Lectura complementaria.

El Lenguaje de Definición de Datos (DDL) y su Aplicación en Bases de Datos Relacionales

1. Introducción al Lenguaje SQL y el Rol de DDL

1.1. Fundamentos de SQL: Lenguaje de Consulta Estructurado

El Lenguaje de Consulta Estructurado, o SQL por sus siglas en inglés, constituye el estándar de facto para la gestión y manipulación de bases de datos relacionales. Su naturaleza declarativa permite a los usuarios especificar qué datos desean obtener o modificar, sin necesidad de definir el procedimiento exacto para lograrlo. Una base de datos relacional se estructura como una colección de tablas interconectadas, donde cada tabla es una organización bidimensional de datos que se compone de filas y columnas.[1] La capacidad de definir y controlar la estructura de estas tablas y sus relaciones es fundamental para el diseño y la administración de cualquier sistema de información robusto.

1.2. Los Subconjuntos de SQL: DDL, DML, DCL y TCL

Para comprender la amplitud y el propósito de SQL, es fundamental segmentar sus comandos en sublenguajes, cada uno con una función específica.[4] Esta clasificación permite un enfoque didáctico y funcional en la gestión de bases de datos.

- **Lenguaje de Definición de Datos (DDL):** Su propósito principal es la gestión de la estructura o el esquema de la base de datos.[4] Mediante sentencias DDL, se crean, alteran y eliminan objetos como tablas, esquemas, vistas, índices, permisos y alias.[1] Las operaciones DDL son de naturaleza irreversible y se consideran de auto-commit, lo que significa que sus cambios son permanentes en la estructura de la base de datos y no pueden ser deshechos con un ROLLBACK.[4]
- **Lenguaje de Manipulación de Datos (DML):** Este sublenguaje se centra en la interacción con los datos contenidos en los objetos definidos por DDL.[4] Los comandos más comunes incluyen SELECT para recuperar registros, INSERT para insertar nuevos datos, UPDATE para modificar datos existentes y DELETE para eliminar filas.[4] A diferencia de DDL, las operaciones DML no son de auto-commit, lo que permite la reversión de los cambios a través de comandos como ROLLBACK.[4]
- **Lenguaje de Control de Datos (DCL):** Se ocupa de la administración de los derechos, privilegios y permisos de los usuarios sobre los objetos de la base de datos.[4] Las sentencias GRANT y REVOKE son los comandos principales para otorgar y retirar permisos de acceso, respectivamente.[4]

- **Lenguaje de Control Transaccional (TCL):** Su función es la gestión de las transacciones, lo que garantiza la consistencia, la atomicidad y la durabilidad de las operaciones en la base de datos.[4] Los comandos COMMIT y ROLLBACK permiten guardar el trabajo de forma permanente o deshacer las modificaciones realizadas desde el último COMMIT, respectivamente.[7]

A continuación, se presenta una tabla comparativa para una referencia rápida de estos subconjuntos:

Lenguaje	Propósito	Comandos Principales	Nivel de Operación	Cambios Reversibles
DDL	Definición de la estructura de la base de datos	CREATE, ALTER, DROP, TRUNCATE, RENAME	Esquema/Metadatos	No (Auto-commit)
DML	Manipulación de los datos en las tablas	SELECT, INSERT, UPDATE, DELETE, MERGE	Contenido/Registros	Sí (ROLLBACK)
DCL	Control de acceso y permisos	GRANT, REVOKE	Seguridad/Permisos	Sí (ROLLBACK)
TCL	Gestión de transacciones	COMMIT, ROLLBACK, SAVEPOINT	Transaccionalidad	Sí (ROLLBACK)

1.3. El Rol y el Alcance del Lenguaje de Definición de Datos (DDL)

El Lenguaje de Definición de Datos (DDL) es la base sobre la cual se construye y se mantiene una base de datos relacional. Su propósito es proporcionar una interfaz para describir y definir la organización y las características de los datos.[9] Esto incluye la creación de tablas que son la unidad fundamental para la organización de los datos, la gestión de esquemas para la agrupación lógica de objetos, y la definición de vistas, secuencias, y índices que optimizan el rendimiento.[1]

Las sentencias DDL se utilizan para establecer las reglas que regirán la base de datos, incluyendo la aplicación de restricciones que garantizan la integridad de los datos, como las claves primarias y foráneas.[2] La ejecución de un comando DDL es una operación con un alto grado de riesgo, ya que un error puede alterar o eliminar permanentemente la estructura de la base de datos. Por lo tanto, su uso requiere una comprensión profunda de su funcionamiento y un cuidadoso proceso de diseño.[6]

La diferencia entre DDL y DML no es meramente una cuestión de los comandos utilizados, sino una distinción fundamental en el nivel de abstracción en el que operan. DDL trabaja en el nivel de los metadatos de la base de datos, es decir, en el "plano" de la estructura, definiendo cómo se organizarán los datos. En contraposición, DML opera en el nivel de los datos mismos, el "contenido" de la base de

datos. Esta diferencia en la naturaleza de las operaciones tiene una consecuencia directa en su comportamiento. Las sentencias DDL, al alterar de forma permanente el diseño subyacente, son intrínsecamente irreversibles y se auto-commit.[4] Este comportamiento contrasta con la naturaleza transaccional de DML, que permite la reversibilidad de los cambios, lo que es esencial para la manipulación de datos en un entorno de producción donde los errores deben poder deshacerse.[4] Por lo tanto, la elección de una sentencia DDL versus una DML depende del objetivo: modificar el diseño o modificar el contenido.

2. El Estándar ANSI SQL: Un Marco para la Estandarización

2.1. ¿Qué es el Estándar ANSI SQL?

El estándar ANSI SQL, desarrollado por el Instituto Nacional Estadounidense de Estándares (ANSI) y la Organización Internacional de Normalización (ISO), es una definición formal del lenguaje SQL diseñada para asegurar la consistencia y la interoperabilidad entre los distintos Sistemas de Gestión de Bases de Datos (SGBD).[10] Su objetivo principal es que el código SQL escrito conforme al estándar pueda ser ejecutado en múltiples plataformas con pocas o ninguna modificación, mitigando el riesgo de "vendor lock-in", que se produce cuando una empresa se vuelve dependiente de un solo proveedor.[11]

2.2. Historia y Evolución del Estándar (SQL-86 a la Actualidad)

La primera formalización de SQL se produjo en 1986 con el estándar SQL-86 de ANSI, seguido de cerca por la versión de la ISO en 1987.[10] Esta versión inicial estableció las bases para las operaciones de DDL (CREATE, DROP) y DML (SELECT, INSERT, UPDATE, DELETE).[11] Una revisión menor se publicó en 1989 (SQL-89).[10]

Sin embargo, el hito más significativo en la estandarización fue SQL-92 (también conocido como SQL-2). Esta versión de 1992 expandió notablemente el estándar, incorporando funcionalidades cruciales de DDL como el soporte para restricciones de integridad de datos, incluyendo PRIMARY KEY, FOREIGN KEY y UNIQUE.[11] Las revisiones posteriores, como SQL:1999, introdujeron características más avanzadas, como los tipos de datos definidos por el usuario, las consultas recursivas y los disparadores. La evolución del estándar ha continuado, con revisiones publicadas en 2003, 2006, 2008, 2011, 2016 y la más reciente en 2023, lo que demuestra un esfuerzo constante por adaptarse a las necesidades del mercado y la tecnología.[10]

