AWAKELAB

Emprendimiento digital con tecnologías Web

Aprendizaje Esperado

2. Construir sentencias utilizando el lenguaje de manipulación de datos DML para la modificación de los datos existentes en una base de datos a partir de un modelo de datos existente.

3.3.- Sentencias para la manipulación de datos

3.3.1.- Data Manipulation Language

El lenguaje de manipulación (DML) se encarga del manejo de los datos de una base de datos como las que hemos visto hasta el momento. Algunas declaraciones DML se ven como sentencias comunes del lenguaje natural hablado o escrito y son fáciles de entender. Sin embargo, dado el alto grado de detalle de control que permite SQL sobre los datos, otras instrucciones pueden llegar a ser extremadamente complejas.

Si una instrucción DML incluye múltiples expresiones, cláusulas, predicados (Que explicaremos en este apartado) o subconsultas, entender qué es lo que ésta realmente hace puede ser una hazaña. Sin embargo, las sentencias SQL de mayor complejidad se pueden llegar a entender dividiendo sus componentes y analizando una parte a la vez.

Las instrucciones DML que encontraremos en SQL son INSERT, UPDATE, DELETE y SELECT. Estas instrucciones pueden constar de una variedad de partes, incluidas varias cláusulas. Cada cláusula puede incorporar expresiones de valor, conectores lógicos, predicados, funciones de agregación y subconsultas. Utilizando estas cláusulas en las instrucciones SQL se puede realizar una discriminación fina de registros de la base de datos. Revisaremos aquí algunos de estos elementos:

Expresiones de valor

Puede utilizar expresiones de valor para combinar dos o más valores. Varias clases de valor existen expresiones, correspondientes a los diferentes tipos de datos:

```
Numérico, String, Datetime, Intervalo, Booleano, Definido por el Usuario, Fila, Colección
```

Los tipos booleano, definido por el usuario, de fila y de colección se introdujeron con SQL: 1999. Es posible que algunas implementaciones aún no las admitan todas. Si deseamos utilizar estos datos tipos, debemos asegurarnos que la implementación que deseamos utilizar las incluya.

Expresiones numéricas

Para combinar valores numéricos, usamos la suma (+), resta (-), multiplicación (*), y operadores de división (/). Las siguientes líneas son ejemplos de expresiones de valor numérico:

```
12 - 7
15/3 - 4
6 * (8 + 2)
```

Los valores de estos ejemplos son literales numéricos. Estos valores también pueden ser de nombres de columna, parámetros, variables o subconsultas, siempre que estas se evalúen a un valor numérico. A continuación, se muestran algunos ejemplos:

```
SUBTOTAL + IMPUESTO + ENVÍO
6 * KILOMETROS/HORAS
meses/12
```

Los dos puntos en el último ejemplo indican que el siguiente término (meses) es un

parámetro o una variable del lenguaje principal.

Expresiones de string

Las expresiones de valor de string pueden incluir el operador de concatenación (||). Podemos utilizar concatenación para unir dos strings, como se muestra a continuación:

Expresión	Resultado
concat('inteligencia ' ,'militar')	'inteligencia militar'
concat(CIUDAD, ' ', PROVINCIA, ' ', REGION)	String con ciudad, provincia y región; todos separados por un espacio

Algunas implementaciones de SQL usan '+' o '||' como operador de concatenación. Consulte la documentación del motor de base de datos utilizado, para ver qué operador utiliza. Algunas implementaciones pueden incluir operadores de string distintos de la concatenación, pero el estándar ISO SQL no admite dichos operadores. La concatenación se aplica a strings binarios, así como a strings de texto.

Expresiones de valor de Datetime e intervalos

Las expresiones de valor Datetime manejan fechas y tiempos. Datos de tipo DATE, TIME, TIMESTAMP y INTERVAL pueden aparecer en expresiones de valor Datetime. El resultado de su evaluación es siempre otro Datetime. Podemos sumar o restar un intervalo de un Datetime y especificar información de zona geográfica.

Un ejemplo de este tipo de expresiones es:

```
now() + INTERVAL '7' DAY
```

lo que podría ser utilizado por una aplicación, por ejemplo, para determinar que debe ejecutar cierta acción luego de pasados 7 días después de la fecha actual.

Las expresiones de valor de intervalo manejan diferencias (Tiempo transcurrido) entre un Datetime y otro. Hay dos tipos de intervalos: año-mes y día-tiempo. Se pueden mezclar ambos en una expresión.

Como ejemplo de un intervalo, supongamos que alguien devuelve en una biblioteca un libro posteriormente a la fecha de vencimiento. Utilizando este tipo de expresiones de intervalo como se muestra a continuación podríamos calcular cuántos días de atraso posee la acción de devolución del libro y calcular la penalidad correspondiente.

```
(FechaDevolucion – FechaVencimiento) DAY
```

El intervalo debe ser año-mes o día-tiempo. Por lo tanto, es necesario especificar cuál de estos desea utilizarse. En el último ejemplo hemos indicado DAY.

Expresiones de valor booleano

Una expresión de valor booleano verifica la verdad de un predicado. Los siguientes son ejemplos de este tipo de expresiones para una columna Clase y un valor SENIOR:

```
(Clase = SENIOR) IS TRUE
NOT (Clase = SENIOR) IS TRUE
(Clase = SENIOR) IS FALSE
(Clase = SENIOR) IS UNKNOWN
```

Expresiones de valores definidos por el usuario

Los tipos definidos por el usuario (UDT) representan otro ejemplo de características que llegaron a SQL: 1999 que provienen del mundo de la programación orientada a objetos. Como programadores SQL, no estamos

restringidos sólo a los tipos de datos definidos en la especificación SQL. Podemos definir nuestros propios tipos de datos, utilizando los principios de tipos de datos abstractos (ADT) que se encuentran en lenguajes de programación orientados a objetos como C ++.

Las expresiones que incorporan elementos de datos de este tipo definido por el usuario deben evaluar a un elemento del mismo tipo.

Expresiones de valor de fila

Una expresión de valor de fila especifica el valor de una fila de datos. El valor de fila puede consistir de una, dos o más expresiones. Por ejemplo:

```
('Albert Einstein', 'Professor', 1918)
```

Donde esta fila podría pertenecer a una tabla Facultad, donde se muestre el nombre, el rango y el año de contratación de un colaborador.

