NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

VER 1.0

Course MANUAL

INFORMATION MANAGEMENT

This course provides students with the theoretical knowledge and practical skills in the utilization of databases and database management systems in the ICT applications. The logical design, physical design and implementation of relational databases are all covered in this course.

INFORMATION TECHNOLOGY

JULIE ANNE ANGELES-CRYSTAL
ROGER C. PRIMO JR.
MARK JAMES G. CAYABYAB
JOHN IVAN C. MAURAT

INFORMATION MANAGEMENT

Table of Contents

01
Topic 1
DATA VS. INFORMATION
INTRODUCTION TO DATABASE
PURPOSE OF DATABASE
DATABASE ARCHITECTURE

02
Topic 2
DATA MODELLING AND DATA MODELS
THE EVOLUTION OF DATA MODELS

03
Topic 3
RELATIONAL DATABASE MODEL
LOGICALVIEW DATA
DATA DICTIONARY

04
Topic 4
ENTITY RELATIONSHIP MODEL
DEVELOPING ER DIAGRAM

05
Topic 5
EXTEBDED ENTITY RELATIONSHIP MODE
ENTITY INTEGRITY

06
Topic 6
DATABASE TABLES NORMALIZATION
THE NEED FOR NORMALIZATION
THE NORMALIZATION PROCESS

07
Topic 7
INTRODUCTION RO SQL
DATA DEFINNITION COMMANDS
DATA MANIPULATION COMMANDS
SELECT QUERIES

MODULE

#07

NATIONAL UNIVERSITY

MODULE GOALS

- Introduction to SQL
- Data Definition Commands
- Data Manipulation Commands
- SELECT Queries
- Additional Data Definition Commands
- Additional SELECT Query Keywords

FLEX Course Material

Introduction to Structured Query Language (SQL)

WEEK 11-13 MODULE

Education that works.

LESSON #

#7.1

Introduction to SQL

SQL, **S**tructured **Q**uery **L**anguag e, is a programming language designed to manage data stored in relational databases.

SQL operates through simple, declarative statements.

This keeps data accurate and secure, and it helps maintain the integrity of databases, regardless of size.

WHAT IS SQL?

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL COMMANDS

SELECT - extracts data from a database
UPDATE - updates data in a database
DELETE - deletes data from a database
INSERT INTO - inserts new data into a database
CREATE DATABASE - creates a new database
ALTER DATABASE - modifies a database
CREATE TABLE - creates a new table
ALTER TABLE - modifies a table
DROP TABLE - deletes a table
CREATE INDEX - creates an index (search key)
DROP INDEX - deletes an index

SQL SYNTAX

SQL Statements

- Most of the actions you need to perform on a database are done with SQL statements.
- SQL statements consists of keywords that are easy to understand.
- The following SQL statement returns all records from a table named "Customers":

Select all records from the Customers table:

```
SELECT * FROM Customers;
```

Semicolon after SQL Statements?

- Some database systems require a semicolon at the end of each SQL statement.
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.
- In this tutorial, we will use semicolon at the end of each SQL statement.

SQL SYNTAX

The SQL SELECT Statement

- The SELECT statement is used to select data from a database.

Return data from the Customers table:

```
SELECT CustomerName, City FROM Customers;
```

Syntax

```
SELECT column1, column2, ...
FROM table_name;
```

Select ALL columns

- If you want to return all columns, without specifying every column name, you can use the SELECT * syntax:

Return all the columns from the Customers table:

```
SELECT * FROM Customers;
```

The SQL SELECT DISTINCT Statement

- The SELECT DISTINCT statement is used to return only distinct (different) values

Select all the different countries from the "Customers" table:

```
SELECT DISTINCT Country FROM Customers;
```

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SQL SYNTAX

The SQL WHERE Clause

- The WHERE clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.

Select all customers from Mexico:

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

Text Fields vs. Numeric Fields

- SQL requires single quotes around text values (most database systems will also allow double quotes).
- However, numeric fields should not be enclosed in quotes:

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

SQL SYNTAX

Operators in The WHERE Clause

- You can use other operators than the = operator to filter the search.

Select all customers with a CustomerID greater than 80:

```
SELECT * FROM Customers
WHERE CustomerID > 80;
```

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL SYNTAX

The SQL ORDER BY

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.

