

Práctica 07

Lenguaje de Manipulación de Datos (DML) Básico

1. Objetivo General

Insertar, actualizar, consultar y eliminar los datos de una base de datos.

2. Objetivos Secundarios

- Conocer la sintaxis correcta para realizar funciones de manipulación de datos dentro de un SDBD.
- Introducir el concepto de consulta básica.

3. Introducción

Una base de datos, cualquiera que ésta sea, por ejemplo: un archivero, tiene entre sus finalidades, el almacenar datos de una manera ordenada para su posterior consulta, actualización, inserción y eliminación, dependiendo de las necesidades del cliente. Esta misma finalidad aplica para las bases de datos computacionales, es por ello que resulta fundamental conocer el mecanismo para realizar cualquiera de estas cuatro actividades dentro del SDBD que se encarga de la administración de la base de datos sobre la cual se pretende trabajar.

Como se vio en *Lenguaje de Definición de Datos (DDL)* (Práctica 05), el lenguaje que se propone en estas prácticas para comunicarse con el SDBD es SQL. Este lenguaje maneja varias palabras reservadas para la parte de DDL pero también para la parte de DML.

Para su estudio, podemos dividir las palabras reservadas de SQL para DML en cuatro categorías:

1. Inserción de datos
2. Actualización de datos
3. Consulta de datos
4. Eliminación de registros

3.1. Inserción de datos

Una base de datos relacional contiene tablas con columnas que poseen características singulares y que se relacionan entre sí para modelar un problema de la realidad. Estas tablas han sido construidas y detalladas mediante el *Lenguaje de Definición de Datos (DDL)* (Práctica 05) y cuestiones de *Integridad* (Práctica 06) para almacenar información en forma de registros. Mediante DDL es posible definir el esquema que permitirá almacenar los datos.

En la Tabla 7.1 se puede observar una tabla que almacenará por cada registro la temperatura máxima y mínima de una ciudad.

Tabla 7.1 - Esquema de la tabla Temperatura_Ciudad.

<i>Ciudad</i>	<i>Temperatura Mínima</i>	<i>Temperatura Máxima</i>

La tabla recién creada se encuentra vacía, haciendo uso del DML es posible agregar renglones a través del SMBD. La Tabla 7.2 se muestra ya a Temperatura_Ciudad con algunos registros.

Tabla 7.2 - Tabla con dos inserciones.

<i>Ciudad</i>	<i>Temperatura Mínima</i>	<i>Temperatura Máxima</i>
Los Angeles	12°C	28°C
Washington	3°C	15°C

Para llevar a cabo la inserción de datos a través de un SMBD, SQL trabaja con palabras reservadas cuya sintaxis se muestra en la Figura 7.1.

```
INSERT INTO nombre_tabla (columna1, columna2, ..., columnaN)
VALUES (valor1, valor2, ..., valorN);
```

Figura 7.1 - Sintaxis de inserción de SQL.

En esta sintaxis encontramos:

1. **INSERT INTO**
Palabras reservadas que en conjunto, comienzan la instrucción de insertar un registro en una tabla específica.
2. **nombre_tabla**
Es el nombre de la tabla destino de la inserción, es decir, la tabla donde se insertará el nuevo registro.
3. **(**
El símbolo de apertura de paréntesis, después del nombre de la tabla, sirve para indicar que comenzará con la lista de columnas donde se insertarán los nuevos valores (el nuevo registro). En esta lista podemos definir de manera personalizada el orden en el que queremos que se inserten los datos del registro.
4. **columna1, columna2, ... , columnaN**
Es el listado de las columnas donde se insertarán los datos de un registro. El orden en el que se escriban las columnas es el orden en el que se insertarán los nuevos datos.

5. **)**
El símbolo de cierre de paréntesis, indica el término de la lista de columnas donde se insertarán los nuevos valores.
6. **VALUES**
Palabra reservada para indicarle al SMBD que la lista que sigue a continuación son los valores del nuevo registro.
7. **(**
El símbolo de apertura de paréntesis, indica la apertura de la lista de valores que se insertarán para formar el nuevo registro.
8. **valor1, valor2, ... , valorN**
Es el listado de valores que se insertarán para formar el nuevo registro. El orden en el que se escriban estos valores es el orden en el que se insertarán los nuevos datos.
9. **)**
El símbolo de cierre de paréntesis, indica el cierre de la lista de valores que se insertarán para formar el nuevo registro.
10. **;**
Indica la finalización de la instrucción.

