an Inner Join:

```
use library;

SELECT category.subcategory, book.title
FROM category
INNER JOIN book
```

```
ON category.id_category = book.id_category
where category.id_category >= 5
```

- The query result would be:

subcategory	title
Historical Novel	Les Misérables
Mystery	The Name of the Rose

- The same result could be obtained without a JOIN as follows:

```
SELECT category.subcategory, book.title
FROM category, book
where category.id_category = book.id_category
and category.id_category >=5
```

4-2 Left outer join

A left outer join specifies that all left outer rows should be returned. All rows in the left table that do not meet the specified condition are included in the result set, and the output columns of the other table are set to NULL.

- Example
- We want to display, for each subcategory that has an ID >= 5, the titles of published books. If the subcategory does not have a matching title, the subcategory will still be published.
- To do this, we can use the following query:

```
SELECT category.subcategory, book.title
FROM category
LEFT OUTER JOIN book
ON category.id_category = book.id_category
WHERE category.id_category >=5
```

- And the result would be:

subcategory	title
Historical Novel	Les Misérables
Mystery	The Name of the Rose
Physics	NULL

subcategory	title
Chemistry	NULL
Antiquity	NULL
Middle Ages	NULL

- We can also display the number of titles per subcategory for categories with an ID >= 5. The query would be:

```
SELECT category.subcategory, count(Book.title) as 'Number'
FROM category
LEFT OUTER JOIN book
ON category.id_category = book.id_category
where category.id_category >= 5
group by category.subcategory;
```

- And the result would be:

subcategory	Number
Antiquity	0
Chemistry	0
Middle Ages	0
Mystery	1
Physics	0
Historical Novel	1

4-3 Right outer join

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

Example

For example, if we want to publish the corresponding book title for each subcategory with an ID greater than or equal to 5, only consider subcategories that have a title. However, if there is a title that does not have a corresponding subcategory, the title will be displayed.

```
SELECT category.subcategory, book.title
FROM category
RIGHT OUTER JOIN book
```

```
ON category.id_category = book.id_category
WHERE book.id_category >= 5
```

- In our case, we will have the same result as for the INNER JOIN:

subcategory	title
Historical Novel	Les Misérables
Mystery	The Name of the Rose

4-4 Full Outer Join

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

Example

```
SELECT category.subcategory, book.title
FROM category
FULL OUTER JOIN book
ON category.id_category = book.id_category
where category.id_category >= 5
```

- In our case, we will have the same result as for the LEFT OUTER JOIN:

subcategory	title
Historical Novel	Les Misérables
Mystery	The Name of the Rose
Physics	NULL
Chemistry	NULL
Antiquity	NULL
Middle Ages	NULL

4-5 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 book.title, author.last_name FROM book
CROSS JOIN author
```

```
where id_book>=7
and author.id_author>=8;
```

title	name
The Name of the Rose	Eco
Harry Potter and the Philosopher's Stone	Eco
Dune	Eco
The Name of the Rose	Tolkien
Harry Potter and the Philosopher's Stone	Tolkien
Dune	Tolkien

5 - INSERT Statement:

The INSERT statement adds rows to a table. Here is an example of using the INSERT statement to add two records to the Author table.

```
INSERT INTO author(id_author, last_name, first_name)
VALUES(10, 'Flaubert', 'Gustave'),

(11, 'Camus', 'Albert');
```

To insert rows into a table with an *IDENTITY* column, the values for the Identity column must not be included.

- The INSERT statement can be used in combination with the SELECT statement to copy values from one table to another:
- Examples

```
INSERT INTO test2 (id_test2, val2, date2)
SELECT id_test1, val1, date1 FROM test1;
```

6 - UPDATE Statement

The *UPDATE* statement modifies the data in existing rows in a table (or view).

This example uses the *UPDATE* statement to update the phone_number field for the row where the phone_number value is NULL:

```
USE sw;
```

```
UPDATE Employee
SET phone_number = '555-555-1212'
where phone_number is NULL
```

- To give the same value to all rows, simply remove the WHERE clause.

This example increases the salary amount by 10% for salaries between 40000 and 50000.

```
USE sw;

UPDATE Employee
SET salary = salary + (salary * 0.1)
WHERE salary BETWEEN 40000 and 50000;
```

6-1 Including Subqueries in an UPDATE Query

- In the following example, books with a title containing "Harry Potter" in the Books table will have the subcategory value in the Category table changed to "Fantasy-Fiction".

```
Use Library
Update Category
SET subcategory = 'Fantasy-Fiction'
WHERE category_id IN (SELECT category_id FROM Book WHERE title like '%Harry
Potter%');
```

7- DELETE Statement

The *DELETE* statement removes rows from a set of records. *DELETE* specifies the table or view containing the rows to be deleted, and only one table or row can be listed at a time. *WHERE* is a standard *WHERE* clause that limits the deletion to selected 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 deleted (except for indexes, the table name, and constraints).
2. DELETE cannot be used with a view that has a FROM clause naming more than one table. (Delete can only affect one base table at a time.)

Here are three different DELETE queries that can be used.

1. Deleting all rows from a table.

```
Use SW;

DELETE FROM Employee
```

2. Deleting selected rows:

```
Use Library;

DELETE FROM Author
WHERE id_author=11;
```

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

- The query deletes authors who do not have a book in the book table.