Sort the products by price:

```
SELECT * FROM Products
ORDER BY Price;
```

DESC

- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Sort the products from highest to lowest price:

```
SELECT * FROM Products
ORDER BY Price DESC;
```

Order Alphabetically

- For string values the ORDER BY keyword will order alphabetically:

Sort the products alphatbetically by ProductName:

```
SELECT * FROM Products
ORDER BY ProductName;
```

SQL SYNTAX

Alphabetically DESC

- To sort the table reverse alphabetically, use the DESC keyword:

Sort the products by ProductName in reverse order:

```
SELECT * FROM Products
ORDER BY ProductName DESC;
```

ORDER BY Several Columns

- The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

Using Both ASC and DESC

- The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

SQL SYNTAX

The SQL AND Operator

- The WHERE clause can contain one or many AND operators.
- The AND operator is used to filter records based on more than one condition, like if you want to return all customers from Spain that starts with the letter 'G':

Select all customers from Spain that starts with the letter 'G':

```
SELECT *
FROM Customers
WHERE Country = 'Spain' AND CustomerName LIKE 'G%';
```

AND vs OR

- The AND operator displays a record if *all* the conditions are TRUE.
- The OR operator displays a record if *any* of the conditions are TRUE.

All Conditions Must Be True

- The following SQL statement selects all fields from Customers where Country is "Germany" AND City is "Berlin" AND PostalCode is higher than 12000:

```
SELECT * FROM Customers
WHERE Country = 'Germany'
AND City = 'Berlin'
AND PostalCode > 12000;
```

SQL SYNTAX

Combining AND and OR

- You can combine the AND and OR operators.
- The following SQL statement selects all customers from Spain that starts with a "G" or an "R".
- Make sure you use parenthesis to get the correct result.

Select all Spanish customers that starts with either "G" or "R":

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');
```

- Without parenthesis, the select statement will return all customers from Spain that starts with a "G", *plus* all customers that starts with an "R", regardless of the country value:

Select all customers that either:
are from Spain and starts with either "G", *or*
starts with the letter "R":

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND CustomerName LIKE 'G%' OR CustomerName LIKE 'R%';
```

The SQL OR Operator

- The WHERE clause can contain one or more OR operators.
- The OR operator is used to filter records based on more than one condition, like if you want to return all customers from Germany but also those from Spain:

Select all customers from Germany or Spain:

```
SELECT *
FROM Customers
WHERE Country = 'Germany' OR Country = 'Spain';
```

SQL SYNTAX

OR vs AND

- The OR operator displays a record if *any* of the conditions are TRUE.
- The AND operator displays a record if *all* the conditions are TRUE.

At Least One Condition Must Be True

- The following SQL statement selects all fields from Customers where either City is "Berlin", CustomerName starts with the letter "G" or Country is "Norway":

```
SELECT * FROM Customers
WHERE City = 'Berlin' OR CustomerName LIKE 'G%' OR Country = 'Norway';
```

Combining AND and OR

- You can combine the AND and OR operators.
- The following SQL statement selects all customers from Spain that starts with a "G" or an "R".
- Make sure you use parenthesis to get the correct result.

Select all Spanish customers that starts with either "G" or "R":

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');
```

- Without parenthesis, the select statement will return all customers from Spain that starts with a "G", *plus* all customers that starts with an "R", regardless of the country value:

Select all customers that either:
are from Spain and starts with either "G", *or*
starts with the letter "R":

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND CustomerName LIKE 'G%' OR CustomerName LIKE 'R%';
```

SQL SYNTAX

The NOT Operator

- The NOT operator is used in combination with other operators to give the opposite result, also called the negative result.
- In the select statement below we want to return all customers that are NOT from Spain:

Select only the customers that are NOT from Spain:

```
SELECT * FROM Customers
WHERE NOT Country = 'Spain';
```

- In the example above, the NOT operator is used in combination with the = operator, but it can be used in combination with other comparison and/or logical operators. See examples below.

