

Curso SQL

Nelson López Centeno

Curso SQL

- ☐ Consultas - SELECT
 - ☐ Combinar - JOIN
 - ☐ Anexar - UNION
 - ☐ Subconsultas y CTE
 - ☐ Funciones de ventana
- ☐ Modificar Datos - INSERT, UPDATE, DELETE
- ☐ Modificar Estructura - CREATE, ALTER, DROP
 - ☐ Vistas
 - ☐ Procedimientos Almacenados
 - ☐ Triggers
- ☐ Rendimiento

Configuración del entorno de trabajo



MySQL

<https://dev.mysql.com/downloads/mysql/>



DBeaver

<https://dbeaver.io/download/>

SELECT

**SELECT DISTINCT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT**

Orden de ejecución

**FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
LIMIT**

TIPOS DATOS

Numéricos

INT, SMALLINT, TINYINT, BIGINT

DECIMAL, NUMERIC

FLOAT, REAL

Cadenas Caracteres

CHAR

VARCHAR

TEXT

Fecha y Hora

DATE

TIME

DATETIME

Binario

BINARY

VARBINARY

BLOB

FUNCIONES DE TEXTO

UPPER, LOWER

LEFT, RIGHT

LENGTH

TRIM, LTRIM, RTRIM, BTRIM

SUBSTRING

CONCAT

FUNCIONES DE FECHA Y HORA

CURRENT_DATE

CURRENT_TIME

CURRENT_TIMESTAMP

DATEADD

DATEDIFF

DATEPART

FILTRADO - WHERE

=, <>, <, >, <=, >=

AND, OR, NOT

BETWEEN ... AND

IN (...)

LIKE

CASE

CASE exp

WHEN valor1 THEN res1

WHEN valor2 THEN res2

WHEN valor3 THEN res3

ELSE res4

END

CASE

WHEN cond1 THEN res1

WHEN cond2 THEN res2

WHEN cond3 THEN res3

ELSE res4

END

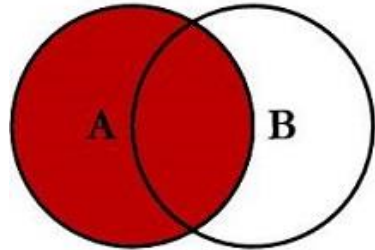
FUNCIONES DE AGREGACIÓN

COUNT
SUM
MIN, MAX
AVG

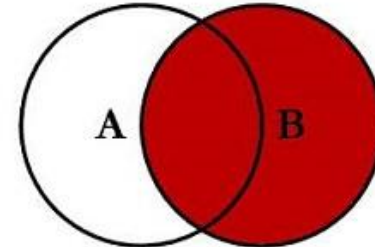
**Estas funciones se utilizan
frecuentemente en
combinación con
GROUP BY y con
HAVING**

JOIN

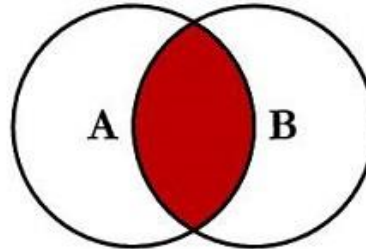
SQL JOINS



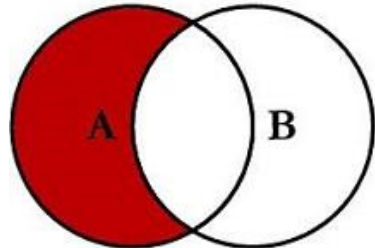
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



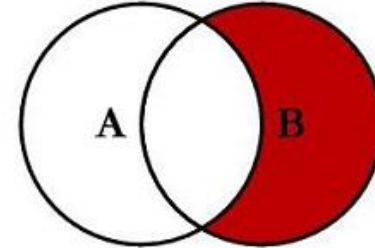
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



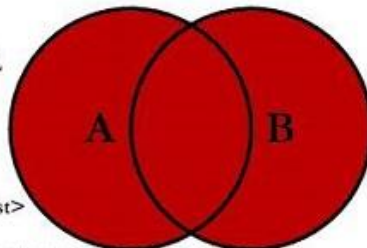
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



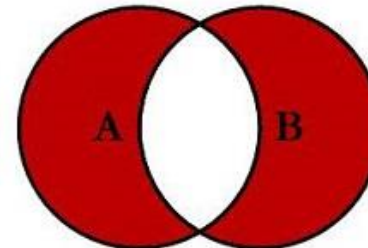
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