```
Use library;

DELETE FROM Author
WHERE id_author NOT IN (SELECT DISTINCT id_author FROM book);
```

8 - Advanced Examples of Data Manipulation with the SELECT Statement:

In most examples, we will use the Employee table from the SW database described previously:

8-1 Combining Multiple SELECT Statements

- Considering the Employee table from the SW database, described previously, we want to know the first and last names of the employees with the highest salaries. Therefore, we combine two SELECT statements:
- The included query allows us to find the value of the highest salary.
- The other query allows us to find the ID, last name, first name, and salary of the employees whose salary equals the maximum salary:
- The SQL query would be:

```
use sw;
select idEmployee, firstname, lastname, Salary
from Employee
where salary = (select max(salary) from employee);
```

- The result would be:

idEmployee	firstname	lastname	Salary
1006	Robert	Johnson	65000

8-2 Using the CASE Expression

- The 'CASE' expression evaluates a list of conditions and returns one of several possible result expressions.
- It can take two forms:
- The simple expression `CASE` compares an expression to a set of simple expressions to determine the result.
- The search expression `CASE` evaluates a set of Boolean expressions to determine the result.
- Syntax

```
-- Simple CASE expression:
CASE input_expression
WHEN when_expression THEN result_expression [ ...n ]
[ ELSE else_result_expression ]
END

-- Search CASE expression:
CASE
WHEN Boolean_expression THEN result_expression [ ...n ]
[ ELSE else_result_expression ]
END
```

Examples

- Using the Employee table in the SW database, described previously, we want to create a phone directory containing the IDs, last names, first names, and phone numbers of employees. For employees who do not have a defined phone number value, we return the phrase "No Phone Number".
- The query would be:

```
select idemployee, firstname, lastname,
case
when phone_number is not null then phone_number
else 'No Phone Number'
end as 'Phone Number'
from Employee;
```

- or:

```
select idemployee, firstname, lastname,
case len(phone_number)
when 12 then phone_number
else 'No Phone Number'
end as 'Phone Number'
from Employee;
```

8-3 Extracting Information from Databases

- To get a list of all databases on a SQL Server:

```
select * from sys.databases;
```

- To get only the names of existing databases:

```
select name from sys.databases;
```

- To return the names, IDs, and creation dates of existing databases:

```
SELECT name, database_id, create_date
FROM sys.databases;
```

- To return the list of table names in a database (in the example: DB= sw):

```
USE sw;
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
```

or:

```
USE sw;
SELECT name AS TableName
FROM sys.tables;
```

- To return the list of view names in a database (in the example: DB= sw):

```
use sw;
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'VIEW'
```

or:

```
USE sw;
SELECT name AS ViewName
FROM sys.Views;
```

- To return the list of column names and their data types from the "Employee" table:

```
USE sw;
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Employee';
```

- return the list of constraints from the "Employee" table (DB: sw)):

```
USE sw;
SELECT CONSTRAINT_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'Employee';
```

8-4 Creating a View

- A view is a virtual table whose contents are defined by a query.
- Like a table, a view consists of a set of named data columns and rows.
- The data rows and columns come from tables referenced in the query defining the view and are dynamically generated when the view is referenced.
- To create a view containing only the employee ID, first and last name, and phone number, one of the following two queries can be used:

```
use sw
go
CREATE VIEW Phone_Directory
AS
select idEmployee, firstname, LastName, Phone_Number
from Employee;
```

- or:

```
CREATE VIEW Phone_Directory
AS
select idEmployee, firstname, LastName, Phone_Number
from sw.dbo.Employee;
```

- It is possible to select the values from the table as follows:

```
use sw;
select * from phone_directory;
```

8-5 Combining Multiple Joins

It is possible to combine multiple joins to extract the required information. For example, considering the following tables (belonging to the Ecommerce database):

Products(ProductID, Name, Description, Price, QuantityInStock, CategoryID)
Customers(CustomerID, Last_Name, First_Name, Address, Email, Phone)
Orders(OrderID, CustomerID, OrderDate, OrderStatus)
Categories(CategoryID, Name)
OrderLines(OrderLineID, OrderID, ProductID, Quantity)

Knowing that in:

- Products Table:
  - CategoryID: foreign key to the Categories table
- Orders:
  - CustomerID: foreign key to the Customers table
  - OrderStatus: can take one of the following values: in progress, shipped, Delivery
- OrderLines:
  - OrderID: foreign key to the Orders table
  - ProductID: foreign key to the Products table
- Two Inner Join statements can be combined to retrieve the IDs and dates of orders whose total value exceeds €5000.

```
use Ecommerce;

SELECT OrderLines.OrderID, Orders.OrderDate, SUM(OrderLines.Quantity *
Products.Price) AS TotalOrders

FROM OrderLines

INNER JOIN Orders ON Orders.OrderID = OrderLines.OrderID
```

```
INNER JOIN Products ON OrderLines.ProductID = Products.ProductID

GROUP BY OrderLines.OrderID, Orders.OrderDate

HAVING SUM(OrderLines.Quantity * Products.Price) > 5000;
```

References

- Watt, Adrienne, and Nelson Eng. *Database Design*. 2nd Edition. BCcampus, 2014
- Microsoft 2024, *What are the SQL database functions?*, accessed November 22, 2024, https://learn.microsoft.com/en-us/sql/t-sql/functions/functions?view=sql-server-ver16.
- Microsoft 2024, *Language Elements (Transact-SQL)*, accessed November 22, 2024, https://learn.microsoft.com/en-us/sql/t-sql/language-elements/language-elements-transact-sql?view=sql-server-ver16.
- Microsoft 2024, Databases, accessed November 22, 2024, https://learn.microsoft.com/en-us/sql/relational-databases/databases/databases?view=sql-server-ver16
- Microsoft 2024, # Transact-SQL statements, accessed November 22, 2024, https://learn.microsoft.com/en-us/sql/t-sql/statements/statements?view=sql-server-ver16