INTRO TO SQL
WEEK 2

APRIL 18, 2023 – MAY 4TH 2023

WEEK 2: APRIL 25 & APRIL 28TH

INSTRUCTOR: SYLVIA VARGAS SQLSYLVIA@GMAIL.COM

LINKEDIN: HTTPS://WWW.LINKEDIN.COM/IN/SYLVIAVARGAS/

BLOG: HTTPS://SYLVIAVARGAS.COM/

BLOG (IN PROGRESS): HTTP://SHESATECHIE.ORG/

ALL SLIDES ON GITHUB: HTTPS://GITHUB.COM/SQLSYLVIA/GDI-SQL

WEEK 2 AGENDA – MORE DATA MANIPULATION

We will continue using the **W3Schools** Database for class.

https://www.w3schools.com/sql/trysql.asp?filename=trysql_editor

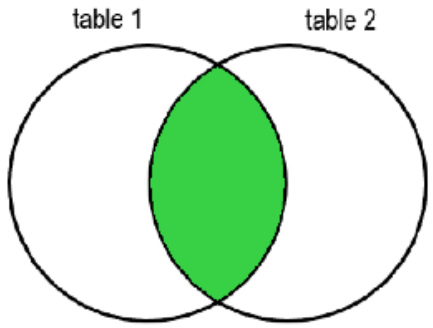
This week's class topics include:

- JOIN REVIEW
- OUTER JOINS
- SUBQUERIES and Common Table Queries
- INSERT
- UPDATE
- ALTER TABLE
- CREATE TABLE

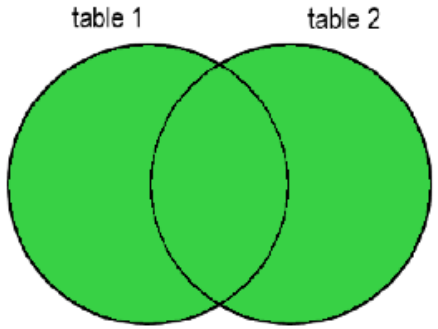
JOIN REVIEW

- These VENN DIAGRAMS illustrate the output of various JOIN commands.