2.3. La Importancia de la Estandarización

El cumplimiento del estándar ANSI SQL genera una serie de beneficios sustanciales para el desarrollo de software y la industria en general. En primer lugar, promueve la portabilidad de las aplicaciones, lo que facilita la migración de sistemas entre diferentes SGBD con un esfuerzo mínimo.[11] Esta portabilidad es crítica para la flexibilidad empresarial. En segundo lugar, la estandarización proporciona transparencia y promueve un consenso en el sector.[12] Al seguir una guía técnica común, los productos y servicios pueden ser comparables, lo que genera confianza en los clientes y simplifica el comercio internacional al reducir

las barreras técnicas.[12] Además, las normas internacionales sirven como una base técnica sólida para la legislación nacional, ya que aportan un consenso internacional sobre las mejores prácticas.[13]

2.4. Diferencias entre Implementaciones Específicas y el Estándar

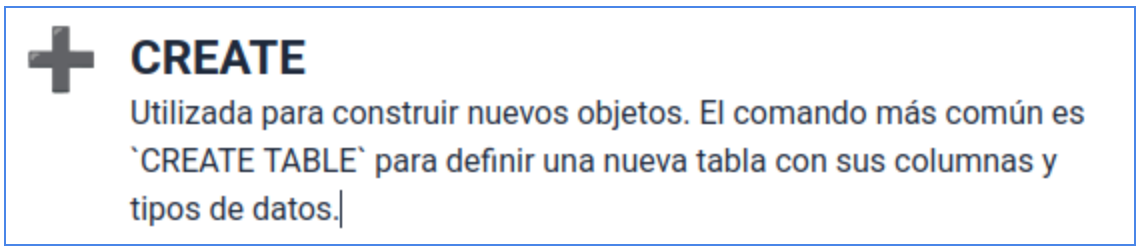
A pesar de la existencia de un estándar, las implementaciones de SQL a menudo presentan extensiones o dialectos específicos de cada proveedor. Por ejemplo, la sintaxis para crear tablas temporales globales puede variar significativamente, utilizando la sentencia DECLARE GLOBAL TEMPORARY TABLE en IBM Db2 o prefijos especiales (# o ##) en SQL Server.[1] Del mismo modo, aunque el estándar ANSI define CREATE TABLE... AS SELECT para crear una tabla a partir de una consulta, algunos sistemas como SQL Server ofrecen una alternativa como la sentencia SELECT... INTO.[2]

La coexistencia del estándar ANSI con las extensiones propietarias no es una contradicción, sino un reflejo del equilibrio necesario entre la interoperabilidad y la innovación. El estándar proporciona una base común, una lingua franca que permite a los desarrolladores moverse entre sistemas sin tener que reaprender la sintaxis fundamental.[11] Sin embargo, los proveedores de SGBD, como IBM y Microsoft, introducen extensiones no estándar para ofrecer funcionalidades avanzadas o un rendimiento optimizado que aún no están contempladas en el estándar o que les permiten diferenciarse de la competencia.[2] Estas extensiones abordan necesidades específicas del mercado, como los mecanismos de auto-incremento de columnas, la gestión de la memoria o las características de seguridad, que el proceso de estandarización, por su naturaleza consensual y deliberada, no puede abordar con la misma rapidez.[2] Comprender esta dualidad es crucial para los estudiantes, quienes deben dominar el núcleo del estándar ANSI y, al mismo tiempo, estar conscientes de los dialectos específicos de cada proveedor que encontrarán en el entorno laboral.

3. Sentencias Fundamentales de DDL: Creación, Modificación y Eliminación de Objetos

3.1. CREATE: Creación de Objetos de Base de Datos

La sentencia CREATE es el punto de partida de cualquier esquema de base de datos. Es el comando DDL principal para la creación de objetos, siendo CREATE TABLE su uso más fundamental.[6]



3.1.1. Sintaxis de CREATE TABLE

La sintaxis básica para la creación de una tabla es la siguiente:

```
CREATE TABLE <nombre_tabla> (<definición_columna_1>,
<definición_columna_2>,...);
```

```
CREATE TABLE nombre_tabla (
   columna1 tipo_dato [restricciones],
   columna2 tipo_dato [restricciones],
   ...
   [restricciones_de_tabla]
);
```

Cada definición de columna debe especificar el nombre de la columna, su tipo de dato, y opcionalmente, una o más restricciones de columna.[2]

Por ejemplo, se puede crear una tabla simple de estudiantes que incluya un identificador, el nombre y la fecha de nacimiento, definiendo el tipo de dato para cada columna.[15]

```
CREATE TABLE estudiante (
    estudiante_id INT PRIMARY KEY,
    nombre VARCHAR(200) NOT NULL,
    fecha_nacimiento DATE
);
```

3.1.2. Cláusulas y Definiciones de Columnas

Dentro de la sentencia CREATE TABLE, la definición de cada columna sigue un patrón estructurado:

```
nombre_columna tipo_dato [restricciones_de_columna]
```

El tipo de dato es fundamental para la correcta definición del esquema, ya que determina el tipo de información que la columna puede almacenar (por ejemplo, enteros, cadenas de caracteres, fechas, etc.).[16] Además, se pueden aplicar restricciones como NOT NULL para asegurar que la columna no pueda quedar vacía, o PRIMARY KEY para designarla como un identificador único.[15]

```
-- Patrón general
-- nombre_columna tipo_dato [restricciones_de_columna]

CREATE TABLE persona (
  persona_id      INTEGER GENERATED ALWAYS AS IDENTITY,   -- identificador
autogenerado (estándar)
  nombre          VARCHAR(100)        NOT NULL,
  email           VARCHAR(255)        NOT NULL UNIQUE,     -- restricción
de unicidad
  fecha_nac       DATE,
  salario         DECIMAL(12,2)       DEFAULT 0.00,
  activo          BOOLEAN             DEFAULT TRUE,
  creado_en       TIMESTAMP           DEFAULT CURRENT_TIMESTAMP,
  -- Restricciones a nivel de tabla (también válidas en ANSI)
  CONSTRAINT pk_persona PRIMARY KEY (persona_id),
  CONSTRAINT ck_salario_no_neg CHECK (salario >= 0),
  CONSTRAINT ck_email_formato CHECK (POSITION('@' IN email) > 1)
);
```

3.1.3. Creación de Tablas a partir de otras Tablas

DDL ofrece funcionalidades avanzadas para la creación de tablas que simplifican el proceso y aprovechan estructuras o datos existentes.