Como ejemplo, se muestra la sintaxis de la inserción de un registro en la tabla *Temperatura_Ciudad*. Ver Figura 7.2.

```
INSERT INTO Temperatura_Ciudad (Ciudad,Temperatura_Minima,Temperatura_Maxima)
VALUES ('Los Angeles','12°C','28°C');
```

Figura 7.2 - Ejemplo de inserción en la tabla *Temperatura_Ciudad*.

Cabe mencionar que los elementos descritos en los puntos 3, 4 y 5 son opcionales, en caso de no colocarlos, el SMBD asume que se utilizarán todas las columnas de la tabla en el orden en el que se encuentran en definidos de izquierda a derecha. En el caso que se requiera insertar los datos en columnas específicas, la lista de las columnas puede variar en orden, así como en el número de columnas. Por ejemplo, si quisiéramos insertar solo el nombre de la ciudad, el código se vería como se muestra en la Figura 7.3.

```
INSERT INTO Temperatura_Ciudad (Ciudad) VALUES ('Ciudad de México');
```

Figura 7.3 - Ejemplo de inserción en la tabla *Temperatura_Ciudad* sobre un sólo atributo.

3.2. Consulta de datos

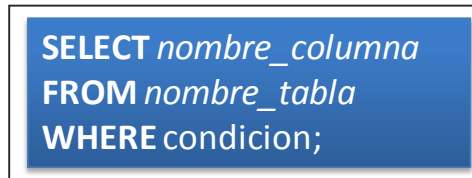
Los datos almacenados en una base de datos, no generan información por sí solos. Es por eso que se requiere consultarlos, ordenarlos, contarlos, sumarlos, etc., para generar dicha información.

Por ejemplo, retomemos la tabla de las ciudades y sus temperaturas. Si trabajáramos para un centro climatológico, podríamos querer saber qué ciudades han presentado las temperaturas más bajas para poder hacer predicciones sobre el clima.

Como otro ejemplo, supongamos que somos los dueños de una pizzería de entrega a domicilio. Dicha pizzería cuenta con una base de datos que almacena todos los pedidos y sus detalles, así como los clientes con sus teléfonos y domicilios. Para incrementar la utilidad se ha decidido invertir en promociones y publicidad.

Conocer las características de las pizzas que más se consumen y el tipo de cliente que más ordena, es primordial para ofrecer las mejores promociones e invertir correctamente en publicidad. Para lograr esto, necesitaríamos hacer una explotación específica de los datos.

SQL ofrece la capacidad de poder realizar consultas con palabras reservadas. A continuación se presenta la sintaxis básica para realizar consultas. Ver Figura 7.4.



```
SELECT nombre_columna
FROM nombre_tabla
WHERE condicion;
```

Figura 7.4 - Sintaxis de consulta de SQL.

En esta sintaxis encontramos:

1. **SELECT**
Palabra reservada para indicar que atributos se mostrarán como resultado de la consulta.
2. ***nombre_columna***
Es el nombre de la columna, o columnas separadas por comas, que queremos consultar.
3. **FROM**
Palabra reservada para indicarle al SDBD la tabla de la cual obtener la columna que necesitamos.
4. ***nombre_tabla***
Es el nombre de la tabla que contiene la columna o columnas que solicitamos.
5. **WHERE**
Palabra reservada para indicar bajo qué condición seleccionaremos los registros que queremos consultar.
6. ***condicion***
Es una característica o conjunto de características que tiene que cumplir los registros para ser seleccionados.
7. **;**
Indica la finalización de la instrucción.

La Figura 7.5 muestra un ejemplo del uso de esta estructura de consulta.

```
SELECT Ciudad, Temperatura Mínima, Temperatura Máxima
FROM Temperatura_Ciudad
WHERE Ciudad = 'Washington';
```

Figura 7.5 - Ejemplo de consulta a la tabla Temperatura_Ciudad.

El resultado de la anterior consulta se muestra en la Tabla 7.3.

Tabla 7.3. Ejemplo de consulta a la tabla Temperatura_Ciudad.

<i>Ciudad</i>	<i>Temperatura Mínima</i>	<i>Temperatura Máxima</i>
Los Angeles	12°C	28°C

3.3. Actualización de datos

Aunque el modelo de la base de datos y la creación de las tablas haya sido eficiente, es probable que a través del tiempo, se necesiten hacer cambios a los datos almacenados. Esto se debe a que la información cambia debido a múltiples razones, como por ejemplo: errores en la captura de los datos, cambios en las políticas internas de la organización dueña de la base de datos, cambios en particular como la dirección o teléfono de clientes o proveedores, entre otros.