Expresiones de valor de colección

Una expresión de valor de colección es evaluada a un arreglo.

Expresiones de valor de referencia

Este tipo de expresiones son evaluadas a un valor que referencia a otro componente de la base de datos, tal como una columna.

Predicados

Los predicados son los equivalentes SQL de las proposiciones lógicas. Por ejemplo:

```
"El empleado es senior"
```

En una tabla que contiene información sobre los empleados, el dominio de la columna CLASE puede ser SENIOR, MEDIO, JUNIOR o NULL. Podemos usar el predicado CLASE = SENIOR para filtrar filas en las que el predicado es False, reteniendo sólo aquellos para los que el predicado es True. A veces, el valor de un predicado en una fila es Desconocido (NULL). En esos casos, podemos optar por descartar la fila o retenerla.

CLASE = SENIOR es un ejemplo de predicado de comparación. SQL tiene seis operadores de comparación. Un predicado de comparación simple usa uno de estos operadores. A continuación, se muestran los predicados de comparación y algunos ejemplos.

Operadores y predicados de comparación:

Operador	Comparación	Expresión
=	Igual que	Clase = SENIOR
<>	Distinto que	Clase <> SENIOR
<	Menor que	Clase < SENIOR
>	Mayor que	Clase > SENIOR
<=	Menor o igual que	Clase <= SENIOR
>=	Mayor o igual que	Clase >= SENIOR

En el ejemplo anterior, solo las dos primeras filas (Clase = SENIOR y Clase <> SENIOR) tienen sentido. Las demás categorías de valores no tendrán sentido en este caso no numérico pues serán ordenadas alfabéticamente en forma ascendente. Esta interpretación en casos de variables categóricas no necesariamente será lo deseado.

Conectores lógicos

Los conectores lógicos le permiten construir predicados complejos a partir de otros simples. Por ejemplo, supongamos que deseamos identificar a los empleados más exitosos de una base de datos de una compañía. Dos proposiciones que podrían identificar a estos empleados pueden leerse de la siguiente manera:

"El empleado es senior"
"Los años de experiencia del empleado son menos de 2 años".

Podemos utilizar el conector lógico AND para crear un predicado compuesto que identifica los registros de empleados que deseamos, como en el siguiente ejemplo:

```
Clase = SENIOR AND Experiencia < 2
```

Si usamos el conector AND, ambos predicados deben ser verdaderos para que el predicado compuesto sea verdadero. Podemos utilizar el conector OR cuando deseemos que el predicado compuesto se evalúe como verdadero si cualquiera de los componentes del predicado es verdadero. NOT es el tercer conector lógico. Estrictamente hablando, NOT no conecta dos predicados, sino que invierte el valor de verdad del predicado único al que se aplica. Por ejemplo, la siguiente expresión:

```
NOT (Clase = SENIOR)
```

Esta expresión es verdadera sólo si Clase no es igual a SENIOR.

Funciones de conjunto

A veces, la información que desea extraer de una tabla no se relaciona con filas individuales sino más bien a conjuntos de filas. SQL proporciona funciones de conjunto (o agregadas) para hacer frente a tales situaciones. Estas funciones son COUNT, MAX, MIN, SUM y AVG. Cada función realiza una acción que extrae datos de un conjunto de filas en lugar de una sola fila.

COUNT: La función COUNT devuelve el número de filas en la tabla especificada. Para contar el número de empleados senior adelantados del ejemplo anterior, usaríamos la siguiente sentencia:

```
SELECT COUNT (*)
```

```
FROM Empleado
WHERE Clase = SENIOR AND Experiencia < 2;
```

MAX: La función MAX devuelve el valor máximo que ocurre en una columna específica. Supongamos que deseamos encontrar el empleado de mayor edad de la compañía. La siguiente sentencia nos entregaría la fila apropiada:

```
SELECT Nombre, Apellido, Edad
FROM Empleado
WHERE Edad = (SELECT MAX(Edad) FROM Empleado);
```

Esta sentencia devuelve todos los empleados cuyas edades son iguales a la edad máxima. Es decir, si la edad del empleado mayor es 40, esta sentencia devuelve los nombres, apellidos y edades de todos los empleados que tengan 40 años.

Esta consulta utiliza una subconsulta. La subconsulta SELECT MAX(Edad) FROM Empleado es insertada dentro de la consulta principal.

MIN: La función MIN funciona igual que MAX excepto que MIN busca el valor mínimo en la columna especificada en lugar del máximo. Para encontrar al empleado más joven inscrito, podemos utilizar la siguiente consulta:

```
SELECT Nombre, Apellido, Edad
FROM Empleado
WHERE Edad = (SELECT MIN(Edad) FROM Empleado);
```

Esta consulta devuelve todos los empleados cuya edad es igual a la edad del empleado más joven.

SUM: La función SUM suma los valores en una columna especificada. La columna debe ser un dato de tipo numérico, y el valor de la suma debe pertenecer al rango de ese tipo. Por lo tanto, si la columna es de tipo

SMALLINT, la suma no debe ser mayor que el límite superior del tipo de datos SMALLINT. Por ejemplo, si tuviéramos una base de datos con una tabla Factura que contuviera un registro de todas las ventas en la columna TotalVenta. Para encontrar el valor total de todas las ventas registradas, utilizaríamos la función SUM de la siguiente manera:

```
SELECT SUM(TotalVenta) FROM Factura;
```

AVG: La función AVG devuelve el promedio de todos los valores en la columna especificada. Al igual que la función SUM, AVG se aplica solo a columnas con tipo de datos numéricos. Para encontrar el valor de la venta promedio, considerando todas las transacciones en la base de datos, usaríamos la función AVG de esta manera:

```
SELECT AVG(TotalVenta) FROM Factura;
```

Los valores nulos no tienen valor, por lo que si alguna de las filas de la columna TotalVenta contiene valores nulos, esas filas se ignorarán en el cálculo del valor de la venta promedio.