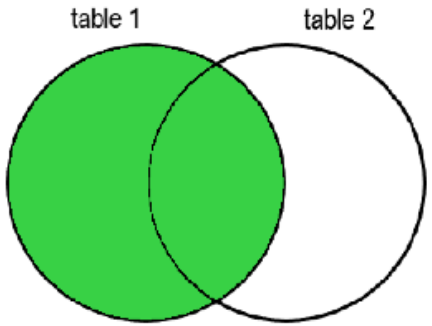
INNER JOIN



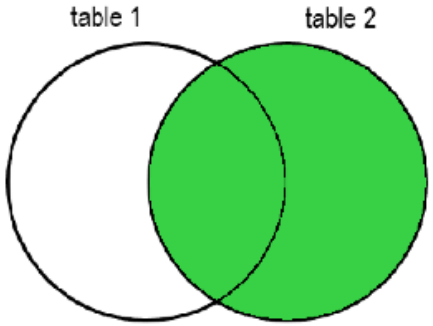
FULL OUTER JOIN
CARTESIAN PRODUCT



LEFT JOIN

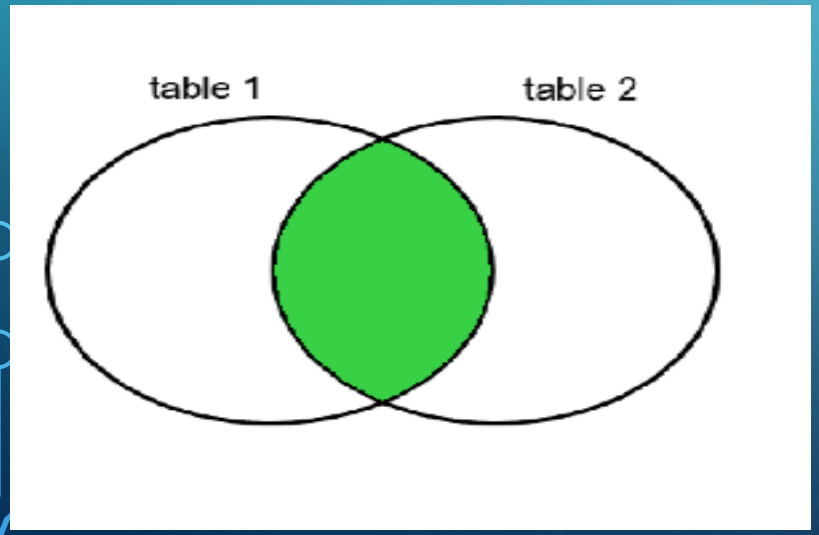


RIGHT JOIN



JOIN

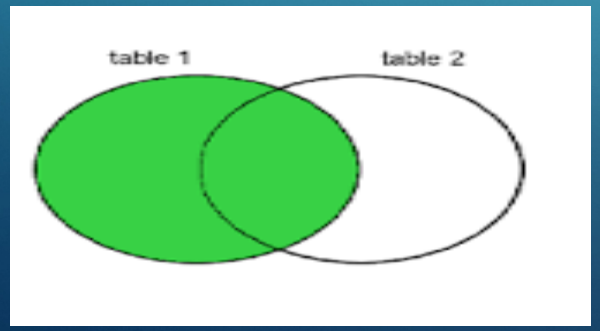
The inner join will keep only the information from the two joined tables that is related




```
SELECT Employees.LastName
      , Orders.orderID
FROM Orders
JOIN Employees
ON Employees.EmployeeID =
Orders.EmployeeID
```

LEFT JOIN = LEFT OUTER JOIN

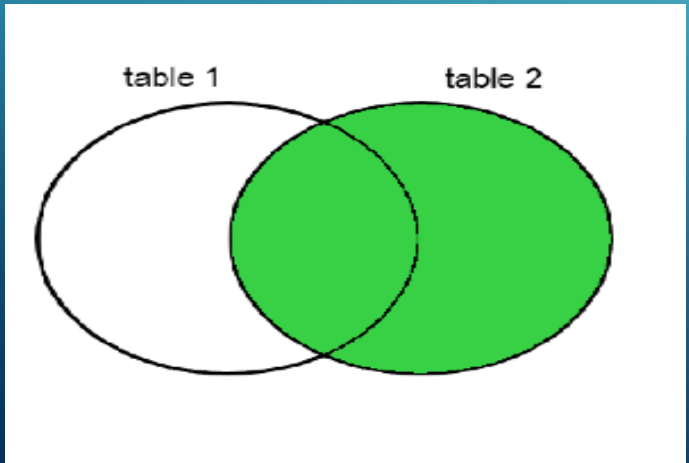
A LEFT JOIN returns all the rows from the left table and the matching rows from the right table. If there is no match in the right table, the result will contain NULL values.




```
SELECT Employees.LastName
, Orders.orderID
FROM Orders
LEFT JOIN Employees
ON Employees.EmployeeID =
Orders.EmployeeID
```

RIGHT JOIN = RIGHT OUTER JOIN

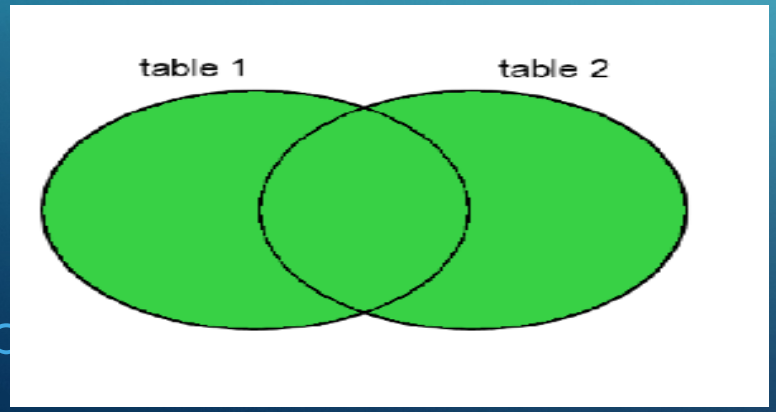
A RIGHT JOIN returns all the rows from the right table and the matching rows from the left table. If there is no match in the left table, the result will contain NULL values.




```
SELECT Employees.LastName
     , Orders.orderID
FROM Orders
RIGHT JOIN Orders
ON Employees.EmployeeID =
Orders.EmployeeID
```

FULL OUTER JOIN = CARTESIAN PRODUCT

A RIGHT JOIN returns all the rows from the right table and the matching rows from the left table. If there is no match in the left table, the result will contain NULL values.