A manera de ejemplo, en la Tabla 7.4 se muestra una tabla con cuatro registros previamente insertados.

Tabla 7.4 - Tabla de Temperatura_Ciudad.

<i>Ciudad</i>	<i>Temperatura Mínima</i>	<i>Temperatura Máxima</i>
Los Angeles	12°C	28°C
Washington	3°C	15°C
Monterrey	8°C	39°C
Rio de Janeiro	6°C	37°C

Cuando existen datos en una tabla, podemos realizar cambios en ellos utilizando lenguaje SQL. La Tabla 7.5 muestra la misma tabla con algunos cambios o actualizaciones.

Tabla 7.5 - Tabla Temperatura_Ciudad con actualizaciones.

<i>Ciudad</i>	<i>Temperatura Mínima</i>	<i>Temperatura Máxima</i>
Los Angeles	8°C	28°C
Washington	2°C	15°C
Monterrey	8°C	42°C
Rio de Janeiro	6°C	38°C

Para llevar a cabo las actualizaciones necesarias en los datos de una tabla, los SMBD trabajan con palabras reservadas de SQL, cuya sintaxis se muestra en la Figura 7.6.

```
UPDATE nombre_tabla
SET columna_1 = nuevo_valor
WHERE condición;
```

Figura 7.6. Sintaxis de actualización de SQL.

En esta sintaxis encontramos:

1. **UPDATE**
Palabra reservada para comenzar la instrucción de actualización de datos.
2. **nombre_tabla**
Es el nombre de la tabla en donde se encuentran los datos a los que se les realizarán los cambios.
3. **SET**
Palabra reservada para definir la columna o conjunto de columnas que se actualizarán.
4. **columna_1**
Es el nombre de la columna donde se encuentran los datos que se actualizarán.
5. **=**
El signo igual sirve para definir el nuevo valor que tomará la columna que se declaró anteriormente.
6. **nuevo_valor**
Es el valor que aparecerá en lugar del anterior.
7. **WHERE**
Palabra reservada para indicar bajo que condición se seleccionarán los registros donde se hará la actualización.
8. **condicion**
Es una característica o conjunto de características que tiene que cumplir los registros para ser actualizados.
9. **;**
Indica la finalización de la instrucción.

Como ejemplo, se muestra en la Figura 7.7, la sintaxis de la actualización de un campo en un registro de la tabla *Temperatura_Ciudad*.

```
UPDATE Temperatura_Ciudad
SET Temperatura_Minima = '8°C'
WHERE Ciudad = 'Los Angeles';
```

Figura 7.7 - Ejemplo de actualización en la tabla *Temperatura_Ciudad*.

3.4. Eliminación de registros.

De la misma manera que se comentó en el punto de Actualización de datos, los datos son sensibles a cambios de diversa naturaleza, uno de estos cambios puede ser también la eliminación de registros. Este proceso consiste en borrar un renglón o renglones específicos de una tabla. Es importante saber que si se requiriera eliminar un campo específico de un renglón, se deberá hacer por el método de Actualización de datos y no por Eliminación de registros.

En la Tabla 7.6 se muestra la tabla completa de ciudades y temperaturas de una base de datos. Por cuestiones de políticas es necesario eliminar la ciudad de Rio de Janeiro. Este proceso se realiza a través de la Eliminación de registros.

Tabla 7.6. Tabla con eliminación del registro para la ciudad Rio de Janeiro.

<i>Ciudad</i>	<i>Temperatura Mínima</i>	<i>Temperatura Máxima</i>
Los Angeles	8°C	28°C
Washington	2°C	15°C
Monterrey	8°C	42°C

SQL ofrece la capacidad de poder hacer estas consultas con palabras reservadas. A continuación, en la Figura 7.8, se muestra la sintaxis para realizar el proceso de eliminación.

```
DELETE FROM nombre_tabla
WHERE condicion;
```

Figura 7.8 - Sintaxis de eliminación de SQL.

En esta sintaxis podemos encontrar:

1. **DELETE FROM**
Palabras reservadas para indicarle al SDB que se hará la eliminación de un registro completo.
2. **nombre_tabla**
Es el nombre de la tabla de donde se eliminará el registro.

3. **WHERE**

Palabra reservada para indicar bajo qué condición se seleccionará el registro que se desea eliminar.