- **CREATE TABLE AS SELECT:** Esta sentencia es una poderosa combinación de DDL y DML.[2] Crea una nueva tabla y la puebla con los resultados de una sentencia SELECT en un solo comando.[19] La nueva tabla hereda los nombres de las columnas y los tipos de datos del conjunto de resultados de la consulta SELECT.[19] Si la consulta utiliza expresiones o funciones, se recomienda encarecidamente utilizar alias de columna para dar nombres significativos a las columnas

resultantes, lo que hace el esquema más legible.[19]

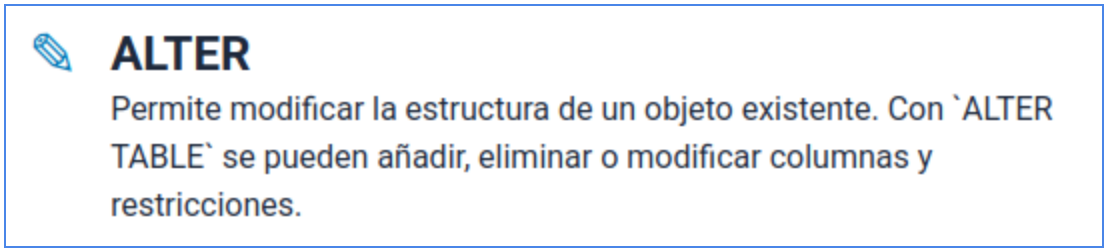
- **CREATE TABLE LIKE:** Este comando crea una tabla nueva y vacía que duplica la estructura de una tabla existente.[1] Esto es útil cuando se necesita replicar el esquema de una tabla, incluyendo la definición de sus columnas y sus atributos, sin copiar sus datos.[2]

```
-- Crear tabla nueva copiando estructura y datos
CREATE TABLE empleados_backup AS
SELECT id, nombre, salario
FROM empleados
WHERE activo = TRUE;


-- Crear tabla nueva copiando solo estructura
CREATE TABLE empleados_vacios
LIKE empleados;
```

3.2. ALTER: Modificación de la Estructura de Tablas

La sentencia ALTER TABLE es una herramienta esencial para la evolución de los esquemas de bases de datos. Permite realizar cambios en la estructura de una tabla existente sin la necesidad de eliminarla y volver a crearla, lo que evita la pérdida de datos.[6]



3.2.1. Adición y Eliminación de Columnas

Para añadir una o más columnas a una tabla, se utiliza la cláusula ADD:

```
ALTER TABLE <nombre_tabla> ADD <nombre_columna> <tipo_dato>
[restricciones];
```

Es posible agregar múltiples columnas en una única sentencia, separando cada definición con una coma.[23]

Para eliminar una columna, se utiliza la cláusula DROP COLUMN:

```
ALTER TABLE <nombre_tabla> DROP COLUMN <nombre_columna>;
```

Se debe proceder con extrema precaución, ya que esta operación es irreversible y elimina permanentemente tanto la columna como todos los datos que contenía.22

```
-- Agregar una columna
ALTER TABLE empleado
   ADD fecha_ingreso DATE;

-- Agregar múltiples columnas
ALTER TABLE empleado
  ADD (telefono VARCHAR(20), direccion VARCHAR(150));

-- Eliminar una columna
ALTER TABLE empleado
  DROP COLUMN direccion;
```

3.2.2. Modificación de la Definición de Columnas

La cláusula ALTER COLUMN o MODIFY COLUMN (dependiendo del SGBD) se usa para cambiar la definición de una columna, como su tipo de dato o su tamaño.[22] Por ejemplo:

```
ALTER TABLE <nombre_tabla> ALTER COLUMN <nombre_columna> <nuevo_tipo_dato>;
```

Aunque esta sentencia es poderosa, existen limitaciones, como la incapacidad de cambiar un tipo de dato que cause la pérdida de información o de reducir el tamaño de una columna que ya contiene datos más grandes.[14]

```
-- Cambiar tipo de dato de una columna
ALTER TABLE empleado
ALTER COLUMN telefono VARCHAR(30);

-- Renombrar columna (no estándar, depende del SGBD)
-- Se suele hacer con: ALTER TABLE ... RENAME COLUMN ...
```

3.2.3. Gestión de Restricciones

La gestión de restricciones se realiza con la sentencia ALTER TABLE utilizando las cláusulas ADD CONSTRAINT para añadir una nueva restricción y DROP CONSTRAINT para eliminar una existente.[23] Para modificar una restricción, el proceso estándar es eliminarla y luego volver a crearla con la nueva definición.[23]

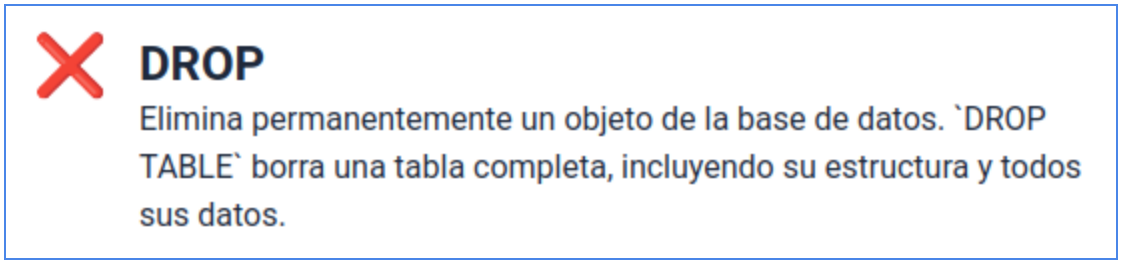
```
-- Agregar restricción de clave foránea
ALTER TABLE pedido
ADD CONSTRAINT fk_pedido_cliente
FOREIGN KEY (cliente_id) REFERENCES cliente(id);

-- Agregar restricción de verificación
ALTER TABLE empleado
ADD CONSTRAINT ck_salario_no_negativo
CHECK (salario >= 0);

-- Eliminar restricción
ALTER TABLE empleado
DROP CONSTRAINT ck_salario_no_negativo;
```

3.3. DROP: Eliminación de Objetos de Base de Datos

El comando DROP es una de las sentencias DDL más definitivas y peligrosas. Se utiliza para eliminar permanentemente un objeto de la base de datos.[6]



3.3.1. Sintaxis de DROP TABLE

Para eliminar una tabla, se usa la siguiente sintaxis:

```
DROP TABLE <nombre_tabla>;
```

La cláusula opcional IF EXISTS es una buena práctica, ya que evita un error si la tabla especificada no

existe.[28]

```sql
-- Eliminar una tabla
DROP TABLE empleado;

-- Con verificación previa (si el motor lo soporta)
DROP TABLE IF EXISTS empleado;
```