3.3.2. Actualizando la información de una tabla

Para modificar algunos datos de un registro, usamos el comando UPDATE. Este comando no puede agregar o eliminar registros (Esas responsabilidades se delegan a otros comandos). Sin embargo, si existe un registro, puede modificar sus datos incluso afectando a múltiples campos de una vez y aplicando condiciones. La sintaxis general de una instrucción UPDATE es:

```
UPDATE nombre_tabla SET
<columna1> = <valor>,
<columna2> = <valor>,
<columna3> = <valor>
. . .
```

```
WHERE <condicion>;
```

Si volvemos a nuestro ejemplo de mibasededatos y la tabla profesor, podemos probar esta manera de actualizar datos. Intentemos darle un aumento de 200.000 en su sueldo a todos los profesores que ganen menos de 1.000.000. Utilizando la sintaxis anterior aplicada a este ejemplo tendremos:

```
UPDATE profesor SET
sueldo = sueldo + 200000
WHERE sueldo < 1000000;
```

Lo que dará origen a la siguiente actualización de la tabla profesores, con un aumento en su sueldo a todos los que recibían menos de 1.000.000:

Para el comando UPDATE también es posible elegir múltiples columnas para hacer actualizaciones. Por ejemplo, si deseamos llenar los datos que se encuentran vacíos en el registro correspondiente al profesor Romilio Recabarren, podemos ejecutar la siguiente instrucción, con la que modificaremos los valores NULL de fecha_de_contratacion y sueldo:

```
UPDATE profesor SET
fecha_de_contratacion='2011-10-22',
sueldo=500000
WHERE profesor_id=10;
```

3.3.3.- Borrando información de una tabla

Podemos utilizar el comando DELETE para eliminar registros de una tabla. Esto significa que podemos elegir qué registros deseamos eliminar en función de una condición o eliminar todos los registros, pero no podemos eliminar ciertos campos de un registro usando esta declaración. La sintaxis general de la instrucción DELETE es dado a continuación:

```
DELETE FROM nombre_tabla
WHERE <condicion>;
```

Si bien poner una cláusula condicional en DELETE es opcional, es casi siempre usado, simplemente porque no usarlo haría que todos los registros de una tabla se eliminen, lo que rara vez es una necesidad real.

Volviendo a nuestro ejemplo, supongamos que los profesores de mayor y menor sueldo serán reasignados a otras regiones y deben ser retirados de esta base de datos. En este caso necesitaremos una instrucción DELETE con una condición que además nos permitirá aplicar algo de lo aprendido recientemente en relación a subconsultas y a funciones de conjuntos. Para lograr lo requerido tendremos que ejecutar:

```
DELETE FROM profesor
WHERE
sueldo = (SELECT MAX(sueldo) FROM profesor)
OR
sueldo = (SELECT MIN(sueldo) FROM profesor);
```

3.3.4.- Ingresando información a una tabla

Cada tabla de la base de datos comienza vacía. Después de crear una tabla, ésta no es más que una estructura que no contiene datos. Para que la tabla sea útil, debemos ingresar algunos datos en ella. Los datos pueden estar en una de las siguientes formas:

- **Aún sin formato digital:** Si los datos aún no están en formato digital, alguien probablemente tendrá que ingresarlos manualmente, registro por registro.
- **En algún tipo de formato digital:** Si los datos ya están en formato digital, pero tal vez no en el formato de las tablas de la base de datos que estamos utilizando, tendremos que traducirlos apropiadamente y luego insertarlos en la base de datos.

- **En el formato digital correcto:** Si los datos ya están en formato digital y en el formato correcto, éstos están listos para ser transferidos a la nueva base de datos.

A continuación, abordamos la inserción de datos a una tabla para los tres casos mencionados. Dependiendo del formato en que se encuentren los datos, es posible que puedan ser transferidos a la base de datos en una sola operación, o puede que necesiten ser ingresados un registro a la vez. Cada registro de datos que ingresamos corresponde a una fila en una tabla de la base de datos.

Agregando datos fila por fila

La mayoría de los Sistemas de Gestión de Bases de Datos (DBMS – Database Management System), tales como PostgreSQL, MySQL, SQL Server, SQLite, Oracle; poseen herramientas relacionadas que facilitan la entrada de datos basada en una interfaz gráfica, para importación de datos desde una fuente externa, como archivos, o a través de ingreso manual tipo formulario. El operador puede ingresar datos a través de esta interfaz.

En términos de instrucciones SQL, el ingreso de una sola fila en una tabla de base de datos, el comando INSERT usa la siguiente sintaxis:

```
INSERT INTO nombre_table [(columna_1, columna_2, ..., columna_n)]
VALUES (valor_1, valor_2, ..., valor_n) ;
```

Se indica con paréntesis cuadrados [ ] que los nombres de columnas son opcionales. El orden predeterminado de la lista de columnas es el orden de las columnas de la tabla. Si ponemos los VALORES en el mismo orden que las columnas de la tabla, estos elementos serán ingresados en las columnas correctas, ya sea que especifique esas columnas explícitamente o no. Si deseamos especificar los VALORES en un orden diferente al orden de las columnas en la tabla, debemos enumerar los nombres de las columnas en el mismo orden que la lista de valores que incluyamos en VALUES.

Para ingresar un registro en la tabla profesor, que hemos utilizado como ejemplo en secciones anteriores, usamos la siguiente sintaxis (Que corresponde a uno de los registros mostrados en la tabla profesor anteriormente):

```
INSERT INTO profesor
(nombre, apellido, escuela, fecha_de_contratacion, sueldo)
VALUES
('Caupolicán', 'Catrileo', 'Santiago de la extremadura', '2000-10-26', 780000 );
```

La columna profesor_id no se incluye en esta sintaxis pues al ser la llave primaria y única, es generada automáticamente por el motor de base de datos. El resto de los valores corresponden a los de las columnas que hemos indicado explícitamente.

Agregando datos a un subconjunto de columnas

A veces, se desea materializar la existencia de un registro incluso si no tenemos aún todos los valores para el mismo. En una tabla de una base de datos, podemos insertar una fila sin completar los datos de todas las columnas. Si queremos la tabla en la primera forma normal, debemos insertar suficientes datos para distinguir la nueva fila de todas las demás filas de la tabla). Insertar la llave primaria de la nueva fila es suficiente para este propósito. Es posible que la tabla posea restricciones para no aceptar valores nulos de algunas columnas y en ese caso deberemos incorporarlas. Las columnas no ingresadas, y que así lo permitan poseerán valores nulos.

