Universidad Autónoma de Chile
Facultad de Ingeniería
Ingeniería en Informática

Solución guía Nº 05

Base de Datos II

Docente: Sergio Antonio Baltierra Valenzuela

28 **de abril de** 2020

Operadores SQL

Requisitos de Software

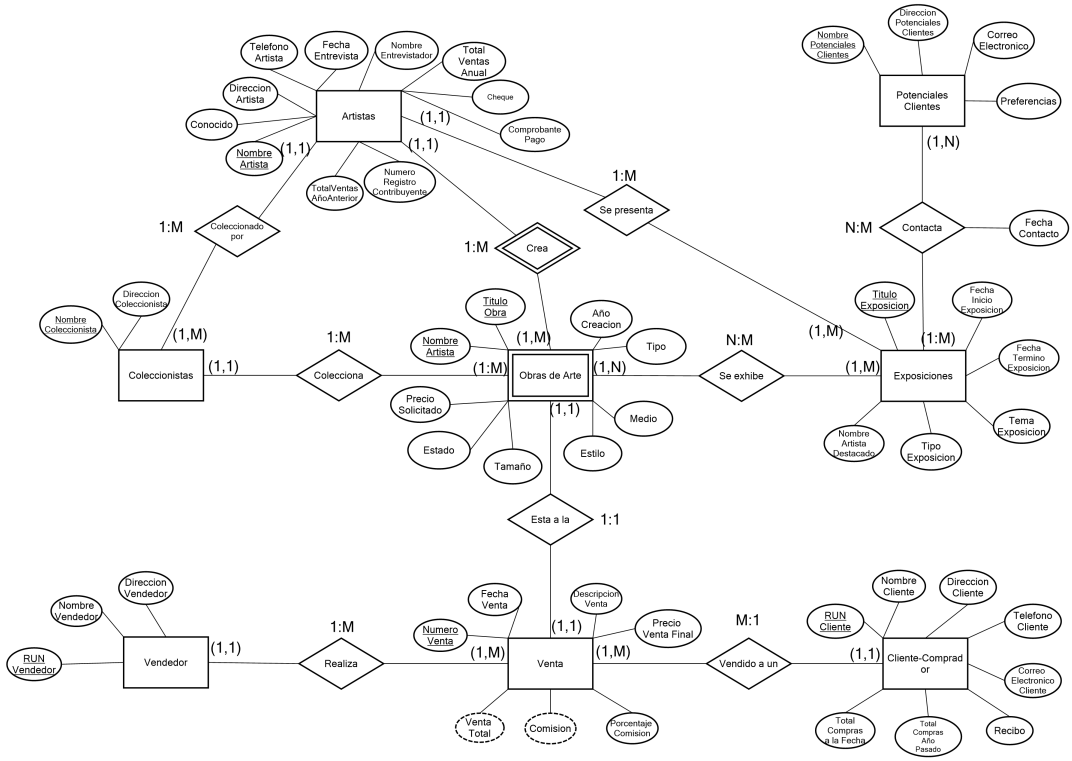
A continuación para el desarrollo de esta guía debe tener instalado en su equipo los siguientes softwares:

- Editor de código como: Sublime Text 3, Visual Studio Code o similares.