NOT LIKE

Select customers that does not start with the letter 'A':

```
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'A%';
```

NOT BETWEEN

Select customers with a customerID not between 10 and 60:

```
SELECT * FROM Customers
WHERE CustomerID NOT BETWEEN 10 AND 60;
```

SQL SYNTAX

NOT IN

Select customers that are not from Paris or London:

```
SELECT * FROM Customers
WHERE City NOT IN ('Paris', 'London');
```

NOT Greater Than

Select customers that are not from Paris or London:

```
SELECT * FROM Customers
WHERE City NOT IN ('Paris', 'London');
```

NOT Less Than

Select customers with a CustomerId not greater than 50:

```
SELECT * FROM Customers
WHERE NOT CustomerID > 50;
```

Note: There is a not-less-then operator: !< that would give you the same result.

SQL SYNTAX

The SQL INSERT INTO Statement

- The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

- It is possible to write the INSERT INTO statement in two ways:
- 1. Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

- 2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

INSERT INTO Example

- The following SQL statement inserts a new record in the "Customers" table:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode,
Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006',
'Norway');
```

SQL SYNTAX

Insert Data Only in Specified Columns

- It is also possible to only insert data in specific columns.
- The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

Insert Multiple Rows

- It is also possible to insert multiple rows in one statement.
- To insert multiple rows of data, we use the same INSERT INTO statement, but with multiple values:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode,
Country)
VALUES
('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),
('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'Sandnes', '4306', 'Norway'),
('Tasty Tee', 'Finn Egan', 'Streetroad 19B', 'Liverpool', 'L1 0AA', 'UK');
```

- Make sure you separate each set of values with a comma ,.

SQL SYNTAX

What is a NULL Value?

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

How to Test for NULL Values?

- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

SQL SYNTAX

The IS NULL Operator

- The IS NULL operator is used to test for empty values (NULL values).
- The following SQL lists all customers with a NULL value in the "Address" field:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

The IS NOT NULL Operator

- The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).
- The following SQL lists all customers with a value in the "Address" field:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

SQL SYNTAX

The SQL UPDATE Statement

- The UPDATE statement is used to modify the existing records in a table.
- UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

UPDATE Table

- The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

UPDATE Multiple Records

- It is the WHERE clause that determines how many records will be updated.
- The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico":

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

Update Warning!

- Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

SQL SYNTAX

The SQL DELETE Statement

- The DELETE statement is used to delete existing records in a table.
- DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

SQL DELETE Example

- The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Delete All Records

- It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
```

- The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

```
DELETE FROM Customers;
```

Delete a Table

- To delete the table completely, use the DROP TABLE statement:

```
DROP TABLE Customers;
```

SQL SYNTAX

The SQL MIN() and MAX() Functions

- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.

MIN Example

Find the lowest price:

```
SELECT MIN(Price)
FROM Products;
```

MAX Example

Find the highest price:

```
SELECT MAX(Price)
FROM Products;
```

SYNTAX

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Set Column Name (Alias)

- When you use MIN() or MAX(), the returned column will be named MIN(*field*) or MAX(*field*) by default. To give the column a new name, use the AS keyword:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

SQL SYNTAX

The SQL COUNT() Function

- The COUNT() function returns the number of rows that matches a specified criterion.

Find the total number of products in the `Products` table:

```
SELECT COUNT(*)
FROM Products;
```

Add a Where Clause

- You can add a WHERE clause to specify conditions:

Find the number of products where `Price` is higher than 20:

```
SELECT COUNT(ProductID)
FROM Products
WHERE Price > 20;
```

Specify Column

- You can specify a column name instead of the asterix symbol (*).
- If you specify a column instead of (*), NULL values will not be counted.

Find the number of products where the `ProductName` is not null:

```
SELECT COUNT(ProductName)
FROM Products;
```

Ignore Duplicates

- You can ignore duplicates by using the DISTINCT keyword in the COUNT function.
- If DISTINCT is specified, rows with the same value for the specified column will be counted as one.

How many *different* prices are there in the `Products` table:

```
SELECT COUNT(DISTINCT Price)
FROM Products;
```

SQL SYNTAX

Use an Alias

- Give the counted column a name by using the AS keyword.

Name the column "number of records":