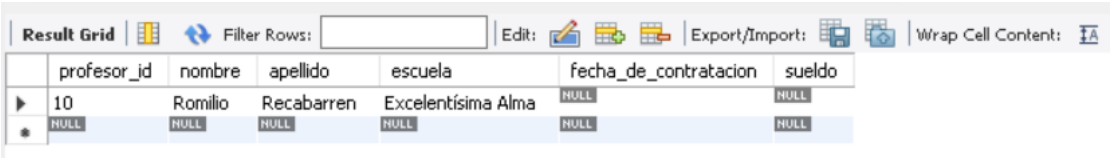
El siguiente ejemplo muestra una entrada de fila parcial de este tipo:

```
INSERT INTO profesor (nombre, apellido, escuela)
VALUES ('Romilio', 'Recabarren', 'Excelentísima Alma');
```

De manera que, si hacemos una consulta para obtener los registros que posean, por ejemplo, sueldo con valor NULL,

```
SELECT * FROM profesor WHERE sueldo IS NULL;
```

obtendremos lo siguiente:



donde los valores nulos (NULL) se muestran como valores vacíos.

Agregando bloques de filas a una tabla

La carga de una tabla de base de datos a una fila a la vez mediante instrucciones INSERT puede ser tedioso. Claramente, si existe la forma de ingresar los datos automáticamente, esto será una mejor opción que tener a una persona sentada en un teclado ingresando datos. La entrada automática de datos es factible, por ejemplo, si los datos existen en forma electrónica, pues alguien ya ha introducido los datos manualmente o se han recopilado de alguna forma automatizada. Si es así, no hay razón para repetir el proceso. Transferir datos de un archivo de datos a otro es una tarea que una máquina puede realizar con una mínima participación humana. Si conocemos las características de los datos de origen y la forma deseada de la tabla de destino, una máquina puede realizar la transferencia de datos automáticamente.

Desde un archivo de datos externo:

Suponga que estamos creando una base de datos para una nueva aplicación. Algunos datos que necesitamos ya existen en un archivo de computador. El archivo puede ser un de texto plano o una tabla en una base de datos creada por un DBMS diferente al que estamos utilizando. Los datos pueden estar en ASCII o en alguna otra codificación determinada. ¿Qué debemos hacer?

Lo más favorable es que los datos que deseamos ingresar estén en un formato comúnmente utilizado. Si los datos están en un formato popular, tendremos muchas posibilidades de encontrar una aplicación de conversión de formato que pueda traducirlos a uno o más formatos que nos sean de utilidad. Nuestro entorno de desarrollo probablemente pueda importar al menos uno de estos formatos. Si tenemos mucha suerte, nuestro entorno de desarrollo podrá manejar el formato de datos original directamente. Los formatos de SQL Server, MySQL y Postgres son algunos de los más utilizados. Si el formato original de los datos es menos común, es posible que debamos someterlo a una conversión.

Si los datos están en un formato antiguo, propietario u obsoleto, como último recurso, podemos contratar un servicio profesional de traducción de datos. Estas empresas se especializan en la traducción y transferencia de datos informáticos de un formato a otro. Poseen experiencia en cientos de formatos, la mayoría de los cuales nadie ha oído hablar y nos podrán entregar los datos en el formato que especifiquemos.

Desde otras tablas:

Podemos insertar nuevos registros en una nueva tabla desde otra usando una combinación de instrucciones INSERT y SELECT. Dado que una consulta SELECT nos devolverá algunos registros, combinándola con un INSERT, éste ingresará estos registros en la nueva tabla. Podemos incluso usar un condicional WHERE para limitar o filtrar los registros que deseamos ingresar en la nueva tabla.

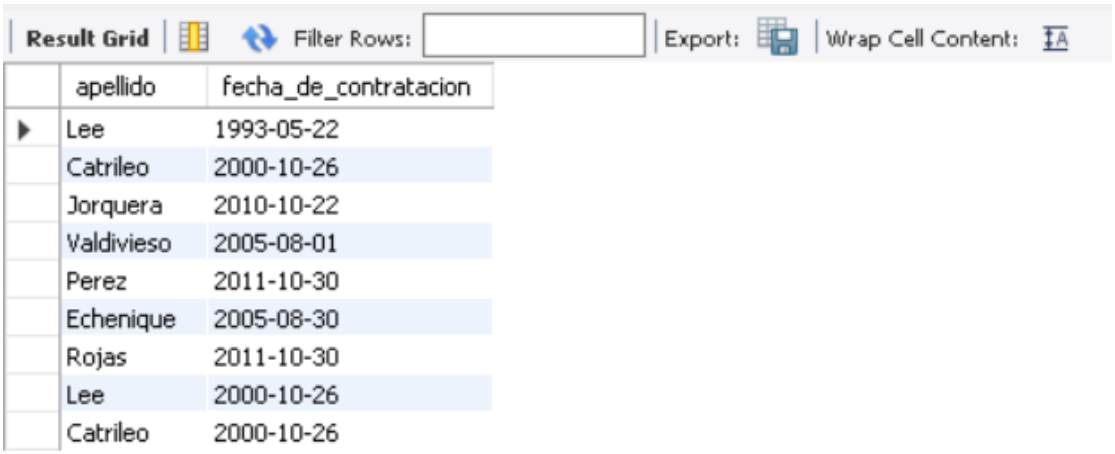
Ahora crearemos una nueva tabla llamada **ex-profesor**, que tendrá solo dos campos: **apellido** y **fecha_de_contratacion**. En esta tabla insertaremos filas de la tabla profesor que tienen valor no nulo en el campo **sueldo**.

```
CREATE TABLE ex_profesor
(
   apellido VARCHAR(50),
   fecha_de_contratacion DATE
);

INSERT INTO ex_profesor
```

```
SELECT
   apellido,
   fecha_de_contratacion
FROM profesor
WHERE sueldo IS NOT NULL;
```

Con lo que, al revisar la tabla con la siguiente consulta obtenemos el siguiente resultado:

```
SELECT * FROM ex_profesor;
```



apellido	fecha_de_contratacion
Lee	1993-05-22
Catrileo	2000-10-26
Jorquera	2010-10-22
Valdivieso	2005-08-01
Perez	2011-10-30
Echenique	2005-08-30
Rojas	2011-10-30
Lee	2000-10-26
Catrileo	2000-10-26

En este resultado vemos que se han transferido todos los valores de **apellido** y **fecha_de_contratacion** de registros de la tabla profesor, excepto el correspondiente a **profesor_id=10** que contiene un valor **NULL** en la columna **sueldo**.