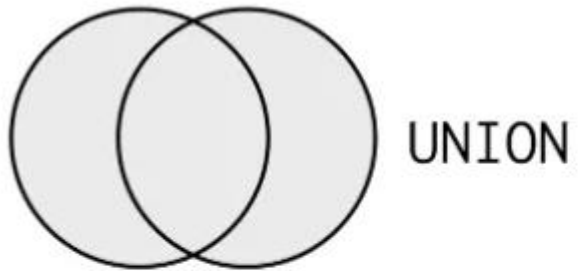
```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

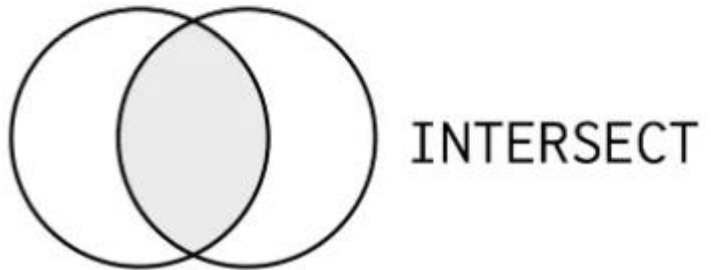
UNION



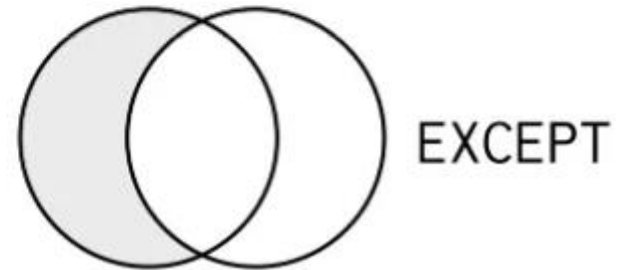
UNION



UNION ALL



INTERSECT



EXCEPT

SUBCONSULTA

Utilizarla como origen de otra consulta

```
SELECT
    SC1.c1,
    SC1.pc3
FROM (
    c1,
    SELECT AVG(c3) AS pc3
    FROM t2
) AS SC1
WHERE SC1.pc3 > 70
```

También se puede
usar con **JOIN**

SUBCONSULTA

Filtrar una consulta basándose en otra

```
SELECT c1
FROM t1
WHERE
    c2 > (
        SELECT AVG(c3)
        FROM t2
    )
```

```
SELECT c1
FROM t1
WHERE
    c2 IN (
        SELECT c3
        FROM t2
        WHERE c4 > 200
    )
```

SUBCONSULTA

Crear nuevas columnas a partir de agregaciones

```
SELECT  
    c1,  
    (  
        SELECT AVG(c3)  
        FROM t2  
    ) AS c2  
FROM t1
```

SUBCONSULTA

Consultas correlacionadas

```
SELECT
  c1,
  c2
FROM t1
WHERE c2 = (
  SELECT MAX(c3)
  FROM t2
  WHERE t1.id = t2.c4
)
```


CTE: Common Table Expression

Organizar consultas complejas

```
WITH cte1 AS (  
    SELECT  
        c1,  
        SUM(c2)  
    FROM t1  
    GROUP BY c1  
)
```

```
SELECT  
    ...  
FROM cte1
```

CTE: Common Table Expression

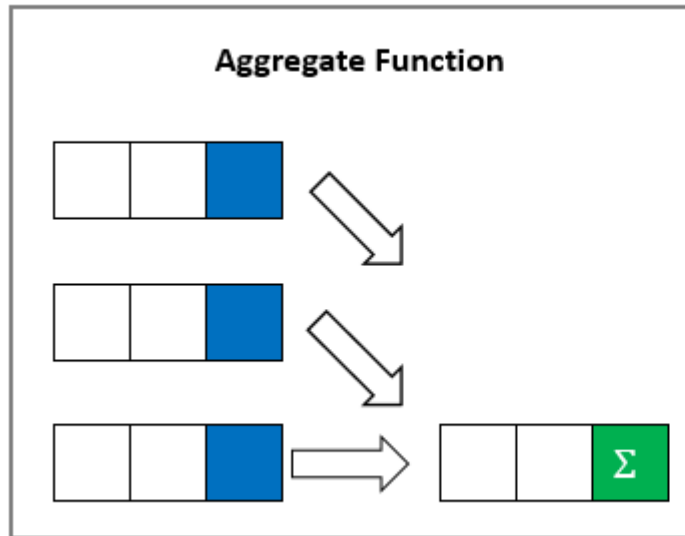
Consulta recursiva

