Escrito por **Adrián Quiroga Linares**.

SQL es el **lenguaje de consulta** de bases de datos más ampliamente utilizado. Aunque se le suele llamar lenguaje de consulta también tiene instrucciones para **crear, modificar y eliminar tablas, insertar modificar y eliminar tuplas y definir restricciones de integridad.**

5.1 Lenguaje de deficion de datos (LDD)

SQL proporciona comandos para la definición y modificación de esquemas de relación y el borrado de relaciones. El **LDD** permite la especificación de un conjunto de relaciones y de su información relativa, incluyendo: - **El esquema de cada relación**. - **El dominio de valores asociado a cada atributo**. - **Las restricciones de integridad** - **El conjunto de índices que se deben mantener para cada relación** - **La información de seguridad y de autorización de cada relación** - **La estructura de almacenamiento físico de cada relación en el disco.**

5.2 Tipos Básicos de Datos

- **char(n)**: Cadena de caracteres de longitud fija, donde (n) es especificado por el usuario. También se puede utilizar **character**.
- **varchar(n)**: Cadena de caracteres de longitud variable, con una longitud máxima de (n) especificada por el usuario. La forma completa **character varying** es equivalente.
- **int**: Número entero, un subconjunto finito de los enteros dependiente de la máquina. **integer** es equivalente.
- **smallint**: Entero pequeño, un subconjunto dependiente de la máquina del tipo de dominio entero.
- **numeric(p, d)**: Número de coma fija con precisión especificada por el usuario. Se compone de (p) dígitos (más el signo), de los cuales d pertenecen a la parte decimal. Por ejemplo, **numeric(3,1)** puede almacenar 44.5, pero no 444.5 ni 0.32.
- **real, double precision**: Números de coma flotante, con precisión dependiente de la máquina.
- **float(n)**: Número de coma flotante cuya precisión es, al menos, de (n) dígitos.

Todos los tipos incluyen un valor especial llamado **valor null** (nulo), que indica la ausencia de un valor.

Tipos de datos de fecha y hora: - **date:** fecha del calendario que contiene el año, el mes y el día del mes - **time:** hora del día en horas, minutos y segundos, se puede almacenar también el huso horario con `time with timezone` - **timestamp:** combinación de date y time

5.3 Definición de tablas en SQL

Las relaciones se definen mediante el comando `CREATE TABLE`

```
CREATE TABLE departamento (
    nombre_dept VARCHAR(20),
    edificio VARCHAR(15),
    presupuesto NUMERIC(12,2),
    PRIMARY KEY (nombre_dept)
);
```

Relación Base: Una tabla con nombre correspondiente a una relación del esquema de la Base de Datos dónde las tuplas están almacenadas físicamente en el Gestor de Bases de Datos.

1. **PRIMARY KEY** $(A_j1, A_j2, ..., A_jm)$:
    - Los atributos forman la clave primaria de la relación.
    - Los atributos deben ser no nulos y únicos.
2. **FOREIGN KEY** $(A_k1, A_k2, ..., A_kn)$ **REFERENCES s**:
    - Indica que los valores de los atributos deben corresponder con los de la clave primaria de otra relación.
    - Ejemplo: En la tabla `asignatura`, `foreign key (nombre_dept) references departamento`.
    - Además si el nombre de ambos atributos que se relacionan no coindicen hay que especificarlo: `foreign key (nombre_dept) references departamento(nombre)`

```
CREATE TABLE sección (
    asignatura_id VARCHAR(8),
    secc_id VARCHAR(8),
    semestre VARCHAR(6),
    año NUMERIC(4,0),
    edificio VARCHAR(15),
    número_aula VARCHAR(7),
    franja_horaria_id VARCHAR(4),
    PRIMARY KEY (asignatura_id, secc_id, semestre, año),
    FOREIGN KEY (asignatura_id) REFERENCES asignatura
);
```

5.4 Modificación de Tablas

Insertar una tupla: `sql INSERT INTO profesor (atrib1,...,atribn) VALUES (10211, 'Smith', 'Biología', 66000);`