Debemos mencionar que los datos insertados deben adherir a las restricciones que posee la nueva tabla desde su creación, por ejemplo, si un campo está especificado como único, debemos insertar datos que cumplan con esa condición para que nuestra operación sea exitosa.

3.3.5.- Utilización de secuencias para asignar identificadores

En MySQL, para tener una columna que vaya aumentando su valor registro a registro, se debe usar el concepto del “auto incremento” (AUTO_INCREMENT). El incremento automático permite generar automáticamente un número único cuando se inserta un nuevo registro en una tabla. A menudo, este es el campo de clave principal que nos gustaría que se creará automáticamente cada vez que se inserta un nuevo registro.

La siguiente instrucción SQL define la columna "profesor_id" como un campo de clave primaria de incremento automático en la tabla "profesor":

```
CREATE TABLE profesor (
   profesor_id int primary key auto_increment,
   nombre varchar(25),
   apellido varchar(50),
   escuela varchar(50),
   fecha_de_contratacion date,
   sueldo int
);
```

MySQL usa la palabra clave AUTO_INCREMENT para realizar una función de incremento automático.

De forma predeterminada, el valor inicial de AUTO_INCREMENT es 1 y se incrementará en 1 para cada nuevo registro.

Para permitir que la secuencia AUTO_INCREMENT comience con otro valor, use la siguiente instrucción SQL:

```
ALTER TABLE profesor AUTO_INCREMENT=20;
```

Para insertar un nuevo registro en la tabla "profesor", no tendremos que especificar un valor para la columna "profesor_id" (se agregará un valor único automáticamente):

```

```

```
INSERT INTO profesor (nombre, apellido, escuela)
VALUES ('Alfonsina','Aranda','Escuela Rural de Petorca');
```

Finalmente, el resultado obtenido posterior a la consulta anterior será el siguiente. Como se puede apreciar, el registro insertado toma el valor indicado en los pasos anteriores.


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

profesor_id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
1	Bruce	Lee	Republica Popular China	1993-05-22	780945
2	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
3	Jericho	Jorquera	A-18 Abrazo de Maipu	2010-10-22	67500
4	Juan Alberto	Valdivieso	Sagrada Concepcion	2005-08-01	3400000
5	Juanita	Perez	Gabriela Mistral	2011-10-30	234000
6	Nicolas	Echenique	Bendito Corazón de María	2005-08-30	8900000
7	Pablo	Rojas	E-34	2011-10-30	300000
8	Wong	Lee	Santiago de la extremadura	2000-10-26	780000
9	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
10	Romilio	Recabarren	Excelentísima Alma	NULL	NULL
20	Alfonsina	Aranda	Escuela Rural de Petorca	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL

3.3.6.- Insertar, actualizar y borrar datos con integridad referencial

Una llave foránea (FOREIGN KEY) es una columna o un grupo de columnas en una tabla que hacen referencia a la llave primaria (PRIMARY KEY) de otra tabla.
La tabla que contiene la FOREIGN KEY se denomina tabla referendo, hija o secundaria. Y la tabla a la que hace referencia la FOREIGN KEY se denomina tabla referenciada, madre o principal.

Una tabla puede tener varias llaves foráneas en función de sus relaciones con otras tablas.

La restricción de llave foránea ayuda a mantener la integridad referencial de los datos entre las tablas principal y secundaria.

Una restricción de llave foránea indica que los valores en una columna o un grupo de columnas en la tabla secundaria son iguales a los valores en una columna o un grupo de columnas de la tabla principal.

Sintaxis de restricción de llave foránea

```
[CONSTRAINT nombre_foreign_key]
FOREIGN KEY(columnas_foreign_key)
REFERENCES tabla_principal(columnas_tabla_principal)
[ON DELETE accion_eliminar]
[ON UPDATE accion_actualizar]
```

En esta sintaxis:

- Primero, se especifica el nombre de la restricción de llave foránea después de la palabra CONSTRAINT. La cláusula CONSTRAINT es opcional. Si lo omitimos, se asignará un nombre generado automáticamente.
- En segundo lugar, especificamos una o más columnas de clave foránea entre paréntesis después de las palabras clave de FOREIGN KEY.

- En tercer lugar, especificamos la tabla principal y las columnas de llave primaria a las que hacen referencia las columnas de llave foránea en la cláusula REFERENCES.
- Finalmente, especificamos las acciones de eliminación y actualización en las cláusulas ON DELETE y ON UPDATE.

Las acciones de eliminación y actualización determinan los comportamientos cuando se elimina y actualiza la llave primaria de la tabla principal. Dado que la llave primaria rara vez se actualiza, la acción ON UPDATE no se usa a menudo en la práctica. Nos centraremos en la acción ON DELETE.

Se admiten las siguientes acciones:

- SET NULL
- SET DEFAULT
- RESTRICT
- NO ACTION
- CASCADE

Para ejemplificar esta situación, haremos uso de las tablas asignatura y departamento, creadas en ejemplos anteriores. Como es sabido, existe una relación de dependencia lógica entre ambas tablas, pero no existe una manera que impida errores de integridad en los datos.

En este ejemplo, la tabla departamento es la tabla principal y la tabla asignatura es la tabla secundaria.

Cada departamento tiene cero o muchas asignaturas y cada asignatura pertenece a cero o un departamento. Por tanto, se debe crear una restricción que mantenga la coherencia entre los datos:

```
ALTER TABLE asignatura ADD CONSTRAINT fk_departamento
FOREIGN KEY (departamento_id)
REFERENCES departamento (departamento_id);
```

La columna departamento_id en la tabla asignatura es la columna de llave foránea que hace referencia a la columna de llave primaria con el mismo nombre en la tabla departamento.