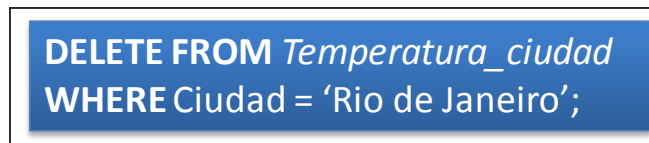
4. **condicion**

Es una característica o conjunto de características que tiene que cumplir los registros para ser eliminados.

5. **;**

Indica la finalización de la instrucción.

Como ejemplo, en la Figura 7.9, se muestra la sintaxis de la eliminación de un registro de la tabla *Temperatura_Ciudad*.

A blue rectangular box with a thin black border containing white text. The text is a SQL DELETE statement: 'DELETE FROM Temperatura_ciudad WHERE Ciudad = 'Rio de Janeiro';'. The table name 'Temperatura_ciudad' is italicized.

```
DELETE FROM Temperatura_ciudad
WHERE Ciudad = 'Rio de Janeiro';
```

Figura 7.9 - Ejemplo de eliminación de registro en la tabla *Temperatura_Ciudad*.

4. Ejercicios

(NOTA: Resuelve los siguientes ejercicios en relación al proyecto que realizarás durante el curso, en dado caso que no tengas un proyecto, utiliza la información en el apéndice NFL-ONEFA parte 07 al final de esta práctica para realizarlos)

1. Realiza al menos 5 **Inserciones para cada tabla de la base de datos**. Guarda y entrega la sintaxis que utilizaste en un archivo .sql.
2. Realiza 2 modificaciones de los datos insertados para cada tabla. Guarda y entrega la sintaxis que utilizaste en un archivo .sql.
3. Realiza 2 consultas por tabla. Guarda y entrega la sintaxis que utilizaste en un archivo .sql.
4. Realiza 2 eliminaciones de cada tabla. Guarda y entrega la sintaxis que utilizaste en un archivo .sql.

Entregables requeridos para prácticas subsecuentes:

- Inserciones para cada tabla de la base de datos

5. Apéndice NFL-ONEFA parte 07

Crea la base de datos NFL-ONEFA (si no la tienes) utilizando el código que se encuentra al término de estas instrucciones. Una vez creadas las tablas realiza los ejercicios de la sección 4 de ésta práctica.

```
-- CREATE DATABASE NFL-ONEFA
```

```
-- DROP DATABASE NFL-ONEFA
```

```
CREATE TABLE jugador (  
    id_jugador INT NOT NULL CHECK(id_jugador > 0),  
    apellido_jugador VARCHAR(20) NOT NULL,  
    nombre_jugador VARCHAR(20) NOT NULL,  
    fecha_nacimiento DATE NOT NULL,  
    universidad VARCHAR(30) NOT NULL,  
    PRIMARY KEY(id_jugador)  
);  
  
CREATE TABLE linea_ofensiva (  
    id_jugador INT NOT NULL CHECK(id_jugador > 0),  
    posicion VARCHAR(2) NOT NULL CHECK(posicion IN ('C', 'LS', 'OG', 'OL', 'OT')),  
    pancakes INT NOT NULL CHECK(pancakes >= 0),  
    PRIMARY KEY(id_jugador),  
    FOREIGN KEY(id_jugador) REFERENCES jugador(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE  
);  
  
CREATE TABLE defensiva (  
    id_jugador INT NOT NULL CHECK(id_jugador > 0),  
    posicion VARCHAR(2) NOT NULL CHECK(posicion IN ('DB', 'DE', 'DL', 'DT', 'LB')),  
    tackles INT NOT NULL CHECK(tackles >= 0),  
    sacks INT NOT NULL CHECK(sacks >= 0 AND sacks <= tackles),  
    ff INT NOT NULL CHECK(ff >= 0),  
    ints INT NOT NULL CHECK(ints >= 0),  
    PRIMARY KEY(id_jugador),  
    FOREIGN KEY(id_jugador) REFERENCES jugador(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE  
);  
  
CREATE TABLE kicker (  
    id_jugador INT NOT NULL CHECK(id_jugador > 0),  
    posicion VARCHAR(2) NOT NULL CHECK(posicion IN ('K')),  
    fga INT NOT NULL CHECK(fga >= 0),  
    fgm INT NOT NULL CHECK(fgm >= 0 AND fgm <= fga),  
    blocked INT NOT NULL CHECK(blocked >= 0 AND blocked <= (fga - fgm)),  
    longest INT NOT NULL CHECK(longest BETWEEN 0 AND 117),  
    PRIMARY KEY(id_jugador),  
    FOREIGN KEY(id_jugador) REFERENCES jugador(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE  
);  
  
CREATE TABLE punter (  
    id_jugador INT NOT NULL CHECK(id_jugador > 0),  
    posicion VARCHAR(2) NOT NULL CHECK(posicion IN ('P')),
```

```

punts INT NOT NULL CHECK(punts >= 0),
net INT NOT NULL CHECK(net >= 0),
longest INT NOT NULL CHECK(longest BETWEEN 0 AND 100),
in20 INT NOT NULL CHECK(in20 >=0 AND in20 <= punts),
PRIMARY KEY(id_jugador),
FOREIGN KEY(id_jugador) REFERENCES jugador(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE quarterback (
    id_jugador INT NOT NULL CHECK(id_jugador > 0),
    posicion VARCHAR(2) NOT NULL CHECK(posicion IN ('QB')),
    tds INT NOT NULL CHECK(tds >= 0 AND tds <= att),
    ints INT NOT NULL CHECK(ints >= 0 AND ints <= att),
    yds INT NOT NULL,
    att INT NOT NULL CHECK(att >= 0),
    comp INT NOT NULL CHECK(comp >= 0 AND comp <= att),
    PRIMARY KEY(id_jugador),
    FOREIGN KEY(id_jugador) REFERENCES jugador(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE runner (
    id_jugador INT NOT NULL CHECK(id_jugador > 0),
    posicion VARCHAR(2) NOT NULL CHECK(posicion IN ('RB', 'FB')),
    carries INT NOT NULL CHECK(carries >= 0),
    fumbles INT NOT NULL CHECK(fumbles >= 0 AND fumbles <= carries),
    yds INT NOT NULL,
    tds INT NOT NULL CHECK(tds >= 0 AND tds <= carries),
    PRIMARY KEY(id_jugador),
    FOREIGN KEY(id_jugador) REFERENCES jugador(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE receiver (
    id_jugador INT NOT NULL CHECK(id_jugador > 0),
    posicion VARCHAR(2) NOT NULL CHECK(posicion IN ('TE', 'WR')),
    rec INT NOT NULL CHECK(rec >= 0),
    yds INT NOT NULL,
    fumbles INT NOT NULL CHECK(fumbles >= 0 AND fumbles <= rec),
    tds INT NOT NULL CHECK(tds >= 0 AND tds <= rec),
    PRIMARY KEY(id_jugador),
    FOREIGN KEY(id_jugador) REFERENCES jugador(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE equipo (
    id_equipo VARCHAR(3) NOT NULL,
    nombre_equipo VARCHAR(20) NOT NULL,
    nickname_equipo VARCHAR(20) NOT NULL,
    division VARCHAR(10) NOT NULL CHECK(division IN ('Norte', 'Sur', 'Este', 'Oeste')),
    conferencia VARCHAR(10) NOT NULL CHECK(conferencia IN ('Americana', 'Nacional', 'Centro', '12 Grandes')),
    record_G INT NOT NULL CHECK(record_G BETWEEN 0 AND 16),
    record_P INT NOT NULL CHECK(record_P BETWEEN 0 AND 16),
    record_E INT NOT NULL CHECK(record_E BETWEEN 0 AND 16),

```

```

    anyo_fund INT NOT NULL CHECK(anyo_fund BETWEEN 1900 AND 2008),
    CHECK((record_G + record_E + record_P) = 16),
    PRIMARY KEY(id_equipo)
);

```

```

CREATE TABLE entrenador (
    id_entrenador INT NOT NULL CHECK(id_entrenador > 0),
    apellido_entrenador VARCHAR(20) NOT NULL,
    nombre_entrenador VARCHAR(20) NOT NULL,
    cargo VARCHAR(2) NOT NULL CHECK(cargo IN ('HC', 'OC', 'DC', 'SC')),
    PRIMARY KEY(id_entrenador)
);

```

```

CREATE TABLE ciudad (
    id_ciudad INT NOT NULL CHECK(id_ciudad > 0),
    nombre_ciudad VARCHAR(20) NOT NULL,
    estado_ciudad VARCHAR(2) NOT NULL,
    clima VARCHAR(20) NOT NULL CHECK(clima IN ('Templado', 'Frio', 'Caluroso')),
    habitantes BIGINT NOT NULL CHECK(habitantes BETWEEN 0 AND 999999999),
    indice_crecimiento INT NOT NULL,
    region VARCHAR(30) NOT NULL CHECK(region IN ('Suroeste', 'Sureste', 'Medioeste', 'Mediooeste', 'Noreste',
'Noroeste', 'Central')),
    ingreso_promedio INT NOT NULL,
    PRIMARY KEY(id_ciudad)
);