Eliminar todas las tuplas de una tabla:
`sql DELETE FROM estudiante;`

Eliminar relaciones tablas: `sql DROP TABLE profesor;`

Modificar la estructura de tablas: - **Añadir atributos**:

```
    ALTER TABLE profesor ADD sueldo int;
    ```

  - **Eliminar atributos**:
    ```sql
    ALTER TABLE profesor DROP sueldo;
    ```

5.5 Actualizaciones

Actualizar todas las tuplas
```sql
UPDATE profesor SET sueldo = sueldo * 1.05;
```

5.6 Vistas

Son relaciones actuales que contienen el resultado de las consultas para hacer visible al usuario los datos que este puede ver. En SQL las vistas se definen con :

```
CREATE VIEW v AS (expresión de consulta);
```

Vista: El resultado dinámico de una o más operaciones relacionales sobre las relaciones base (y, opcionalmente, vistas previas) para producir otra relación. Una vista, como relación virtual, no tiene por qué existir físicamente en el Gestor de la Base de Datos, sino que puede producirse cuando se solicite por parte del usuario.

Razones para usar vistas: 1. **Seguridad:** Algunos datos deben permanecer ocultos para ciertos usuarios. Por ejemplo, un empleado puede necesitar ver el ID, nombre y departamento de un profesor, pero no su sueldo. Esto se logra mediante una consulta SQL que solo devuelve la información permitida.

2. **Personalización:** A veces, es útil crear relaciones que sean más intuitivas para los usuarios. Por ejemplo, se podría querer listar todas las secciones de asignaturas de un departamento específico en un semestre determinado. Para ello, se utiliza una consulta SQL que filtra la información necesaria.

Vistas Materializadas

Las **vistas materializadas** son una extensión de las vistas tradicionales en SQL que permiten almacenar los resultados de una consulta en la base de datos. Esto significa que, a diferencia de las vistas normales, que se recalculan cada vez que se accede a ellas, las vistas materializadas guardan un conjunto de datos que se puede consultar directamente.

1. **Mantenimiento Inmediato:** La vista se actualiza automáticamente cada vez que se modifica una de las tablas que la definen.

2. **Mantenimiento Perezoso:** La vista se actualiza solo cuando se accede a ella, lo que puede resultar en datos desactualizados hasta que se consulte.
3. **Actualización Periódica:** Algunas bases de datos permiten que la vista se actualice en intervalos programados, lo que puede llevar a que los datos estén desfasados.

- **Rendimiento Mejorado:** Almacenar resultados de consultas complejas o agregaciones puede acelerar significativamente el tiempo de respuesta, especialmente en aplicaciones que requieren respuestas rápidas.
- **Reducción de Carga:** Las vistas materializadas permiten evitar la lectura de grandes conjuntos de datos subyacentes, al trabajar con un conjunto de datos más pequeño.

Si bien las vistas materializadas ofrecen ventajas en términos de rendimiento, también conllevan costos adicionales, como: - **Almacenamiento:** Necesitan espacio adicional en disco para almacenar los resultados. - **Sobrecarga de Actualización:** El mantenimiento de la vista puede requerir recursos y tiempo, especialmente si se actualiza con frecuencia.

La creación de tablas usando `CREATE TABLE ... AS` es similar a la creación de vistas (`CREATE VIEW`). La diferencia principal es que los datos en una tabla son copiados en el momento de la creación, mientras que una vista siempre refleja los resultados actualizados de la consulta que la define. Por ejemplo:

```
CREATE VIEW vista_profesores AS
SELECT * FROM profesor WHERE nombre_dept = 'Música';
```

Aquí, `vista_profesores` siempre mostrará los datos actuales de la tabla `profesor` donde `nombre_dept` es 'Música', pero no almacena esos datos de forma física como lo hace `t1`.

5.5 Restricciones de Integridad

Las **restricciones de integridad** garantizan la consistencia de la base de datos, especialmente durante las actualizaciones de información, evitando que las modificaciones realizadas por usuarios den lugar a una **pérdida de consistencia** de los datos. ## Restricciones sobre una sola relación 1. **Not Null:** prohibe la inserción de valores nulos para ese atributo, las claves primarias reciben esta restricción automáticamente. 2. **Unique:** indica que esos atributos forman una clave candidata, ningún par de tuplas, puede tener el mismo valor para esos atributos. 3. **Check(p):** especifica un predicado que deben satisfacer todas las tuplas de una relación.

```
CREATE TABLE profesor (
     ID INT PRIMARY KEY,
     nombre VARCHAR(100) NOT NULL
);

CREATE TABLE departamento (
    nombre_dept VARCHAR(50) UNIQUE,
    presupuesto DECIMAL(10, 2)
);
```

```sql
CREATE TABLE asignatura (
    asignatura_id INT PRIMARY KEY,
    nombre VARCHAR(100),
    créditos INT CHECK (créditos > 0)
);
```

Integridad referencial

Se fuerza a que los valores válidos en las claves externas sean valores que existan en las claves candidatas referenciadas (*fundamentalmente para la consistencia interna de la base de datos*).

```sql
foreign key (nombre_dept) references departamento(nombre_dept)
```

SQL permite manejar violaciones de integridad referencial de distintas maneras. Las acciones más comunes son:

- **Cascade**: Si se borra o actualiza un registro en la tabla referenciada, el cambio se propaga a las tablas que referencian ese registro.

  ```sql
  foreign key (nombre_dept) references departamento on delete cascade on update cascade
  ```

- **Set null**: Al borrar o actualizar un registro referenciado, los valores de las columnas referenciadas se establecen a `null`.

  ```sql
  foreign key (nombre_dept) references departamento(nombre_dept) on delete set null
  ```

- **Set default**: Similar a `set null`, pero los valores se establecen a un valor predeterminado.

  ```sql
  foreign key (nombre_dept) references departamento(nombre_dept) on delete set default
  ```

Ejemplo SQL

```sql
-- Crear la tabla Categoría
CREATE TABLE Categoria (
    id_categoria INT PRIMARY KEY,
    nombre_categoria VARCHAR(100) NOT NULL
);

-- Crear la tabla Producto con clave foránea que utiliza ON DELETE SET NULL
CREATE TABLE Producto (
    id_producto INT PRIMARY KEY,
    nombre_producto VARCHAR(100) NOT NULL,
    id_categoria INT,
    FOREIGN KEY (id_categoria) REFERENCES Categoria(id_categoria)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);
```

```sql
-- Crear la tabla Proveedor
CREATE TABLE Proveedor (
    id_proveedor INT PRIMARY KEY,
    nombre_proveedor VARCHAR(100) NOT NULL
);

-- Crear la tabla Pedido con clave foránea que utiliza ON DELETE CASCADE
CREATE TABLE Pedido (
    id_pedido INT PRIMARY KEY,
    id_proveedor INT NOT NULL,
    FOREIGN KEY (id_proveedor) REFERENCES Proveedor(id_proveedor)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- Crear la tabla Cliente
CREATE TABLE Cliente (
    id_cliente INT PRIMARY KEY,
    nombre_cliente VARCHAR(100) NOT NULL,
    pais_origen VARCHAR(50) DEFAULT 'Desconocido'
);

-- Crear la tabla Venta con clave foránea que utiliza ON DELETE SET DEFAULT
CREATE TABLE Venta (
    id_venta INT PRIMARY KEY,
    id_cliente INT NOT NULL,
    FOREIGN KEY (id_cliente) REFERENCES Cliente(id_cliente)
    ON DELETE SET DEFAULT
    ON UPDATE CASCADE
);
```

Acciones en violaciones

Cuando ocurre una violación de una restricción de integridad referencial, el sistema puede rechazar la transacción, revertir los cambios, o realizar alguna de las acciones mencionadas (cascade, set null, etc.). Si no es posible resolver la violación con ninguna de estas acciones, se aborta la transacción.

Este conjunto de reglas asegura que las relaciones entre las tablas sean coherentes y se mantenga la integridad de los datos.

> [!Definiciones] **Relación base:** Una tabla con nombre correspondiente a una relación del esquema de la Base de Datos dónde las tuplas están almacenadas físicamente en el Gestor de Bases de Datos.
>
> **Vista:** El resultado dinámico de una o más operaciones relacionales sobre las relaciones base (y, opcionalmente, vistas previas) para producir otra relación. Una vista, com relación virtual, no tiene por qué existir físicamente en el

Gestor de la Base de Datos, sino que puede producirse cuando se solicite por parte del usuario.