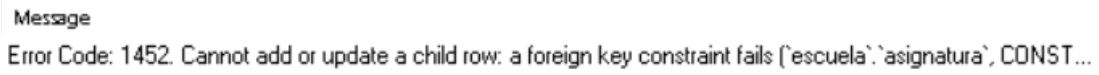
Debido a que la restricción de llave foránea en este ejemplo no tiene definidas las acciones ON DELETE y ON UPDATE, ésta asume su valor por defecto, que es NO ACTION.

Ejemplo NO ACTION

Insertamos los siguientes datos en la tabla asignatura, con el objetivo de explorar el funcionamiento de NO ACTION:

```
INSERT INTO asignatura (asignatura_id, departamento_id, nombre, descripcion) VALUES (8,10,'Física','Ciencia que estudia las propiedades de la materia y de la energía');
```

Al intentar hacer lo anterior, retornará un error:



Ejemplo SET NULL

La opción ON DELETE SET NULL actualiza a NULL los valores de en las columnas FOREIGN KEY de la tabla secundaria en casos en que se eliminen registros de la tabla principal que posean relación con éstas.

Las siguientes instrucciones actualizan la llave foránea antes creada para habilitar la opción SET NULL, y eliminan un registro de la base de datos:

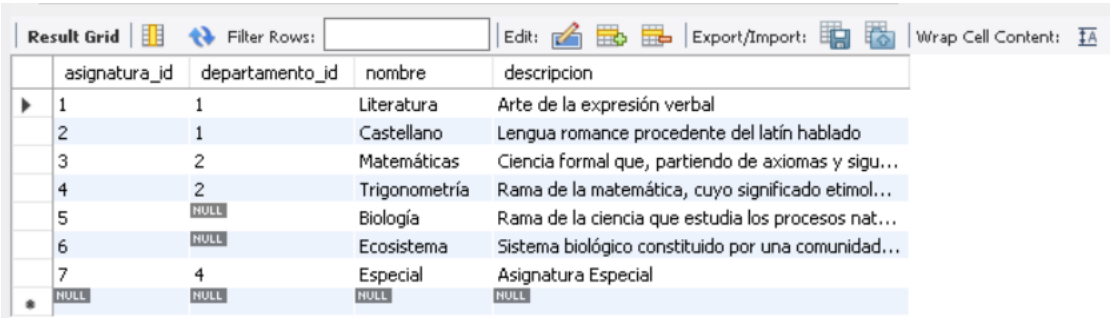
- Eliminar la llave foránea antes creada
- Quitar la característica NOT NULL al campo departamento_id de la tabla asignatura
- Crear nuevamente la llave foránea, pero con la opción SET NULL

```
ALTER TABLE asignatura DROP CONSTRAINT fk_departamento;

ALTER TABLE asignatura MODIFY departamento_id INT;

ALTER TABLE asignatura ADD CONSTRAINT fk_departamento
FOREIGN KEY (departamento_id)
REFERENCES departamento (departamento_id)
ON DELETE SET NULL;
```

El siguiente paso es eliminar un departamento. En este caso se eliminará el registro con ID igual a 3:

```
DELETE FROM departamento WHERE departamento_id = 3;
```

La tabla asignatura, entonces, quedará de la siguiente manera:



asignatura_id	departamento_id	nombre	descripcion
1	1	Literatura	Arte de la expresión verbal
2	1	Castellano	Lengua romance procedente del latín hablado
3	2	Matemáticas	Ciencia formal que, partiendo de axiomas y sigu...
4	2	Trigonometría	Rama de la matemática, cuyo significado etimol...
5	NULL	Biología	Rama de la ciencia que estudia los procesos nat...
6	NULL	Ecosistema	Sistema biológico constituido por una comunidad...
7	4	Especial	Asignatura Especial
NULL	NULL	NULL	NULL

Ejemplo CASCADE

La opción ON DELETE CASCADE elimina automáticamente todas las filas de referencia en la tabla secundaria cuando se eliminan las filas a las que se hace referencia en la tabla principal.

Ejemplo SET DEFAULT

Por su parte, la opción ON DELETE SET DEFAULT establece el valor predeterminado en la columna de FOREIGN KEY de la tabla secundaria cuando se eliminan las filas referenciadas de la tabla principal.

Ejemplo RESTRICT

La acción RESTRICT es similar a NO ACTION. La diferencia solo surge cuando define la restricción de clave externa como DEFERRABLE con un modo INITIALLY DEFERRED o INITIALLY IMMEDIATE. Se deja esto al lector para consultar en la documentación de la base de datos.

3.3.7.- Restricciones en una tabla

Las restricciones son las reglas que se aplican a las columnas de datos de la tabla. Estos se utilizan para evitar que se ingresen datos no válidos en la base de datos. Esto asegura la precisión y confiabilidad de ésta.

Las restricciones pueden ser de nivel de columna o de tabla. Las restricciones a nivel de columna se aplican solo a una columna, mientras que las restricciones a nivel de tabla se aplican a toda la tabla. Definir un tipo de datos para una columna es una restricción en sí misma. Por ejemplo, una columna de tipo FECHA limita la columna a fechas válidas.

Las siguientes son restricciones de uso común:

- NOT NULL: Garantiza que una columna no pueda tener un valor NULL.
- UNIQUE: Garantiza que todos los valores de una columna sean diferentes.
- PRIMARY KEY: Identifica con un valor único cada registro en una tabla.
- FOREIGN KEY: Restringe los datos basados en columnas en otras tablas.
- CHECK: Garantiza que todos los valores de una columna satisfagan determinadas condiciones.

NOT NULL

De forma predeterminada, una columna puede contener valores NULL. Si no desea que una columna tenga un valor NULL, entonces debe definir dicha restricción en la definición de la columna especificando que NULL ahora no está permitido para sus datos. Una restricción NOT NULL siempre se escribe como una restricción de columna.

UNIQUE

La restricción UNIQUE evita que dos registros tengan valores idénticos en una columna en particular. En la tabla EMPRESA, por ejemplo, es posible definir que queremos evitar que dos o más personas tengan la misma edad.