```
SELECT COUNT(*) AS [number of records]
FROM Products;
```

The SQL SUM() Function

- The SUM() function returns the total sum of a numeric column.

Return the sum of all `Quantity` fields in the `OrderDetails` table:

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

SYNATX

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Add a Where Clause

- You can add a WHERE clause to specify conditions:

Return the number of orders made for the product with `ProductID` 11:

```
SELECT SUM(Quantity)
FROM OrderDetails
WHERE ProductId = 11;
```

SQL SYNTAX

SUM() With an Expression

- The parameter inside the SUM() function can also be an expression.
- If we assume that each product in the OrderDetails column costs 10 dollars, we can find the total earnings in dollars by multiply each quantity with 10:

Use an expression inside the `SUM()` function:

```
SELECT SUM(Quantity * 10)
FROM OrderDetails;
```

- We can also join the OrderDetails table to the Products table to find the actual amount, instead of assuming it is 10 dollars:

Join `OrderDetails` with `Products`, and use `SUM()` to find the total amount:

```
SELECT SUM(Price * Quantity)
FROM OrderDetails
LEFT JOIN Products ON OrderDetails.ProductID = Products.ProductID;
```

The SQL AVG() Function

- The AVG() function returns the average value of a numeric column.

Find the average price of all products:

```
SELECT AVG(Price)
FROM Products;
```

- You can add a WHERE clause to specify conditions:

Return the average price of products in category 1:

```
SELECT AVG(Price)
FROM Products
WHERE CategoryID = 1;
```

SQL SYNTAX

- Give the AVG column a name by using the AS keyword

Name the column "average price":

```
SELECT AVG(Price) AS [average price]
FROM Products;
```

- To list all records with a higher price than average, we can use the AVG() function in a sub query:

Return all products with a higher price than the average price:

```
SELECT * FROM Products
WHERE price > (SELECT AVG(price) FROM Products);
```

The SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- The percent sign % represents zero, one, or multiple characters
- The underscore sign _ represents one, single character

Select all customers that starts with the letter "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

SQL SYNTAX

The _ Wildcard

- The _ wildcard represents a single character.
- It can be any character or number, but each _ represents one, and only one, character.

Return all customers from a city that starts with 'L' followed by one wildcard character, then 'nd' and then two wildcard characters:

```
SELECT * FROM Customers
WHERE city LIKE 'L_nd__';
```

The % Wildcard

- The % wildcard represents any number of characters, even zero characters.

Return all customers from a city that *contains* the letter 'L':

```
SELECT * FROM Customers
WHERE city LIKE '%L%';
```

Starts With

- To return records that starts with a specific letter or phrase, add the % at the end of the letter or phrase.

Return all customers that starts with 'La':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'La%';
```

SQL SYNTAX

Tip: You can also combine any number of conditions using AND or OR operators.

Return all customers that starts with 'a' or starts with 'b':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%' OR CustomerName LIKE 'b%';
```

Ends With

- To return records that ends with a specific letter or phrase, add the % at the beginning of the letter or phrase.

Return all customers that ends with 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Tip: You can also combine "starts with" and "ends with":

Return all customers that starts with "b" and ends with "s":

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'b%s';
```

Contains

- To return records that contains a specific letter or phrase, add the % both before and after the letter or phrase.

Return all customers that contains the phrase 'or'

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

SQL SYNTAX

Combine Wildcards

- Any wildcard, like % and _ , can be used in combination with other wildcards.

Return all customers that starts with "a" and are at least 3 characters in length:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

Return all customers that have "r" in the second position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

Without Wildcard

- If no wildcard is specified, the phrase has to have an exact match to return a result

Return all customers from Spain:

```
SELECT * FROM Customers
WHERE Country LIKE 'Spain';
```

SQL SYNTAX

SQL Wildcard Characters

- A wildcard character is used to substitute one or more characters in a string.
- Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Return all customers that starts with the letter 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Wildcard Characters

Symbol	Description
%	Represents zero or more characters
_	Represents a single character
[]	Represents any single character within the brackets *
^	Represents any character not in the brackets *
-	Represents any single character within the specified range *
{}	Represents any escaped character **

- ** Not supported in PostgreSQL and MySQL databases.*
- *** Supported only in Oracle databases.*

SQL SYNTAX

The SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

Return all customers from 'Germany', 'France', or 'UK'

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

NOT IN

- By using the NOT keyword in front of the IN operator, you return all records that are NOT any of the values in the list.