3.3.2. Consideraciones y Precauciones

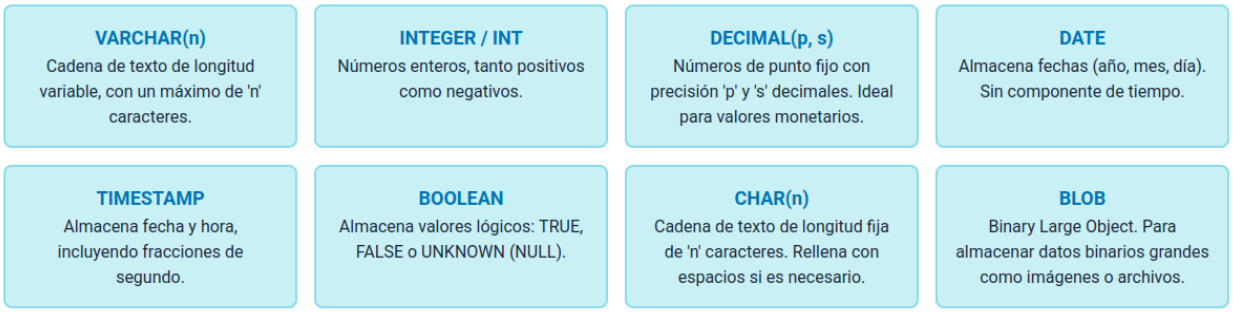
La ejecución de DROP TABLE no solo elimina la tabla, sino también todos los datos que contiene y cualquier índice asociado a ella.[29] Es un comando que debe ser usado con la máxima cautela.[6] En el contexto de las relaciones de clave foránea, un DROP TABLE sobre una tabla padre puede fallar si otras tablas la referencian, a menos que se hayan tomado precauciones para gestionar la integridad referencial.[25]

La existencia de un conjunto de comandos DDL distintos para la creación (CREATE), modificación (ALTER) y eliminación (DROP) de objetos refleja las diferentes fases del ciclo de vida de un esquema de base de datos. CREATE se utiliza para la inicialización y el diseño original. ALTER es la herramienta clave para la evolución incremental y la adaptación del esquema a medida que los requisitos cambian, permitiendo hacer ajustes sin perder los datos subyacentes.[22]

En contraste, DROP representa un acto de demolición total, eliminando la estructura y el contenido de forma permanente.[6] Esta distinción no es arbitraria; está diseñada para proporcionar un control granular sobre las operaciones de DDL, lo que es fundamental para la gestión de bases de datos en entornos de desarrollo y producción, donde la seguridad de los datos y la adaptabilidad del esquema son de vital importancia.

4. Tipos de Datos y su Aplicación en DDL

La elección del tipo de dato es una decisión fundamental en el diseño de una base de datos, ya que define el tipo de información que puede almacenar una columna y afecta directamente a la eficiencia del almacenamiento y al rendimiento de las consultas. El estándar ANSI SQL proporciona una clasificación de tipos de datos que se adapta a una amplia gama de necesidades.[16]



4.1. Tipos de Datos Numéricos

Los tipos de datos numéricos se utilizan para almacenar valores que representan cantidades o pueden ser utilizados en cálculos.[16]

- **INTEGER (o INT):** Para números enteros. Existen variantes como SMALLINT o BIGINT que varían en el rango de valores que pueden almacenar. Se recomiendan para claves primarias o contadores.
- **DECIMAL y NUMERIC:** Para números con decimales de precisión exacta. Son la opción preferida para datos financieros o monetarios, donde la precisión es primordial.
- **FLOAT y DOUBLE PRECISION:** Para números de punto flotante. Son útiles en cálculos científicos o estadísticos donde la precisión no es tan crítica y el rango de valores es muy amplio.

4.2. Tipos de Datos de Cadena de Caracteres

Estos tipos de datos almacenan texto y caracteres, y su elección depende de si la longitud del texto es fija o variable.

- **CHAR(n):** Para cadenas de longitud fija. Almacena siempre *n* caracteres, lo que puede desperdiciar espacio si el texto es más corto.
- **VARCHAR(n):** Para cadenas de longitud variable. Almacena hasta un máximo de n caracteres, pero solo utiliza el espacio necesario, lo que lo hace más eficiente para datos de longitud variable como nombres o descripciones.
- **TEXT y CLOB:** Para textos muy extensos, como artículos o documentos. Se mencionan variantes como NVARCHAR que soportan caracteres Unicode.

4.3. Tipos de Datos de Fecha y Hora

Estos tipos de datos se utilizan para manejar información temporal.

- **DATE:** Almacena solo la fecha (año, mes, día).
- **TIME:** Almacena solo la hora (horas, minutos, segundos).
- **DATETIME y TIMESTAMP:** Almacenan fecha y hora combinadas. TIMESTAMP a menudo se usa para marcar la fecha y hora de la última modificación de un

registro, ya que puede actualizarse automáticamente.

4.4. Tipos de Datos Binarios y Especiales

Estos tipos de datos se usan para almacenar información no textual o con una estructura específica.

- **BINARY y VARBINARY:** Para datos binarios de longitud fija y variable.
- **BLOB:** Para grandes objetos binarios, como archivos de audio, video o imágenes.
- **BOOLEAN o BIT:** Para representar valores lógicos de verdadero o falso.

La selección de tipos de datos es una decisión de diseño crítica que va más allá de la mera categorización de la información.

Una elección incorrecta puede afectar directamente la eficiencia del almacenamiento y el rendimiento de la base de datos. Por ejemplo, el material de investigación menciona que DOUBLE es más preciso, pero también más "pesado" que FLOAT en términos de almacenamiento. Una columna con un tipo de dato innecesariamente grande puede llevar a un desperdicio de espacio y a un aumento en las operaciones de entrada/salida de disco (E/S), lo que, a su vez, ralentiza las consultas que deben procesar una mayor cantidad de datos. La misma lógica se aplica a la diferencia entre CHAR y VARCHAR; si el tamaño de los datos varía, VARCHAR será más eficiente, ya que solo utiliza el espacio necesario para el texto real.Por lo tanto, un diseño de esquema robusto requiere una optimización cuidadosa de los tipos de datos, equilibrando la precisión requerida con el impacto en el rendimiento.

A continuación, se presenta una tabla que resume los tipos de datos comunes en el estándar ANSI:

Categoría	Tipo de Dato ANSI	Descripción	Mejor Uso	Ejemplo de Sintaxis
Numéricos	INTEGER, INT	Entero de tamaño medio.	Claves primarias, contadores.	id INT
	DECIMAL(p, s)	Números de precisión y escala exactas.	Cálculos financieros, precios.	precio DECIMAL(10, 2)
	FLOAT, DOUBLE	Números de punto flotante.	Cálculos científicos o estadísticos.	temperatura DOUBLE
Cadenas	CHAR(n)	Cadena de longitud fija.	Datos con longitud constante (ej., códigos de país).	codigo_pais CHAR(2)
	VARCHAR(n)	Cadena de longitud variable.	Datos de longitud variable (ej., nombres).	nombre VARCHAR(100)
	TEXT, CLOB	Textos muy extensos.	Artículos, descripciones detalladas.	resumen TEXT

Fecha/Hora	DATE	Almacena solo la fecha.	Fechas de nacimiento, eventos.	fecha_nacimiento DATE
	TIME	Almacena solo la hora.	Tiempos del día.	hora_llegada TIME
	TIMESTAMP	Fecha y hora combinadas.	Marcado de tiempo de registros.	fecha_registro TIMESTAMP
Binarios	BINARY, VARBINARY	Datos binarios.	Hashes, claves binarias.	clave_hash BINARY(32)
	BLOB	Objetos binarios grandes.	Imágenes, documentos, archivos.	foto_perfil BLOB
Lógicos	BOOLEAN, BIT	Valores de verdadero o falso.	Estados lógicos (activo/inactivo).	esta_activo BOOLEAN

5. Restricciones (Constraints): Garantizando la Integridad de los Datos

Las restricciones son la principal herramienta de DDL para hacer cumplir las reglas de negocio y garantizar que los datos de la base de datos sean precisos y consistentes.[18] Al aplicar estas reglas directamente al esquema de la base de datos, se previene la inserción de datos no válidos y se mantiene la integridad de las relaciones entre tablas.

5.1. La Importancia de las Restricciones en DDL

Las restricciones actúan como reglas que se aplican a las columnas de una tabla. Su función es fundamental para la integridad de los datos, lo que garantiza que la información almacenada sea válida y coherente. La implementación de estas reglas a través de DDL es un componente esencial del diseño de

bases de datos relacionales, ya que reduce la redundancia de datos y disminuye los problemas de actualización. [31]

5.2. PRIMARY KEY: El Identificador Único de la Entidad

Una clave primaria es una columna o conjunto de columnas que identifica de forma única cada fila en una tabla.[32] Sus propiedades son esenciales para la integridad de la entidad:

- **Unicidad:** No puede haber dos filas con el mismo valor de clave primaria.[34]
- **No Anulabilidad:** Ninguna columna que forme parte de una clave primaria puede contener un valor NULL.[33]
- **Una por Tabla:** Una tabla solo puede tener una clave primaria.[33]

La sintaxis para definir una clave primaria puede ser de dos tipos [34]:

Definición en línea (Inline): La restricción se declara junto a la definición de la columna a la que se aplica.[32]

Por ejemplo:

```
CREATE TABLE empleado ( id_empleado INT PRIMARY KEY, nombre VARCHAR(100) );
```

Definición fuera de línea (Out-of-line): La restricción se define al final de la sentencia CREATE TABLE, lo que permite nombrarla y aplicarla a una o más columnas.[34] Esto es necesario para las claves primarias compuestas.

Por ejemplo:

```
CREATE TABLE pedido (
  id_pedido INT,
  id_cliente INT,
  PRIMARY KEY (id_pedido, id_cliente)
);
```

5.3. FOREIGN KEY: Estableciendo Relaciones entre Entidades

Una clave foránea es una columna o conjunto de columnas que establece una relación entre dos tablas, referenciando la clave primaria (o una clave única) de otra tabla.[25] Este mecanismo es la base de la integridad referencial en el modelo relacional.

La sintaxis más común se define al final de la sentencia CREATE TABLE [18]:

```
FOREIGN KEY (<columnas_hija>) REFERENCES <tabla_padre>(<columnas_padre>)
```

Una característica vital de las claves foráneas son los mecanismos de integridad referencial, que determinan el comportamiento de la base de datos cuando se intenta eliminar o actualizar una fila en la tabla padre.[25]

```
CREATE TABLE cliente (
  cliente_id   INTEGER PRIMARY KEY,
  nombre       VARCHAR(100) NOT NULL
);

CREATE TABLE pedido (
  pedido_id    INTEGER PRIMARY KEY,
  fecha        DATE NOT NULL,
  cliente_id   INTEGER,
  FOREIGN KEY (cliente_id)
    REFERENCES cliente(cliente_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT
);
```

Tabla de Acciones de Integridad Referencial

Opción	Descripción	Impacto en la Integridad de los Datos
ON DELETE CASCADE	Cuando se elimina una fila de la tabla padre, todas las filas dependientes en la tabla hija se eliminan automáticamente.	Mantiene la consistencia al eliminar registros relacionados. Se debe usar con cautela debido a la propagación de la eliminación.
ON DELETE SET NULL	Cuando se elimina una fila de la tabla padre, los valores de la clave foránea en las filas dependientes de la tabla hija se establecen en NULL.	Requiere que la columna de la clave foránea admita valores NULL. Puede generar valores perdidos si el dato es obligatorio.
ON DELETE RESTRICT / ON DELETE NO ACTION	Impide la eliminación de una fila en la tabla padre si existen filas que la referencian en la tabla hija.	Garantiza que no existan "huérfanos" ni referencias rotas. Es el comportamiento predeterminado en muchos SGBD.
ON UPDATE CASCADE	Cuando se actualiza un valor de clave primaria en la tabla padre, el valor de la clave foránea correspondiente en la tabla hija se actualiza automáticamente.	Mantiene las relaciones sincronizadas cuando los valores de la clave primaria cambian.
ON UPDATE RESTRICT /	Impide la actualización de un valor de clave	Previene la creación de referencias rotas

ON UPDATE NO ACTION	primaria en la tabla padre si existen filas que la referencian en la tabla hija.	por actualizaciones.

5.4. UNIQUE: Garantizando la Unicidad de Valores

La restricción UNIQUE garantiza que todos los valores en una columna o conjunto de columnas sean únicos.[18] A diferencia de una clave primaria, una tabla puede tener múltiples restricciones UNIQUE y las columnas con esta restricción pueden contener un valor NULL (solo uno).[34]

```
CREATE TABLE usuario (
  usuario_id   INT PRIMARY KEY,
  email        VARCHAR(255) UNIQUE,          -- cada email debe ser único
  username     VARCHAR(100),
  CONSTRAINT uq_username UNIQUE (username)   -- restricción UNIQUE con nombre
explícito
);
```

5.5. NOT NULL: Asegurando la Obligatoriedad de los Datos

La restricción NOT NULL es una de las más sencillas y directas. Asegura que una columna no pueda contener valores nulos.[18] Se usa para campos obligatorios, como un nombre o una dirección de correo electrónico.

```
CREATE TABLE cliente (
  cliente_id   INT PRIMARY KEY,
  nombre       VARCHAR(100) NOT NULL,   -- obligatorio
  email        VARCHAR(255) NULL        -- opcional
);
```

5.6. DEFAULT: Asignando Valores Predeterminados

La restricción DEFAULT asigna un valor por defecto a una columna si no se especifica un valor explícitamente durante una operación de inserción.[18] El valor predeterminado puede ser un literal (como un número o una cadena de texto) o una expresión o función (como CURRENT_DATE para la fecha actual).[38]

```
CREATE TABLE producto (
  producto_id  INT PRIMARY KEY,
  nombre       VARCHAR(100) NOT NULL,
  precio       DECIMAL(10,2) DEFAULT 0.00,
  creado_en    DATE DEFAULT CURRENT_DATE
);
```

5.7. CHECK: Limitando el Rango de Valores Aceptables

La restricción CHECK limita los valores que pueden ser insertados en una columna basándose en una condición booleana.[18] Es una herramienta poderosa para aplicar reglas de negocio específicas. Por ejemplo, se puede utilizar para asegurar que la edad de un usuario sea mayor de 18 años o que el precio de un producto sea positivo.[40]

```
CREATE TABLE empleado (
  empleado_id  INT PRIMARY KEY,
  nombre       VARCHAR(100) NOT NULL,
  edad         INT CHECK (edad >= 18),
  salario      DECIMAL(10,2),
  CONSTRAINT ck_salario_pos CHECK (salario > 0)
);
```

Tabla de Resumen de Restricciones DDL

Restricción	Propósito	Ejemplo de Sintaxis
PRIMARY KEY	Identifica de forma única cada fila; no admite NULL ni duplicados.	id INT PRIMARY KEY
		CONSTRAINT pk_id PRIMARY KEY (id)
FOREIGN KEY	Establece una relación entre dos tablas, referenciando una clave externa.	FOREIGN KEY (id_cliente) REFERENCES clientes(id)
UNIQUE	Garantiza la unicidad de todos los valores en una columna; permite un único valor NULL.	email VARCHAR(100) UNIQUE
		CONSTRAINT uq_email UNIQUE (email)
NOT NULL	Impide que una columna acepte valores NULL.	nombre VARCHAR(100) NOT NULL
DEFAULT	Asigna un valor predeterminado a la columna si no se especifica uno.	stock INT DEFAULT 0

CHECK	Limita los valores que una columna puede aceptar según una condición.	edad INT CHECK (edad > 18)
		CONSTRAINT ck_edad CHECK (edad > 18)

Las restricciones de integridad de datos son la manifestación sintáctica de las reglas de negocio y los principios de diseño de bases de datos.

No son simples palabras clave, sino la codificación de la lógica del negocio. Por ejemplo, el principio de la integridad de la entidad, que establece que la clave primaria no puede tener valores nulos [31], se implementa directamente con la sintaxis de PRIMARY KEY, que es una combinación de las propiedades NOT NULL y UNIQUE.[33] De manera similar, la integridad referencial, que exige que un valor de clave foránea debe corresponder a un valor de clave primaria existente en la tabla padre [31], se hace cumplir a través de la restricción FOREIGN KEY.[25] La restricción CHECK, por su parte, permite aplicar condiciones específicas del negocio, como asegurar que los datos de una columna cumplan un rango o condición predefinida, lo que valida la información a nivel del esquema.[40] En esencia, estas sentencias DDL no solo definen la estructura, sino que imponen un contrato de datos que asegura que el sistema se adhiera a su diseño lógico.

6: Mejores Prácticas en el Diseño de Esquemas de Bases de Datos

El diseño de un esquema de base de datos no se limita a la sintaxis DDL; es un proceso que requiere una comprensión profunda de los principios de diseño para construir sistemas escalables, eficientes y fáciles de mantener.

6.1. Principios de Normalización (1NF, 2NF, 3NF)

La normalización es un proceso sistemático para reducir la redundancia de datos y mejorar la integridad.[31] El proceso busca aplicar un conjunto de reglas a las relaciones lógicas para minimizar los problemas de actualización y proteger la coherencia de los datos.[31] Las tres primeras formas normales son los principios más utilizados en el diseño de bases de datos relacionales:

- **Primera Forma Normal (1NF):** Requiere que todos los valores de las columnas sean atómicos, lo que significa que no se deben incluir grupos repetitivos dentro de una misma fila.[31]
- **Segunda Forma Normal (2NF):** Se basa en la 1NF y exige que cada columna no clave dependa funcionalmente de la clave primaria completa.[31]
- **Tercera Forma Normal (3NF):** Se basa en la 2NF y elimina las dependencias transitivas, es decir, cuando un atributo no clave depende de otro atributo no clave.[31]

El proceso de normalización es el modelo lógico que luego se traduce en un esquema físico mediante sentencias DDL, donde cada forma normal tiene su contraparte en la definición de claves y relaciones.

6.2. Convenciones de Nomenclatura para Tablas y Columnas

Una nomenclatura clara y consistente es fundamental para la mantenibilidad y la legibilidad de un esquema.[42] Se recomienda el uso de convenciones como **snake_case** (ej. nombre_cliente) o **camelCase** (ej. nombreCliente) de manera uniforme.[42] Una práctica común es nombrar las tablas en plural (ej. clientes) y las columnas en singular (ej. nombre).[43]

```
-- Por convención se va a utilizar tabla en singular, y columnas en singular
CREATE TABLE cliente (
  cliente_id     INT PRIMARY KEY,
  nombre         VARCHAR(100) NOT NULL,
  apellido       VARCHAR(100) NOT NULL,
  email          VARCHAR(150) UNIQUE NOT NULL,
  fecha_alta     DATE DEFAULT CURRENT_DATE
);

CREATE TABLE pedido (
  pedido_id      INT PRIMARY KEY,
  fecha          DATE NOT NULL,
  monto_total    DECIMAL(10,2) CHECK (monto_total >= 0),
```

```
  cliente_id     INT NOT NULL,
  CONSTRAINT fk_pedido_cliente
    FOREIGN KEY (cliente_id) REFERENCES cliente(cliente_id)
);
```

6.3. La Importancia de las Claves Subrogadas vs. Claves Naturales

En el diseño de bases de datos, se distinguen dos tipos de claves primarias:

- **Claves Naturales:** Derivadas de datos del mundo real que ya existen y son únicos (ej. un DNI o una dirección de correo electrónico).[33]
- **Claves Subrogadas (o artificiales):** Son valores únicos generados por el sistema (ej. un id autoincrementable) que no tienen significado fuera de la base de datos.[33]

La recomendación general es utilizar claves subrogadas.[33] Esto se debe a que son estables y no cambian, a diferencia de una clave natural como un correo electrónico, que podría ser actualizado. Si un valor de clave natural se utiliza como clave primaria y luego cambia, cualquier clave foránea que la referencie en otras tablas también debe ser actualizada, lo que constituye una operación costosa y propensa a errores.[25] Las claves subrogadas, al ser inmutables, evitan este problema, lo que garantiza la estabilidad y la integridad de las relaciones a largo plazo.

```
-- Clave Natural
-- Usar DNI como clave primaria
CREATE TABLE persona (
  dni          CHAR(8) PRIMARY KEY,     -- clave natural
  nombre       VARCHAR(100) NOT NULL,
  apellido     VARCHAR(100) NOT NULL,
  email        VARCHAR(150) UNIQUE
);

CREATE TABLE cuenta_bancaria (
  nro_cuenta   INT PRIMARY KEY,
  dni          CHAR(8) NOT NULL,
  saldo        DECIMAL(12,2) DEFAULT 0,
  CONSTRAINT fk_cuenta_persona
    FOREIGN KEY (dni) REFERENCES persona(dni)
);
-- Problema: si el DNI se corrige o cambia, debe actualizarse en todas las tablas
relacionadas.
```

```sql
-- Clave Subrogada
-- Usar un id autoincremental como clave primaria
CREATE TABLE persona (
  persona_id   INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY, -- subrogada
  dni          CHAR(8) UNIQUE,   -- se mantiene como dato, no PK
  nombre       VARCHAR(100) NOT NULL,
  apellido     VARCHAR(100) NOT NULL,
  email        VARCHAR(150) UNIQUE
);

CREATE TABLE cuenta_bancaria (
  cuenta_id    INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  persona_id   INT NOT NULL,
  saldo        DECIMAL(12,2) DEFAULT 0,
  CONSTRAINT fk_cuenta_persona
    FOREIGN KEY (persona_id) REFERENCES persona(persona_id)
);
-- Ventaja: si el DNI cambia, no afecta la clave primaria ni las relaciones.
```

6.4. Estrategias para la Definición de Claves y Restricciones

- **Claves Primarias:** Cada tabla debe tener una clave primaria para identificar de forma única sus registros.[32] Siempre y en cuanto cumpla con la integridad y reglas de negocio, lo ideal es que sea un entero autoincremental para una mejor eficiencia de indexación.[33]
- **Valores Nulos:** Se debe aplicar la restricción NOT NULL de manera predeterminada a la mayoría de las columnas, a menos que el dato sea explícitamente opcional, según se establezca en el modelo de datos.[37]
- **Nomenclatura de Restricciones:** Es una buena práctica nombrar las restricciones de forma explícita (ej. CONSTRAINT <nombre_restricción>).[40] Esto facilita su gestión futura, como la eliminación o modificación con ALTER TABLE, ya que se pueden referenciar por su nombre en lugar de depender de un nombre generado automáticamente por el sistema.[40]

6.5. Consideraciones sobre el Rendimiento y la Mantenibilidad

El diseño del esquema tiene un impacto directo en el rendimiento y la mantenibilidad de la base de datos.

- **Indexación:** Las restricciones de PRIMARY KEY y UNIQUE crean automáticamente índices subyacentes que aceleran las operaciones de búsqueda y unión (join).[33]
- **Tipos de Datos:** La correcta elección de tipos de datos, como el uso de VARCHAR en lugar de

CHAR para datos de longitud variable, minimiza el tamaño de la fila en disco.[16] Una menor cantidad de datos por fila reduce la E/S de disco y la memoria necesaria para procesar las consultas, lo que, en consecuencia, mejora el rendimiento general del sistema.

Las mejores prácticas en DDL no son un conjunto de reglas arbitrarias; son el resultado de la experiencia de la industria en la resolución de problemas de escalabilidad y mantenibilidad.

Por ejemplo, el uso recomendado de claves subrogadas para las claves primarias es una solución pragmática a un problema real de mutabilidad de los datos en entornos de producción. Si se utiliza una clave natural (como un email o DNI), cualquier cambio en este valor requeriría una cascada de actualizaciones en todas las tablas que la referencian, lo que podría desestabilizar el sistema y consumir una cantidad significativa de recursos. La elección de una clave subrogada, que es inmutable, garantiza la estabilidad de las relaciones de datos y el rendimiento del sistema a largo plazo. Este tipo de decisiones de diseño, que se implementan con sentencias DDL, son las que diferencian un esquema robusto de uno que es propenso a errores y difícil de mantener.

Conclusiones

El Lenguaje de Definición de Datos (DDL) ha demostrado que es más que un simple conjunto de comandos para crear y eliminar objetos, define la estructura y las reglas de una base de datos, lo que garantiza la integridad, la portabilidad y la coherencia de los datos en un sistema. Conclusiones más relevantes:

1. **DDL como un Lenguaje de Abstracción:** El papel de DDL se distingue de DML por su enfoque en la estructura (schema) en lugar de en el contenido (data), lo que explica la naturaleza irreversible y auto-commit de sus operaciones.
2. **La Relevancia del Estándar:** El estándar ANSI SQL es esencial para la interoperabilidad y la portabilidad del código, lo que fomenta la colaboración y el comercio en la industria tecnológica. Al mismo tiempo, las extensiones de los proveedores son un reflejo natural de la necesidad de innovación y especialización.
3. **Las Sentencias DDL como Herramientas de Diseño:** Los comandos CREATE, ALTER y DROP no son solo sentencias sintácticas, sino que representan las fases del ciclo de vida de un esquema de base de datos, desde su concepción hasta su evolución y eventual eliminación.
4. **Las Restricciones como Implementación de la Lógica del Negocio:** Las restricciones de integridad de datos (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, DEFAULT, CHECK) son la materialización de las reglas de negocio y los principios de normalización. Su correcta aplicación es la herramienta más poderosa para prevenir errores y mantener la coherencia de los datos.
5. **El Diseño Lógico como Precursor del Diseño Físico:** Las mejores prácticas de diseño, como la normalización y la elección de claves subrogadas, son soluciones pragmáticas a problemas de escalabilidad y mantenibilidad en el mundo real. La implementación de estas prácticas a través de DDL es lo que asegura que un esquema no solo sea funcional, sino que también sea eficiente y fácil de gestionar a largo plazo.