PRIMARY KEY

Como hemos visto anteriormente en otros ejemplos, la restricción PRIMARY KEY identifica de forma única cada registro en una tabla de base de datos. Puede haber más columnas UNIQUE, pero solo una PRIMARY KEY en una tabla. Las usamos para hacer referencia a las filas particulares de una tabla. Además, las PRIMARY KEYs se convierten en FOREIGN KEYS en otras tablas, al crear relaciones entre tablas.

Una PRIMARY KEY es un campo en una tabla, que identifica de forma única cada fila / registro de ésta. Las llaves primarias deben contener valores únicos. Una columna de llave primaria no puede contener valores NULL.

Una tabla sólo puede tener una PRIMARY KEY, que puede constar de uno o varios campos. Cuando se utilizan varios campos como llave primaria, se denominan llave compuesta.

FOREIGN KEY

Una restricción de llave foránea (FOREIGN KEY) especifica que los valores de una columna (o un grupo de columnas) deben coincidir con los valores que aparecen en alguna fila de otra tabla. Decimos que esto mantiene la Integridad Referencial entre dos tablas relacionadas. Se llaman FOREIGN KEY porque las restricciones son externas; es decir, fuera de la tabla. Las FOREIGN KEYs a veces se denominan llave de referencia.

CHECK

La restricción CHECK permite verificar una condición para el valor que se ingresa en un registro. Si la condición se evalúa como falsa, el registro viola la restricción y no se ingresa en la tabla.

Eliminando restricciones

Para eliminar una restricción, necesitamos referirnos a su nombre de la siguiente forma:

```
ALTER TABLE nombre_tabla DROP CONSTRAINT nombre_restriccion;
```

3.4.- Transaccionalidad en las operaciones

3.4.1.- Qué es una transacción y por qué son importantes

Una transacción es una unidad de trabajo que se realiza en una base de datos. Las transacciones son unidades o secuencias de trabajo realizadas en un orden lógico, ya sea de forma manual por un usuario o automáticamente por algún tipo de programa de base de datos.

Una transacción es la propagación de uno o más cambios a la base de datos. Por ejemplo, si está creando, actualizando o eliminando un registro de una tabla, entonces se está realizando una transacción en la tabla. Es importante controlar las transacciones para garantizar la integridad de los datos y manejar los errores de la base de datos.

En la práctica, se usa agrupar múltiples consultas y ejecutarlas todas juntas como parte de una transacción. A fin de entender la importancia de este concepto, se explicará en primer lugar la diferencia entre las tablas MyISAM e InnoDB.

Características de MyISAM

- Se establece por defecto cuando se crea una tabla, salvo que se indique lo contrario.
- Soporta transacciones.
- Realizar bloqueo de registros.
- Soporta un gran número de consultas SQL, lo que se refleja en una velocidad de carga muy rápida para nuestra web.

Como desventaja, señalamos que no realiza bloqueo de tablas, esto puede ser un problema si como se ha mencionado anteriormente hay un acceso simultáneo al mantenimiento de registros por parte de varios usuarios.

Características de InnoDB

- Bloqueo de registros. Importante para accesos múltiples al mantenimiento de tablas, es decir, ejecuciones de sentencias tipo INSERT o UPDATE, éstas ejecuciones tienen una velocidad optimizada.
- Capacidad para soportar transacciones e integridad de datos, es decir previene el alta de datos no adecuados.
- Aplica las características propias de ACID (Atomicity, Consistency, Isolation and Durability), consistentes en garantizar la integridad de las tablas.

Como desventaja, marcamos que al ser un tipo de motor que define un sistema más complejo de diseño de tablas, reduce el rendimiento en velocidad para desarrollo que requieren de un elevado número de consultas.

Qué tipo de tabla usar

Se pueden dar las siguientes situaciones:

- Un solo gestor de mantenimiento para una plataforma que requerirá muchas consultas o visitas: MyISAM
- Necesitas velocidad y mínimo consumo de recursos en servidor, espacio, RAM, etc.: MyISAM
- Varios o muchos gestores de mantenimiento: InnoDB
- Desarrollo donde se prioriza el diseño relacional de bases de datos: InnoDB

Independientemente del sistema que se elija, hay que hacer un buen diseño de la estructura y funcionalidad de la base de datos. Al mismo tiempo, la información o datos no deben almacenarse de cualquier manera. Hay que buscar el mayor aprovechamiento de los recursos que

tenemos a nuestra disposición, tanto a nivel de almacenamiento como de rendimiento.

Además, hay que mantener la consistencia de la información durante todo el ciclo de vida de la base de datos, más aún si los datos que se manejan son críticos; por ejemplo, los salarios de una organización.

Los primeros factores que realizará el analista serán el análisis del sistema, que servirá de modelo, la observación de los elementos que lo componen y la descomposición en partes mucho más pequeñas.

Las tablas del tipo InnoDB están estructuradas de forma distinta que MyISAM, ya que se almacenan en un sólo archivo en lugar de tres, y sus principales características son que permite trabajar con transacciones, y definir reglas de integridad referencial.

El soporte de transacciones que provee MySQL no es algo nuevo en MySQL, ya que desde la versión 3.23 se podía hacer uso de tablas InnoDB; la única diferencia es que con la llegada de la versión 4.0 de MySQL, el soporte para este tipo de tablas es habilitado por default.

Las transacciones aportan una fiabilidad superior a las bases de datos. Si disponemos de una serie de consultas SQL que deben ejecutarse en conjunto, con el uso de transacciones podemos tener la certeza de que nunca nos quedaremos a medio camino de su ejecución. De hecho, podríamos decir que las transacciones aportan una característica de "deshacer" a las aplicaciones de bases de datos.

Para este fin, las tablas que soportan transacciones, como es el caso de InnoDB, son mucho más seguras y fáciles de recuperar si se produce algún fallo en el servidor, ya que las consultas se ejecutan o no en su totalidad. Por otra parte, las transacciones pueden hacer que las consultas tarden más tiempo en ejecutarse.