```

```

CREATE TABLE estadio (
    id_estadio INT NOT NULL CHECK(id_estadio > 0),
    nombre_estadio VARCHAR(30) NOT NULL,
    capacidad INT NOT NULL CHECK(capacidad > 0),
    grama VARCHAR(15) NOT NULL CHECK(grama IN ('Artificial', 'Natural')),
    condicion VARCHAR(10) NOT NULL CHECK(condicion IN ('Excelente', 'Buena', 'Regular', 'Mala', 'Pesima')),
    anyo_construccion INT NOT NULL CHECK(anyo_construccion BETWEEN 1800 AND 2008),
    id_ciudad INT NOT NULL,
    PRIMARY KEY(id_estadio),
    FOREIGN KEY(id_ciudad) REFERENCES ciudad(id_ciudad) ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE jugador_equipo (
    id_jugador INT NOT NULL,
    id_equipo VARCHAR(3) NOT NULL,
    numero INT NOT NULL CHECK(numero BETWEEN 1 AND 99),
    juegos_jugados INT NOT NULL CHECK(juegos_jugados BETWEEN 0 AND 16),
    juegos_iniciados INT NOT NULL CHECK(juegos_iniciados BETWEEN 0 AND 16),
    anyo_adquirido INT NOT NULL CHECK(anyo_adquirido BETWEEN 1900 AND 2008),
    CHECK(juegos_jugados >= juegos_iniciados),
    PRIMARY KEY(id_jugador, id_equipo, anyo_adquirido),
    FOREIGN KEY(id_jugador) REFERENCES jugador(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(id_equipo) REFERENCES equipo(id_equipo) ON DELETE CASCADE ON UPDATE CASCADE
);

```

```
CREATE TABLE equipo_ciudad (  
    id_equipo VARCHAR(3) NOT NULL,  
    id_ciudad INT NOT NULL,  
    apoyo_ciudad INT NOT NULL CHECK(apoyo_ciudad BETWEEN 0 AND 5),  
    PRIMARY KEY(id_equipo),  
    FOREIGN KEY(id_ciudad) REFERENCES ciudad(id_ciudad) ON DELETE CASCADE ON UPDATE CASCADE,  
    FOREIGN KEY(id_equipo) REFERENCES equipo(id_equipo) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE equipo_estadio (  
    id_equipo VARCHAR(3) NOT NULL,  
    id_estadio INT NOT NULL,  
    proyecto_nuevo VARCHAR(30),  
    PRIMARY KEY(id_equipo),  
    FOREIGN KEY(id_estadio) REFERENCES estadio(id_estadio) ON DELETE CASCADE ON UPDATE CASCADE,  
    FOREIGN KEY(id_equipo) REFERENCES equipo(id_equipo) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE equipo_entrenador (  
    id_equipo VARCHAR(3) NOT NULL,  
    id_entrenador INT NOT NULL,  
    anyo_contratado INT NOT NULL CHECK(anyo_contratado BETWEEN 1900 AND 2008),  
    PRIMARY KEY(id_entrenador, anyo_contratado),  
    FOREIGN KEY(id_entrenador) REFERENCES entrenador(id_entrenador) ON DELETE CASCADE ON UPDATE  
CASCADE,  
    FOREIGN KEY(id_equipo) REFERENCES equipo(id_equipo) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE partido (  
    id_partido INT NOT NULL,  
    id_equipo_V VARCHAR(3) NOT NULL,  
    id_equipo_L VARCHAR(3) NOT NULL,  
    marcador_V INT NOT NULL CHECK(marcador_V BETWEEN 0 AND 100),  
    marcador_L INT NOT NULL CHECK(marcador_L BETWEEN 0 AND 100),  
    semana INT NOT NULL CHECK(semana BETWEEN 1 AND 21),  
    PRIMARY KEY(id_partido),  
    FOREIGN KEY(id_equipo_V) REFERENCES equipo(id_equipo) ON DELETE CASCADE ON UPDATE CASCADE,  
    FOREIGN KEY(id_equipo_L) REFERENCES equipo(id_equipo) ON DELETE CASCADE ON UPDATE CASCADE  
);
```