Return all customers that are NOT from 'Germany', 'France', or 'UK':

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

IN (SELECT)

- You can also use IN with a subquery in the WHERE clause.
- With a subquery you can return all records from the main query that are present in the result of the subquery.

Return all customers that have an order in the **Orders** table:

```
SELECT * FROM Customers
WHERE CustomerID IN (SELECT CustomerID FROM Orders);
```

SQL SYNTAX

The SQL BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

Selects all products with a price between 10 and 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

SYNTAX

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

SQL Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword

```
SELECT CustomerID AS ID
FROM Customers;
```

SQL SYNTAX

AS is Optional

- Actually, in most database languages, you can skip the AS keyword and get the same result:

```
SELECT CustomerID ID
FROM Customers;
```

- When alias is used on column:

```
SELECT column_name AS alias_name
FROM table_name;
```

- When alias is used on table:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

SQL JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

SQL SYNTAX

- Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.
- Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

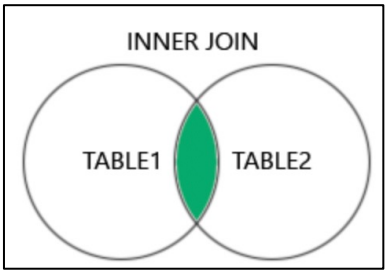
- and it will produce something like this:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different Types of SQL JOINs

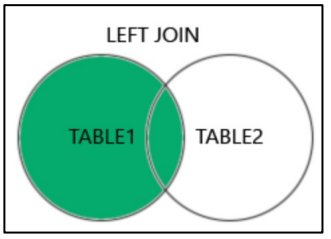
Here are the different types of the JOINs in SQL:

- **(INNER) JOIN:** Returns records that have matching values in both tables

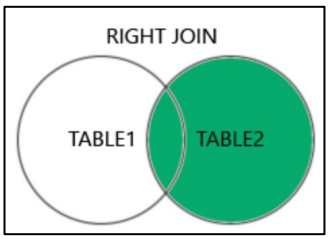


SQL SYNTAX

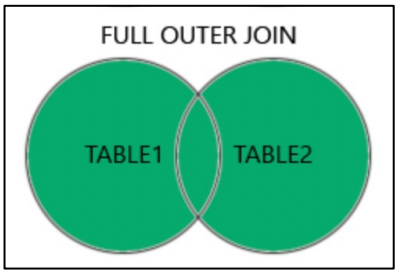
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table



- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table



- **FULL (OUTER) JOIN:** Returns all records when there is a match in either left or right table



SQL SYNTAX

INNER JOIN

- The INNER JOIN keyword selects records that have matching values in both tables.
- Let's look at a selection of the **Products** table:

ProductID	ProductName	CategoryID	Price
1	Chais	1	18
2	Chang	1	19
3	Aniseed Syrup	2	10

- And a selection of the **Categories** table:

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads

- We will join the Products table with the Categories table, by using the CategoryID field from both tables:

Join Products and Categories with the INNER JOIN keyword:

```
SELECT ProductID, ProductName, CategoryName
FROM Products
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;
```

Note: The INNER JOIN keyword returns only rows with a match in both tables. Which means that if you have a product with no CategoryID, or with a CategoryID that is not present in the Categories table, that record would not be returned in the result.

SQL SYNTAX

Naming the Columns

- It is a good practice to include the table name when specifying columns in the SQL statement.

Specify the table names:

```
SELECT Products.ProductID, Products.ProductName, Categories.CategoryName
FROM Products
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;
```

- The example above works without specifying table names, because none of the specified column names are present in both tables. If you try to include CategoryID in the SELECT statement, you will get an error if you do not specify the table name (because CategoryID is present in both tables).

JOIN or INNER JOIN

- JOIN and INNER JOIN will return the same result.
- INNER is the default join type for JOIN, so when you write JOIN the parser actually writes INNER JOIN.