3.3.2. Propiedades de las transacciones: atomicidad, consistencia, aislamiento, durabilidad

Las transacciones tienen las siguientes cuatro propiedades estándar, a las que generalmente se hace referencia con el acrónimo ACID (del inglés Atomicity, Coherence, Isolation, Durability):

- **Atomicidad:** Garantiza que todas las operaciones dentro de la unidad de trabajo se completen con éxito; de lo contrario, la transacción se aborta en el punto de falla y las operaciones anteriores se revierten a su estado anterior.
- **Coherencia:** Garantiza que la base de datos cambie correctamente de estado tras una transacción confirmada con éxito.
- **Aislamiento:** Permite que las transacciones funcionen de forma independiente y transparente entre sí.
- **Durabilidad:** Asegura que el resultado o efecto de una transacción comprometida persista en caso de falla del sistema.

3.4.3.- Confirmación de una transacción

Los comandos de control transaccional se utilizan con los comandos DML INSERT, UPDATE y DELETE únicamente. No se pueden usar al crear tablas o descartarlas porque estas operaciones se confirman automáticamente en la base de datos.

Las transacciones se pueden iniciar usando START. Estas transacciones generalmente persisten hasta que se encuentra el siguiente comando COMMIT o ROLLBACK. Una transacción también hará ROLLBACK si la base de datos se cierra o si ocurre un error.

La siguiente es la sintaxis simple para iniciar una transacción:

```
BEGIN TRANSACTION;
```

El comando COMMIT es el comando transaccional que se utiliza para guardar los cambios invocados por una transacción en la base de datos. Este comando guarda todas las transacciones en la base de datos desde el último comando COMMIT o ROLLBACK.

La sintaxis del comando COMMIT es la siguiente:

```
COMMIT;
```

3.4.4.- Vuelta atrás de una transacción

El comando ROLLBACK es el comando transaccional que se utiliza para deshacer transacciones que aún no se han guardado en la base de datos. El comando ROLLBACK solo se puede utilizar para deshacer transacciones desde que se emitió el último comando COMMIT o ROLLBACK.

La sintaxis del comando ROLLBACK es la siguiente:

```
ROLLBACK;
```

3.3.5.- Modo autocommit

El modo autocommit indica si los resultados de las consultas DML realizadas serán almacenadas directamente en la base de datos. Por defecto esté dato es igual a Verdadero, lo que significa que, por defecto, las transacciones se reflejarán inmediatamente.

Sin embargo, este modo se puede desactivar, a fin de poder confirmar o deshacer el resultado de una transacción. Después de deshabilitar el modo de confirmación automática, estableciendo la variable de confirmación automática en cero, los cambios en las tablas de transacciones seguras (como las de InnoDB o NDB) no se hacen permanentes de inmediato. Debe utilizar COMMIT para almacenar sus cambios en el disco o ROLLBACK para ignorar los cambios.

El ***autocommit*** es una variable de sesión y debe configurarse para cada sesión.

Para entender mejor este punto, haremos un ejemplo:

- En primer lugar, iniciamos la transacción.
- Desactiva el autocommit.
- Inserta dos registros en la tabla profesor.
- Muestra los registros existentes.
- Usa un comando para deshacer la acción.
- Muestra los registros existentes.
- Activa el autocommit.

El código antes indicado se vería de la siguiente manera:

```
START TRANSACTION;
SET autocommit=0;

INSERT INTO profesor VALUES (10, 'Alfonsina', 'Araya', 'Escuela D-255', '2021-01-01', 600000);
INSERT INTO profesor VALUES (11, 'Bernardo', 'Bustos', 'Escuela Z-001', '2021-02-02', 850000);

SELECT * FROM profesor;

ROLLBACK;

SELECT * FROM profesor;

SET autocommit=1;
```

La primera consulta de selección arrojará el siguiente resultado:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

profesor_id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
1	Juanita	Perez	Gabriela Mistral	2011-10-30	234000
2	Bruce	Lee	Republica Popular China	1993-05-22	780945
3	Juan Alberto	Valdivieso	Sagrada Concepcion	2005-08-01	3400000
4	Pablo	Rojas	E-34	2011-10-30	300000
5	Nicolas	Echenique	Bendito Corazón de María	2005-08-30	8900000
6	Jericho	Jorquera	A-18 Abrazo de Maipu	2010-10-22	67500
7	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
8	Pablo	Lizarraga	Sagrado Grial de Montegrande	2004-08-21	6800000
9	Wong	Lee	Santiago de la extremadura	2000-10-26	780000
10	Alfonsina	Araya	Escuela D-255	2021-01-01	600000
11	Bernardo	Bustos	Escuela Z-001	2021-02-02	850000
NULL	NULL	NULL	NULL	NULL	NULL

La segunda consulta de selección, en tanto, mostrará los registros originales, ya que al hacer ROLLBACK se deshacen los cambios.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

profesor_id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
1	Juanita	Perez	Gabriela Mistral	2011-10-30	234000
2	Bruce	Lee	Republica Popular China	1993-05-22	780945
3	Juan Alberto	Valdivieso	Sagrada Concepcion	2005-08-01	3400000
4	Pablo	Rojas	E-34	2011-10-30	300000
5	Nicolas	Echenique	Bendito Corazón de María	2005-08-30	8900000
6	Jericho	Jorquera	A-18 Abrazo de Maipu	2010-10-22	67500
7	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
8	Pablo	Lizarraga	Sagrado Grial de Montegrande	2004-08-21	6800000
9	Wong	Lee	Santiago de la extremadura	2000-10-26	780000
NULL	NULL	NULL	NULL	NULL	NULL

Anexo: Referencias

1.- MySQL: Consulta SELECT
Referencia:
https://www.anerbarrena.com/mysql-select-consultas-base-datos-5426/

2.- Autoincrement
Referencia: https://www.w3schools.com/sql/sql_autoincrement.asp

3.- Show create table
Referencia:
https://dev.mysql.com/doc/refman/8.0/en/show-create-table.html

4.- Sentencias START TRANSACTION, COMMIT, y ROLLBACK
Referencia: https://dev.mysql.com/doc/refman/8.0/en/commit.html

5.- Transacciones en MySQL
Referencia:
https://programacion.net/articulo/transacciones_en_mysql_242

6.- MyISAM vs InnoDB
Referencia: https://www.webreunidos.es/myisam-vs-innodb/