```
WITH cte_recursiva AS (  
    SELECT  
        c1  
    FROM t1  
    INNER JOIN cte_recursiva  
    ...  
)
```

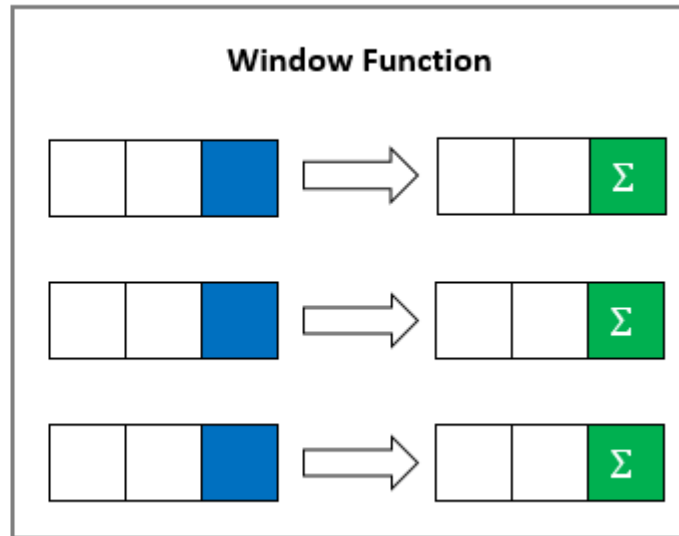
```
SELECT  
    ...  
FROM cte_recursiva  
...
```

Funciones Ventana

```
SELECT  
  c1,  
  c2,  
  SUM(c3)  
FROM T1  
GROUP BY c1,c2
```



```
SELECT  
  c1,  
  c2,  
  SUM(c3) OVER ()  
FROM T1
```



Funciones Ventana

```
SELECT
    spend,
    SUM(spend) OVER (PARTITION BY product ORDER BY transaction_date) AS running_total
FROM product_spend;
```

Output

ORDER BY transaction_date	PARTITION BY product	SUM(spend) spend	running_total
10/07/2022 16:00:00	3.5mm headphone jack	7.99	7.99
07/01/2022 11:00:00	computer mouse	45.00	45.00
08/28/2022 16:00:00	microwave	49.99	49.99
03/01/2023 17:00:00	microwave	34.49	84.48
07/08/2023 16:00:00	microwave	64.95	149.43
12/26/2021 12:00:00	refrigerator	246.00	246.00
03/02/2022 11:00:00	refrigerator	299.99	545.99

Funciones Ventana

OVER se usa con las funciones de agregación:

COUNT(), **SUM()**, **AVG()**, **MIN()**, **MAX()**

y además con:

FIRST_VALUE()

LAST_VALUE()

ROW_NUMBER()

RANK()

DENSE_RANK()

LEAD()

LAG()

Modificar Datos

Al modificar los datos se tienen que **cumplir** las **restricciones** definidas en la base de datos.

- Claves primarias
- Claves foráneas
- Columnas identidad
- Columnas con valores únicos
- Columnas que no admiten NULL
- ...

Modificar Datos - **INSERT**

INSERT INTO T1 (c1, c2)

VALUES

(v1, v2),

(v3, v4),

...

INSERT INTO T1 (c1, c2)

SELECT

c3, c4

FROM T2

WHERE

c5 > 10

Modificar Datos - UPDATE

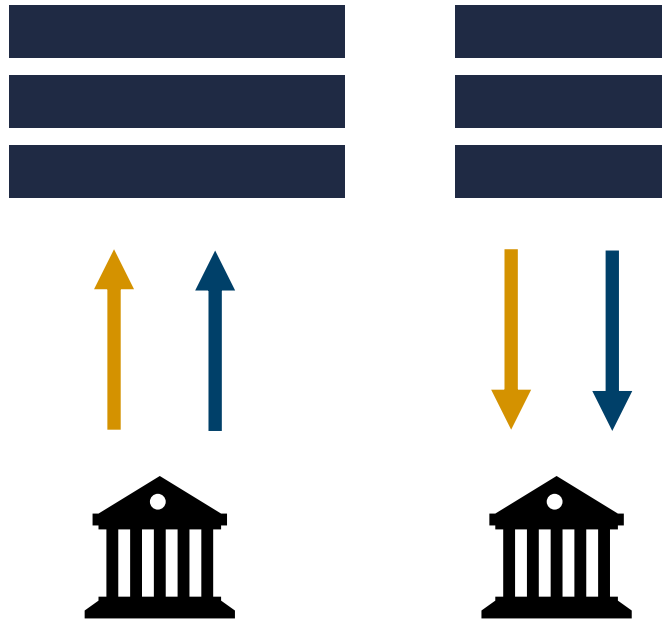
```
UPDATE T1
SET
    c1 = v1,
    c2 = v2
WHERE
    c3 > 10
```