JOIN is the same as INNER JOIN:

```
SELECT Products.ProductID, Products.ProductName, Categories.CategoryName
FROM Products
JOIN Categories ON Products.CategoryID = Categories.CategoryID;
```

SQL SYNTAX

JOIN Three Tables

- The following SQL statement selects all orders with customer and shipper information:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

- SQL LEFT JOIN Keyword
- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.
- LEFT JOIN Syntax
- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.
- LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

SQL SYNTAX

SQL LEFT JOIN Example

- The following SQL statement will select all customers, and any orders they might have:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

- **Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

SQL RIGHT JOIN Keyword

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.
- RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

- **Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

SQL SYNTAX

SQL FULL OUTER JOIN Keyword

- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
- **Tip:** FULL OUTER JOIN and FULL JOIN are the same.
- FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

- **Note:** FULL OUTER JOIN can potentially return very large result-sets!

SQL Self Join

- A self join is a regular join, but the table is joined with itself.
- Self Join Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and *T2* are different table aliases for the same table.

SQL Self Join Example

- The following SQL statement matches customers that are from the same city:

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

SQL SYNTAX

The SQL UNION Operator

- The UNION operator is used to combine the result-set of two or more SELECT statements.
- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order
- UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

- UNION ALL Syntax
- The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

- **Note:** The column names in the result-set are usually equal to the column names in the first SELECT statement.

SQL SYNTAX

The SQL GROUP BY Statement

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.
- GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

- SQL GROUP BY Examples
- The following SQL statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

- The following SQL statement lists the number of customers in each country, sorted high to low:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

SQL SYNTAX

GROUP BY With JOIN Example

- The following SQL statement lists the number of orders sent by each shipper:

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM
Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

The SQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.
- HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

- SQL HAVING Examples
- The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

SQL SYNTAX

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

- The following SQL statement lists the employees that have registered more than 10 orders:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

- The following SQL statement lists if the employees "Davolio" or "Fuller" have registered more than 25 orders:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

SQL SYNTAX

The SQL EXISTS Operator

- The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records.
- EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

- SQL EXISTS Examples
- The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price < 20);
```

- The following SQL statement returns TRUE and lists the suppliers with a product price equal to 22:

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price = 22);
```

SQL SYNTAX

The SQL ANY and ALL Operators

- The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

The SQL ANY Operator

- The ANY operator:
- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition
- ANY means that the condition will be true if the operation is true for any of the values in the range.
- ANY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

The SQL ALL Operator

- The ALL operator:
- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements
- ALL means that the condition will be true only if the operation is true for all values in the range.
- ALL Syntax With SELECT

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

SQL SYNTAX

- ALL Syntax With WHERE or HAVING

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

- **Note:** The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

SQL ANY Examples

- The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10):

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

- The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 99 (this will return TRUE because the Quantity column has some values larger than 99):

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity > 99);
```

SQL SYNTAX

- SQL ALL Examples
- The following SQL statement lists ALL the product names:

```
SELECT ALL ProductName
FROM Products
WHERE TRUE;
```

- The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10):

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

SQL Comments

- Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single Line Comments

- Single line comments start with --.
- Any text between -- and the end of the line will be ignored (will not be executed).
- The following example uses a single-line comment as an explanation:

```
--Select all:
SELECT * FROM Customers;
```

SQL SYNTAX

- The following example uses a single-line comment to ignore the end of a line:

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

- The following example uses a single-line comment to ignore a statement:

```
--SELECT * FROM Customers;
SELECT * FROM Products;
```

Multi-line Comments

- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored.
- The following example uses a multi-line comment as an explanation:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

SQL SYNTAX

- The following example uses a multi-line comment to ignore many statements:

```
/*SELECT * FROM Customers;
SELECT * FROM Products;
SELECT * FROM Orders;
SELECT * FROM Categories;*/
SELECT * FROM Suppliers;
```

- To ignore just a part of a statement, also use the /* */ comment.
- The following example uses a comment to ignore part of a line:

```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

- The following example uses a comment to ignore part of a statement:

```
SELECT * FROM Customers WHERE (CustomerName LIKE 'L%'
OR CustomerName LIKE 'R%' /*OR CustomerName LIKE 'S%'
OR CustomerName LIKE 'T%'*/ OR CustomerName LIKE 'W%')
AND Country='USA'
ORDER BY CustomerName;
```

SQL SYNTAX

SQL Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL SYNTAX

SQL Compound Operators

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressior
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition