WIKIPEDIA
La enciclopedia libre

SQL

SQL (por sus siglas en inglés **Structured Query Language**; en español **lenguaje de consulta estructurada**) es un lenguaje específico de dominio, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.[2] Una de sus principales características es el manejo del álgebra y el cálculo relacional para efectuar consultas con el fin de recuperar, de forma sencilla, información de bases de datos, así como realizar cambios en ellas.

Originalmente basado en el álgebra relacional y en el cálculo relacional, SQL consiste en un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos. El alcance de SQL incluye la inserción de datos, consultas, actualizaciones y borrado, la creación y modificación de esquemas y el control de acceso a los datos. También el SQL a veces se describe como un lenguaje declarativo, también incluye elementos procesales.

SQL fue uno de los primeros lenguajes comerciales para el modelo relacional de Edgar Frank Codd como se describió en su artículo de investigación de 1970 *El modelo relacional de datos para grandes bancos de datos compartidos*. A pesar de no adherirse totalmente al modelo relacional descrito por Codd, pasó a ser el lenguaje de base de datos más usado.

SQL pasó a ser el estándar del Instituto Nacional Estadounidense de Estándares (ANSI) en 1986 y de la Organización Internacional de Normalización (ISO) en 1987. Desde entonces, el estándar ha sido revisado para incluir más características. A pesar de la existencia de ambos estándares, la mayoría de los códigos SQL no son completamente portables entre sistemas de bases de datos diferentes sin otros ajustes.

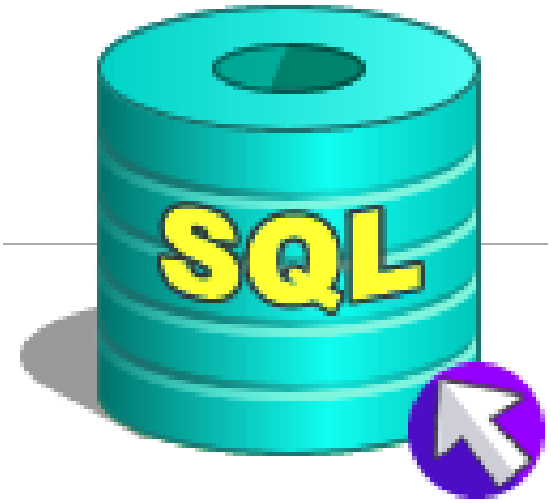
Orígenes y evolución

Los orígenes de SQL están ligados a las bases de datos relacionales, específicamente las que residían en máquinas IBM bajo el sistema de gestión System R, desarrollado por un grupo de la IBM en San José, California.

Al principio fue IBM, e IBM creó SQL. SQL, originalmente un acrónimo de "Lenguaje de consulta estructurado" ("Structured Query Language"), es un lenguaje unificado para definir, consultar, modificar y controlar los datos en una base de datos relacional. Su nombre se pronuncia oficialmente "ess-cue-ell" (según el American National Standards Institute).

El modelo relacional de gestión de bases de datos fue propuesto en 1970 por el Dr. E. F. Codd en el Laboratorio de Investigación de IBM en San José, California, y se desarrolló durante la década siguiente en universidades y laboratorios de investigación. SQL, uno de varios lenguajes que surgieron de este trabajo inicial, ahora se ha apoderado casi por completo del mundo de los lenguajes de bases de datos relacionales. Los proveedores de sistemas de administración de bases de datos relacionales que inicialmente eligieron otros lenguajes han acudido en masa a

SQL



Desarrollador(es)
IBM
https://www.iso.org/standard/76583.html

Información general

Extensiones comunes	sql
Paradigma	Multiparadigma
Apareció en	1974
Diseñado por	Donald D. Chamberlin Raymond F. Boyce
Última versión estable	SQL:2016 (2016)
Sistema de tipos	Estático, Fuerte
Implementaciones	Varias
Dialectos	SQL-85, SQL-88, SQL-91, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016
Influido por	Datalog
Ha influido a	Agena, CQL, LINQ, Windows PowerShell[1]
Sistema operativo	multiplataforma

SQL

Desarrollador
IBM
ISO/IEC 9075-1:2008 (http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498)

Información general

Extensión de archivo	`.sql`

SQL; Las organizaciones de normalización nacionales e internacionales han propuesto una versión codificada del lenguaje.

Tipo de MIME	`application/x-sql`
Lanzamiento inicial	1986
Última versión	SQL:2012 2012
Tipo de formato	Base de datos
Extendido de	lógica de primer orden
Estándar(es)	ISO/IEC 9075
Formato abierto	✓

En 1970, E. F. Codd propone el modelo relacional y asociado a este un sublenguaje de acceso a los datos basado en el cálculo de predicados.[3] Basándose en estas ideas, los laboratorios de IBM definieron el lenguaje SEQUEL (Structured English Query Language) que más tarde fue ampliamente implementado por el sistema de gestión de bases de datos (SGBD) experimental System R, desarrollado en 1977 también por IBM. Sin embargo, fue Oracle quien lo introdujo por primera vez en 1979 en un producto comercial.

El SEQUEL terminó siendo el predecesor de SQL, que es una versión evolucionada del primero. SQL pasa a ser el lenguaje por excelencia de los diversos sistemas de gestión de bases de datos relacionales surgidos en los años siguientes y fue por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, "SQL-86" o "SQL1". Al año siguiente este estándar es también adoptado por ISO.