```
UPDATE T1
JOIN T2 ON T1.id = T2.t1_id
SET
    T1.c1 = T2.c1,
    T1.c2 = T2.c2
```


Modificar Datos - **DELETE**

```
DELETE FROM T1  
WHERE  
    c3 > 10
```

Transacciones - **ACID**



Atomicidad

Consistencia

Aislamiento

Durabilidad

Transacciones - ACID

Atomicidad: Garantiza que cada transacción sea tratada como una sola unidad que tiene éxito por completo, o bien fracasa por completo.

Consistencia: (Integridad) Garantiza que una transacción solo pueda llevar los datos de la base de datos de un estado válido a otro.

Aislamiento: Asegura que la ejecución simultánea de transacciones deja la base de datos en el mismo estado que se habría obtenido si las transacciones se hubieran ejecutado secuencialmente.

Durabilidad: Garantiza que una vez que una transacción se confirma, seguirá confirmada incluso si hay un error del sistema, como puede ser un corte de energía o un bloqueo.

Transacciones explícitas

START TRANSACTION;

COMMIT;

ROLLBACK;

Modificar Estructura

CREATE DATABASE BD1

ALTER DATABASE BD1 MODIFY NAME BD1

DROP DATABASE BD2

Modificar Estructura

```
CREATE TABLE T1 (  
    ID int NOT NULL PRIMARY KEY AUTO_INCREMENT,  
    ID2 int NOT NULL,  
    C1 varchar(50) NOT NULL,  
    C2 varchar(50) NULL,  
    C3 int UNIQUE,  
    FOREIGN KEY (ID2) REFERENCES T2(ID),  
    INDEX C1C2 (C1,C2)  
);
```

```
ALTER TABLE T1 ADD C4 int
```

```
DROP TABLE T1
```

VISTAS

Definición de la vista

```
CREATE VIEW V1 AS  
SELECT  
    c1 AS 'Columna Uno',  
    c2 AS 'Columna Dos'  
FROM T1  
JOIN T2 ON T1.id = T2.id_t1
```

Uso de la vista

```
SELECT * FROM V1  
WHERE  
    c1 > 150
```

También se pueden usar con INSERT o UPDATE pero con algunas restricciones.

En algunos gestores de BD se pueden crear **vistas materializadas**.

VARIABLES

```
SET @v1 = 1;  
SET @v2 = 'Hola';  
SET @v3 = (SELECT MIN(first_name) FROM customer);  
SELECT MAX(first_name) INTO @v4 FROM customer;  
  
SELECT @v1, @v2, @v3, @v4;
```


PROCEDIMIENTOS ALMACENADOS

Definición del procedimiento almacenado

```
CREATE PROCEDURE citycount (IN country CHAR(3), OUT cities INT)
BEGIN
    SELECT COUNT(*) INTO cities FROM world.city
    WHERE CountryCode = country;
END
```

Uso del procedimiento almacenado

```
CALL citycount('ESP', @cities);
SELECT @cities;
```

FUNCIONES

Definición de la función

```
CREATE FUNCTION citycount (IN country CHAR(3)) RETURNS INT
BEGIN
    SELECT COUNT(*) INTO @cities FROM world.city
    WHERE CountryCode = country;
    RETURN @cities;
END
```

Uso de la función

```
SELECT citycount('ESP');
```

TRIGGERS

Definición del trigger

```
CREATE TRIGGER ins_sum BEFORE INSERT ON T1  
    FOR EACH ROW SET @sum = @sum + NEW.amount
```

Uso del trigger

```
SET @sum = 0;  
INSERT INTO T1  
    VALUES(137,14.98),(141,1937.50),(97,-100.00);  
SELECT @sum;
```

TRIGGERS

```
CREATE TRIGGER upd_check BEFORE UPDATE ON T1
  FOR EACH ROW
  BEGIN
    IF NEW.amount < 0 THEN
      SET NEW.amount = 0;
    ELSEIF NEW.amount > 100 THEN
      SET NEW.amount = 100;
    END IF;
  END;
END;
```