SQL Statement:

```
SELECT Orders.customerID, OrderDetails.ProductID
FROM Orders
JOIN OrderDetails
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 101528

CustomerID	ProductID
90	11
90	42
90	72

REFERENCES FOR SQL JOINS

- https://www.freecodecamp.org/news/sql-join-types-inner-join-vs-outer-join-example/#:~:text=There%20are%20three%20types%20of,RIGHT%20JOIN%20%2C%20and%20FULL%20JOIN%20

HAVING CLAUSE – USE WITH GROUP BY

The **HAVING** statement is used to filter data after it has been grouped by the **GROUP BY** statement.

```
SELECT Orders.customerID,
SUM(OrderDetails.quantity) AS total_quantity
FROM Orders
JOIN OrderDetails
    ON OrderDetails.OrderId =
        Orders.OrderID
GROUP BY Orders.customerID
HAVING SUM(OrderDetails.quantity) > 100
```

UNION

The **UNION** operator is used to combine the results of two or more

SELECT statements into a single result set. The **SELECT** statements must have the same number of columns, and the columns must have compatible data types.

Duplicate rows are automatically removed from the result set.

```
SELECT 'Customer' as PersonType, CustomerName, City, Country
 FROM Customers
 WHERE  Country ='Germany'
UNION
SELECT 'Supplier' as PersonType, SupplierName, City, Country
 FROM Suppliers
WHERE  Country ='Germany'
```

SUBQUERIES

A SUBQUERY IS A QUERY NESTED INSIDE ANOTHER STATEMENT SUCH AS SELECT, INSERT, UPDATE, OR DELETE.

```
SELECT
    AVG(Total) average_product_quantity
FROM
    (SELECT SUM(Quantity) as Total
    FROM OrderDetails
    GROUP BY ProductID)
```

COMMON TABLE EXPRESSIONS (CTE)

- The **WITH** clause is considered “temporary” because the result is not permanently stored anywhere in the database schema. It acts as a temporary view that only exists for the duration of the query.

SUBSELECT VERSUS COMMON TABLE EXPRESS

INSERTING DATA - EXAMPLE 1

```
INSERT INTO Suppliers (supplierID
        , supplierName
        , ContactName
        , Address, City, PostalCode
       , Country, Phone)
VALUES (99
    ,'Sylvia The Supplier’
    , 'Sylvia'
    , 'One Microsoft Way’
    , 'Redmond', '10481’
    , 'USA', '(425) 555-1212’);
```

INSERTING DATA WITH A COMMON TABLE EXPRESSION

```
WITH Data as
( SELECT 99 as SupplierId
      , 'John The Supplier' as SupplierName
      , 'John' as ContactName
      , 'One Microsoft Way' as Address
      , 'Redmond' as City
      , '10481' as PostalCode
      , 'USA' as Country
       , '(425) 555-1212' as Phone )
INSERT INTO Suppliers (supplierID, supplierName,
ContactName, Address, City, PostalCode, Country,
Phone)
SELECT supplierID, supplierName, ContactName
            , Address, City, PostalCode, Country, Phone
FROM Data;
```

UPDATING DATA

```
UPDATE Suppliers
SET Phone='(800) 123-4567'
WHERE SupplierID = 99;
```

DELETING DATA

```
DELETE Suppliers
WHERE SupplierID = 99;
```

ALTER TABLE

```
ALTER TABLE [Employees] ADD
COLUMN Country
VARCHAR(50);
```

OPTIONAL HOMEWORK

Using W3Schools database:
https://www.w3schools.com/sql/trysql.asp?filename=trysql_editor

- Add 2 new customers to the Customers' table.
- Add 1 new supplier to the Suppliers' table.
- Add a new column for Country in the Employees table and Update 2 or more records in the Employee table with values for Country.