Bibliografías citadas

1. lenguaje de definición de datos (DDL) - Db2 for i SQL - IBM, fecha de acceso: septiembre 3, 2025, https://www.ibm.com/docs/es/i/7.5.0?topic=programming-data-definition-language
2. Creating tables for ANSI SQL - SQL Server to Aurora PostgreSQL Migration Playbook, fecha de acceso: septiembre 3, 2025, https://docs.aws.amazon.com/dms/latest/sql-server-to-aurora-postgresql-migration-playbook/chap-sql-server-aurora-pg.sql.tables.html
3. Creating tables for ANSI SQL - SQL Server to Aurora MySQL Migration Playbook, fecha de acceso: septiembre 3, 2025, https://docs.aws.amazon.com/dms/latest/sql-server-to-aurora-mysql-migration-playbook/chap-sql-server-aurora-mysql.sql.creatingtables.html
4. What are DDL, DML, and DCL in SQL? | Scaler Topics, fecha de acceso: septiembre 3, 2025, https://www.scaler.com/topics/ddl-dml-dcl/
5. www.ibm.com, fecha de acceso: septiembre 3, 2025, https://www.ibm.com/docs/es/i/7.5.0?topic=programming-data-definition-language#:~:text=El%20lenguaje%20de%20definici%C3%B3n%20de,%2C%20m%C3%A1scaras%2C%20permisos%20y%20alias.&text=Un%20esquema%20proporciona%20una%20agrupaci%C3%B3n%20l%C3%B3gica%20de%20objetos%20SQL.
6. Structuring databases with DDL - dbt Labs, fecha de acceso: septiembre 3, 2025, https://www.getdbt.com/blog/structuring-databases-with-ddl
7. ¿Qué es DDL? Qué significa DML, DCL y TCL + Integridad Referencial - Platzi, fecha de acceso: septiembre 3, 2025, https://platzi.com/blog/que-es-ddl-dml-dcl-y-tcl-integridad-referencial/
8. SQL Commands: DDL, DQL, DML, DCL and TCL With Examples - GeeksforGeeks, fecha de acceso: septiembre 3, 2025, https://www.geeksforgeeks.org/sql/sql-ddl-dql-dml-dcl-tcl-commands/
9. Lenguaje de definición de datos o DDL - thedataschools.com, fecha de acceso: septiembre 3, 2025, https://thedataschools.com/que-es/lenguaje-de-definicion-de-datos-ddl.html
10. SQL - Wikipedia, fecha de acceso: septiembre 3, 2025, https://en.wikipedia.org/wiki/SQL
11. What is ANSI SQL and why it matters - CelerData, fecha de acceso: septiembre 3, 2025, https://celerdata.com/glossary/ansi-sql
12. Todo lo que necesita saber sobre la norma ANSI - Garantell, fecha de acceso: septiembre 3, 2025, https://www.garantell.com/es/blog/todo-lo-que-necesita-saber-sobre-la-norma-ansi
13. ¿Qué es la normalización? - Gob MX, fecha de acceso: septiembre 3, 2025, https://www.gob.mx/cms/uploads/attachment/file/12966/temas_de_interes_3.pdf
14. ALTER TABLE (Transact-SQL) - SQL Server - Microsoft Learn, fecha de acceso: septiembre 3, 2025, https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver17
15. Utilizar CREATE TABLE - IBM, fecha de acceso: septiembre 3, 2025, https://www.ibm.com/docs/es/informix-servers/12.10.0?topic=ssgu8g-12-1-0-com-ibm-ddi-doc-ids-ddi-083-htm
16. Tipo de datos SQL: ¿cuáles son y cómo utilizarlos? - The Information Lab, fecha de acceso: septiembre 3, 2025, https://www.theinformationlab.es/blog/tipo-de-datos-sql/
17. Tipos de datos (PL/SQL) - IBM, fecha de acceso: septiembre 3, 2025, https://www.ibm.com/docs/es/db2/12.1.0?topic=plsql-data-types
18. Integridad de Datos y Restricciones en SQL, fecha de acceso: septiembre 3, 2025, https://urianviera.com/sql/integridad-de-datos-y-restricciones
19. AS SELECT clause of the CREATE TABLE statement - IBM, fecha de acceso: septiembre 3, 2025, https://www.ibm.com/docs/SSGU8G_12.1.0/com.ibm.sqls.doc/ids_sqs_0403.htm
20. CREATE TABLE AS SELECT (CTAS) - Azure Synapse Analytics | Microsoft Learn, fecha de acceso: septiembre 3, 2025, https://learn.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-develop-ctas
21. CREATE TABLE LIKE - Azure Databricks, fecha de acceso: septiembre 3, 2025, https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/sql-ref-syntax-ddl-create-table-like
22. SQL ALTER TABLE - GeeksforGeeks, fecha de acceso: septiembre 3, 2025, https://www.geeksforgeeks.org/sql/sql-alter-add-drop-modify/
23. Sentencia ALTER TABLE en SQL [Sintaxis y ejemplos], fecha de acceso: septiembre 3, 2025, https://sqlearning.com/es/creacion-gestion-bases-datos/alter-table/
24. Creating and modifying constraints - IBM, fecha de acceso: septiembre 3, 2025, https://www.ibm.com/docs/en/ias?topic=constraints-creating-modifying
25. 13.1.18.5 FOREIGN KEY Constraints - Oracle Help Center, fecha de acceso: septiembre 3, 2025, https://docs.oracle.com/cd/E17952_01/mysql-5.7-en/create-table-foreign-keys.html
26. Oracle Drop Constraint from a Oracle Database Table via the Alter Table Command - RazorSQL, fecha de acceso: septiembre 3, 2025, https://razorsql.com/features/oracle_drop_constraint.html
27. DROP CONSTRAINT Clause - IBM, fecha de acceso: septiembre 3, 2025,

https://www.ibm.com/docs/en/informix-servers/12.10.0?topic=statement-drop-constraint-clause
28. DROP TABLE (Transact-SQL) - SQL Server - Microsoft Learn, fecha de acceso: septiembre 3, 2025, https://learn.microsoft.com/en-us/sql/t-sql/statements/drop-table-transact-sql?view=sql-server-ver17
29. DROP TABLE statement - IBM, fecha de acceso: septiembre 3, 2025, https://www.ibm.com/docs/SSGU8G_12.1.0/com.ibm.sqls.doc/ids_sqs_0736.htm
30. Tipos de datos equivalentes de ANSI SQL | Microsoft Learn, fecha de acceso: septiembre 3, 2025, https://learn.microsoft.com/es-es/office/client-developer/access/desktop-database-reference/equivalent-ansi-sql-data-types
31. Normalización de bases de datos - Wikipedia, la enciclopedia libre, fecha de acceso: septiembre 3, 2025, https://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos
32. SQL PRIMARY KEY - Syntax, Use Cases, and Examples | Hightouch, fecha de acceso: septiembre 3, 2025, https://hightouch.com/sql-dictionary/sql-primary-key
33. Clave principal SQL: Tutorial técnico completo - DataCamp, fecha de acceso: septiembre 3, 2025, https://www.datacamp.com/es/tutorial/sql-primary-key
34. constraint - Oracle Help Center, fecha de acceso: septiembre 3, 2025, https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/constraint.html
35. Primary Key Constraint - CockroachDB, fecha de acceso: septiembre 3, 2025, https://www.cockroachlabs.com/docs/stable/primary-key
36. Create Foreign Key Relationships - SQL Server - Microsoft Learn, fecha de acceso: septiembre 3, 2025, https://learn.microsoft.com/en-us/sql/relational-databases/tables/create-foreign-key-relationships?view=sql-server-ver17
37. Restricciones en SQL Server: SQL NOT NULL, UNIQUE y SQL PRIMARY KEY - SQLShack, fecha de acceso: septiembre 3, 2025, https://www.sqlshack.com/es/restricciones-en-sql-server-sql-not-null-unique-y-sql-primary-key/
38. Especifica los valores predeterminados de la columna | BigQuery - Google Cloud, fecha de acceso: septiembre 3, 2025, https://cloud.google.com/bigquery/docs/default-values?hl=es-419
39. DEFAULT clause of CREATE TABLE - HCL Product Documentation, fecha de acceso: septiembre 3, 2025, https://help.hcl-software.com/onedb/1.0.1/sqs/ids_sqs_0101.html
40. SQL CHECK - The Data Schools, fecha de acceso: septiembre 3, 2025, https://thedataschools.com/sql/check.html
41. Cómo usar la restricción CHECK de SQL Server para cumplir una condición booleana, fecha de acceso: septiembre 3, 2025, https://estradawebgroup.com/Post/Como-usar-la-restriccion-CHECK-de-SQL-Server-para-cumplir-una-condicion-booleana/20393
42. Guía Completa de Comandos DDL en SQL - Claude, fecha de acceso: septiembre 3, 2025, https://claude.ai/public/artifacts/105ff1ba-247e-4dd8-928b-ece48ddd16a7
43. Db2 12 - convenciones de nomenclatura en SQL - IBM, fecha de acceso: septiembre 3, 2025, https://www.ibm.com/docs/es/db2-for-zos/12.0.0?topic=elements-naming-conventions
44. learn.microsoft.com, fecha de acceso: septiembre 3, 2025, https://learn.microsoft.com/es-es/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-table-constraints#:~:text=de%20SQL%20dedicado.-,Comentarios,restricci%C3%B3n%20%C3%BAnica%20deben%20ser%20%C3%BAnicos.