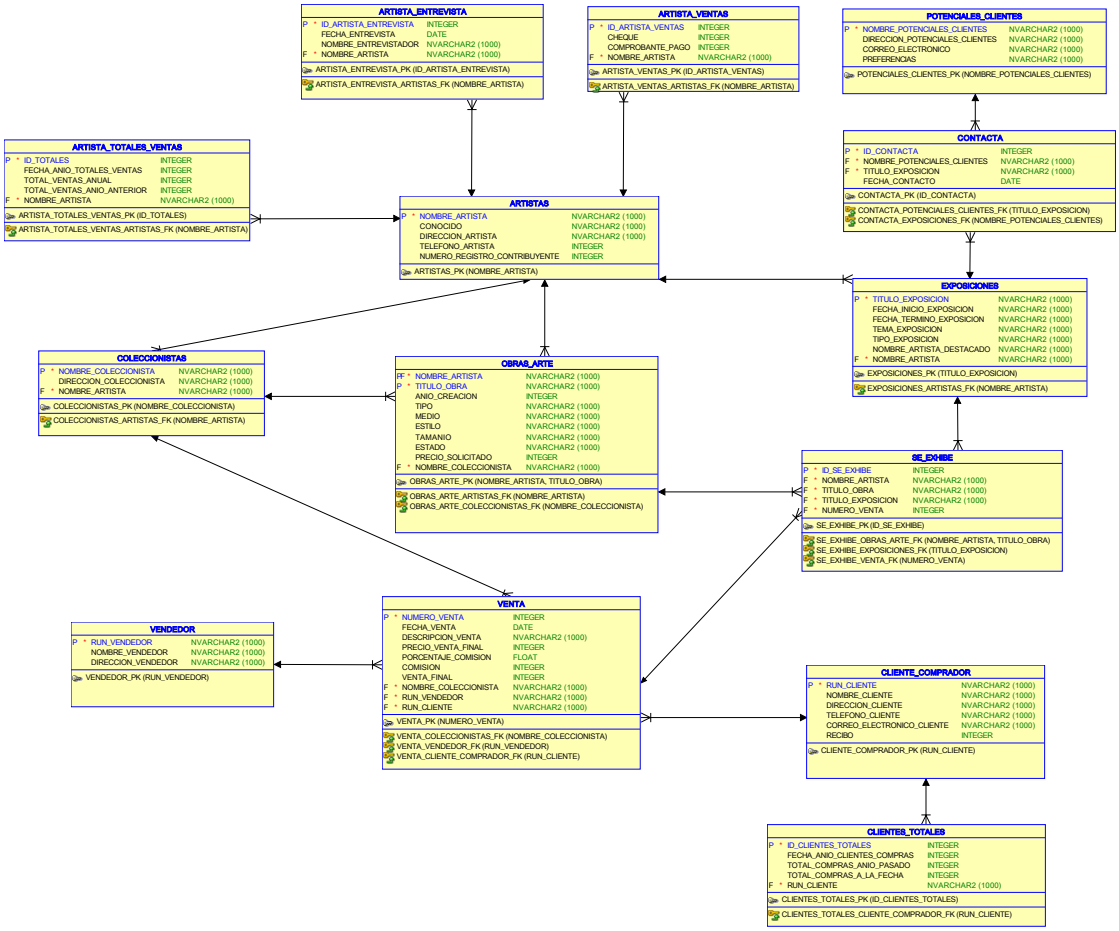
Enunciado

A continuación se presentan los modelos MER y MR:

Modelo MER



Modelo MR



Se pide

Dado el modelo, usted debe:

- Usar los siguientes operadores en consultas de lectura (SELECT):
  - SELECT DISTINCT
  - SELECT TOP
  - HAVING
  - funciones NULL
  - UNION y UNION ALL
  - (self) JOIN

Solución

SELECT DISTINCT

SELECT DISTINCT se usa para devolver solo valores distintos (diferentes).

Dentro de una tabla o consulta, una columna a menudo contiene muchos valores duplicados; y a veces solo desea enumerar los diferentes valores (distintos).

Un ejemplo: Mostrar los distintos campos RUN_VENDEDOR y NOMBRE_VENDEDOR de la tabla VENDEDOR y el campo NOMBRE_COLECCIONISTA de la tabla VENTA.

```
SELECT DISTINCT
    V.RUN_VENDEDOR,
    V.NOMBRE_VENDEDOR,
    VE.NOMBRE_COLECCIONISTA
FROM VENDEDOR V
JOIN VENTA VE ON VE.RUN_VENDEDOR = V.RUN_VENDEDOR
```

SELECT TOP

SELECT TOP se usa para especificar el número de registros a devolver.

La cláusula SELECT TOP es útil en tablas grandes con miles de registros. Devolver una gran cantidad de registros puede afectar el rendimiento.

Observación: la sintaxis de SELECT TOP es distinta según el SGBD. En MySQL hay que usar al final de la consulta el operador LIMIT acompañado del número de tuplas o registros a mostrar.

Un ejemplo: mostrar los dos primeros registros de la tabla ARTISTAS.

```
SELECT
    *
FROM `ARTISTAS`
LIMIT 2;
```

HAVING

HAVING se agregó a SQL porque el operador WHERE no se puedo usar con funciones.

Un ejemplo: Mostrar los campos NOMBRE_CLIENTE y RECIBO de la tabla CLIENTE_COMPRADOR, y el campo TOTAL_COMPRAS_A_LA_FECHA de la tabla CLIENTES_TOTALES. Teniendo un cantidad de NUMERO_VENTA mayor o igual a dos.

```
SELECT
      COUNT(V.NUMERO_VENTA) COUNT_NUMERO_VENTA,
      CC.NOMBRE_CLIENTE,
      CC.RECIBO,
      CT.TOTAL_COMPRAS_A_LA_FECHA
FROM VENTA V
JOIN CLIENTE_COMPRADOR CC ON CC.RUN_CLIENTE = V.
    RUN_CLIENTE
JOIN CLIENTES_TOTALES CT ON CT.RUN_CLIENTE = CC.
    RUN_CLIENTE
GROUP BY CC.NOMBRE_CLIENTE, CC.RECIBO, CT.
    TOTAL_COMPRAS_A_LA_FECHA
HAVING COUNT(V.NUMERO_VENTA) >= 2;
```

funciones NULL

Las funciones NULL nos permiten manipular valores nulos en una consulta, esta manipulación es útil cuando queremos realizar una operación.

Un ejemplo: en la tabla ARTISTA_TOTALES_VENTAS, modificamos el valor del campo TOTAL_VENTAS_ANIO_ANTERIOR a null donde ID_TOTALES = 3.

```
UPDATE ARTISTA_TOTALES_VENTAS
SET TOTAL_VENTAS_ANIO_ANTERIOR = NULL
WHERE ID_TOTALES = 3;
```

En MySQL existen dos funciones que se usan para manipular valores nulos:

IFNULL función que retorna un valor alternativo, predefinido, cuando un valor es nulo.

Un ejemplo: Mostrar los campos NOMBRE_ARTISTA, TOTAL_VENTAS_ANUAL, TOTAL_VENTAS_ANIO_ANTERIOR, agregando una cuarta columna que sería la suma de TOTAL_VENTAS_ANUAL y TOTAL_VENTAS_ANIO_ANTERIOR de la tabla ARTISTA_TOTALES_VENTAS.

```
SELECT
      NOMBRE_ARTISTA,
      TOTAL_VENTAS_ANUAL,
      TOTAL_VENTAS_ANIO_ANTERIOR,
      (TOTAL_VENTAS_ANUAL + IFNULL(
          TOTAL_VENTAS_ANIO_ANTERIOR,0)) SUMA_TOTAL_VENTAS
FROM ARTISTA_TOTALES_VENTAS;
```

Como se puede observar, IFNULL usa dos parámetros, el primer parámetro es el campo a evaluar, si este campo es null, se usará un valor definido en el segundo parámetro de la función IFNULL.

COALESCE función tiene como parámetros de entrada varios valores, retornando el primer argumento no NULL. En caso de que todos los argumentos sean NULL, la función COALESCE devuelve NULL.

Un ejemplo: Mostrar los campos NOMBRE_ARTISTA, TOTAL_VENTAS_ANUAL, TOTAL_VENTAS_ANIO_ANTERIOR, agregando una cuarta columna que sería la suma de TOTAL_VENTAS_ANUAL y TOTAL_VENTAS_ANIO_ANTERIOR de la tabla ARTISTA_TOTALES_VENTAS.

```
SELECT
    NOMBRE_ARTISTA,
    TOTAL_VENTAS_ANUAL,
    TOTAL_VENTAS_ANIO_ANTERIOR,
    (TOTAL_VENTAS_ANUAL + COALESCE(
        TOTAL_VENTAS_ANIO_ANTERIOR, 0 ) ) SUMA_TOTAL_VENTAS
FROM ARTISTA_TOTALES_VENTAS;
```

Como se puede observar, COALESCE usa varios parámetros, en nuestro caso solo dos, donde el primer parámetro será el campo a evaluar, si es nulo, la función COALESCE pasa al segundo parámetro , si no es nulo retorna su valor, que en nuestro caso es cero.

UNION y UNION ALL

UNION se usa para combinar dos o mas queries que contienen datos relacionados, por lo que las **columnas o campos deben ser del mismo tipo de dato**.

La sintaxis es:

```
SELECT COLUMN_A, COLUMN_B FROM TABLE_A

UNION

SELECT COLUMN_C, COLUMN_D FROM TABLE_B
```

Cada instrucción **SELECT dentro de UNION debe tener el mismo número de columnas**. Donde:

- Las columnas también deben tener tipos de datos similares.
- Las columnas en cada instrucción SELECT también deben estar en el mismo orden.

Un ejemplo: Unir las tablas VENTA y CLIENTE_TOTALES, especificamente los campos RUN_CLIENTE y VENTA_FINAL de la tabla VENTA y los campos RUN_CLIENTE y TOTAL_COMPRAS_A_LA_FECHA de la tabla CLIENTES_TOTALES.

```
SELECT
    RUN_CLIENTE,
    VENTA_FINAL
FROM VENTA

UNION

SELECT
    RUN_CLIENTE,
    TOTAL_COMPRAS_A_LA_FECHA
FROM CLIENTES_TOTALES;
```

UNION ALL es similar a UNION, con la diferencia que no elimina valores duplicados.

Un ejemplo: Mostrar usando UNION ALL las tablas VENTA y CLIENTE_TOTALES, especificamente los campos RUN_CLIENTE y VENTA_FINAL de la tabla VENTA y los campos RUN_CLIENTE y TOTAL_COMPRAS_A_LA_FECHA de la tabla CLIENTES_TOTALES.

```
SELECT
    RUN_CLIENTE,
    VENTA_FINAL
FROM VENTA

UNION ALL

SELECT
    RUN_CLIENTE,
    TOTAL_COMPRAS_A_LA_FECHA
FROM CLIENTES_TOTALES;
```

(self) JOIN

(self) JOIN es un tipo de combinación, con la diferencia que la combinación es con la misma tabla.

Un ejemplo: Dada la tabla VENTA, realizar un self JOIN el cual detecte un RUN_VENDEDOR sea un RUN_CLIENTE, para ello primero debemos ingresar un nuevo CLIENTE_COMPRADOR y una nueva VENTA.

```
INSERT INTO `CLIENTE_COMPRADOR`
(`RUN_CLIENTE`, `NOMBRE_CLIENTE`, `DIRECCION_CLIENTE`,
    `TELEFONO_CLIENTE`,
        `CORREO_ELECTRONICO_CLIENTE`, `RECIBO`)
VALUES ('00.000.000-0', 'Aquiles Baeza', 'Privet Drive
   N. 4', '444444444',
        'aquiles.baeza@challa.cl', '4');


INSERT INTO `VENTA`
        (`NUMERO_VENTA`, `FECHA_VENTA`, `
           DESCRIPCION_VENTA`, `PRECIO_VENTA_FINAL`,
                `PORCENTAJE_COMISION`, `COMISION`, `
                   VENTA_FINAL`, `NOMBRE_COLECCIONISTA
                   `,
                `RUN_VENDEDOR`, `RUN_CLIENTE`)
VALUES ('4', '2020-04-28 9:00:0', 'Compra de pintura',
   '300000',
                '11', '33000', '333000', 'Christian
                   Villamil', '00.000.000-0', '
                   00.000.000-0');
```

Realizada la inserción de datos, debemos mostrar los campos NUMERO_VENTA, RUN_CLIENTE, FECHA_VENTA y VENTA_FINAL de la tabla VENTA 1 y el campo RUN_VENDEDOR de la tabla VENTA 2. Donde el RUN_CLIENTE de la tabla VENTA 1 sea igual al RUN_VENDEDOR de la tabla VENTA 2.

```
SELECT DISTINCT
    V1.NUMERO_VENTA,
    V1.RUN_CLIENTE,
    V1.FECHA_VENTA,
    V2.RUN_VENDEDOR,
    V1.VENTA_FINAL
FROM VENTA V1, VENTA V2
WHERE V1.RUN_CLIENTE = V2.RUN_VENDEDOR;
```

Observación: se recomienda usar self JOIN cuando se busca un dato o valor el cual está contenido en dos columnas de la misma tabla, en el ejemplo anterior, un vendedor puede comprar una obra de arte, por ende el RUN_CLIENTE es igual al RUN_VENDEDOR.

Otro uso de self JOIN son en tablas con relación recursiva, el cual la

clave primaria (PK) debe hacer referencia a sí misma en una nueva columna o columna diferente de la misma tabla, clave foránea (FK).