Sin embargo, este primer estándar no cubría todas las necesidades de los desarrolladores e incluía funcionalidades de definición de almacenamiento que se consideró suprimirlas. Así que, en 1992, se lanzó un nuevo estándar ampliado y revisado de SQL llamado "SQL-92" o "SQL2".

En la actualidad SQL es el estándar *de facto* de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio.

El ANSI SQL sufrió varias revisiones y agregados a lo largo del tiempo:

Año	Nombre	Alias	Comentarios
1986	SQL-86	SQL-87	Primera publicación hecha por ANSI. Confirmada por la Organización Internacional de Normalización en 1987.
1989	SQL-89		Revisión menor.
1992	SQL-92	SQL2	Revisión mayor.
1999	SQL:1999	SQL2000	Se agregaron expresiones regulares, consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos.
2003	SQL:2003		Introduce algunas características de XML, cambios en las funciones, estandarización del objeto sequence y de las columnas autonuméricas.[4]
2006	SQL:2006		ISO/IEC 9075-14:2006 Define las maneras en las cuales SQL se puede utilizar conjuntamente con XML. Define maneras de importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML.
2008	SQL:2008		Permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursores. Incluye los disparadores del tipo INSTEAD OF. Añade la sentencia TRUNCATE.[5]
2011	SQL:2011		Datos temporales (PERIOD FOR). Mejoras en las funciones de ventana y de la cláusula FETCH.
2016	SQL:2016		Permite búsqueda de patrones, funciones de tabla polimórficas y compatibilidad con los ficheros JSON.

Características generales de SQL

SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones.[6]

Es un lenguaje declarativo de "alto nivel" o "de no procedimiento" que, gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros —y no a registros individuales— permite una alta productividad en codificación y la orientación a objetos. De esta forma, una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros. SQL también tiene las siguientes características:

- **Lenguaje de definición de datos**: El LDD de SQL proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación.
- **Lenguaje interactivo de manipulación de datos**: El LMD de SQL incluye lenguajes de consultas basado tanto en álgebra relacional como en cálculo relacional de tuplas.
- **Integridad**: El LDD de SQL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos.
- **Definición de vistas**: El LDD incluye comandos para definir las vistas.
- **Control de transacciones**: SQL tiene comandos para especificar el comienzo y el final de una transacción.
- **SQL incorporado y dinámico**: Esto quiere decir que se pueden incorporar instrucciones de SQL en lenguajes de programación como: C++, C, Java, PHP, COBOL, Pascal y Fortran.
- **Autorización**: El LDD incluye comandos para especificar los derechos de acceso a las relaciones y a las vistas.

Tipos de datos

Algunos de los tipos de datos básicos de SQL son:

Nùmeros enteros:

- **TYNYINT(tamaño):** -128 a 127 normal. 0 a 255 sin signo. La cantidad máxima de dígitos se puede especificar entre paréntesis
- **SMALLINT(tamaño):** -32768 a 32767 normal. 0 a 65535 sin signo. La cantidad máxima de dígitos se puede especificar entre paréntesis
- **MEDIUMINT(tamaño):** -8388608 a 8388607 normal. 0 a 16777215 sin signo. La cantidad máxima de dígitos se puede especificar entre paréntesis
- **INT(tamaño):** -2147483648 a 2147483647 normal. 0 a 4294967295 sin signo. La cantidad máxima de dígitos se puede especificar entre paréntesis
- **BIGINT(tamaño):** -9223372036854775808 a 9223372036854775807 normal. 0 a 18446744073709551615 sin signo. La cantidad máxima de dígitos se puede especificar entre paréntesis

Nùmeros en punto flotante:

- **FLOAT(tamaño, d):** Un pequeño número con un punto decimal flotante. La cantidad máxima de dígitos se puede especificar en el parámetro de tamaño. El número máximo de dígitos a la derecha del punto decimal se especifica en el parámetro d
- **DOBLE (tamaño, d)**: Un número grande con un punto decimal flotante. La cantidad máxima de dígitos se puede especificar en el parámetro de tamaño. El número máximo de dígitos a la derecha del punto decimal se especifica en el parámetro d
- **DECIMAL (tamaño, d)**: Un DOBLE almacenado como una cadena, lo que permite un punto decimal fijo. La cantidad máxima de dígitos se puede especificar en el parámetro de tamaño. El número máximo de dígitos a la derecha del punto decimal se especifica en el parámetro d

Fechas y tiempos

- **DATE ():** Una fecha. Formato: AAAA-MM-DD Nota: el rango admitido es de '1000-01-01' a '9999-12-31'
- **DATETIME ():** * Una combinación de fecha y hora. Formato: AAAA-MM-DD HH: MI: SS Nota: el rango admitido es de '1000-01-01 00:00:00' a '9999-12-31 23:59:59'
- **TIMESTAMP ():** * Una marca de tiempo. Los valores de TIMESTAMP se almacenan como el número de segundos desde la época de Unix ('1970-01-01 00:00:00' UTC). Formato: AAAA-MM-DD HH: MI: SS Nota: el rango admitido es de '1970-01-01 00:00:01' UTC a '2038-01-09 03:14:07' UTC
- **TIME ():** Un tiempo. Formato: HH: MI: SS Nota: el rango admitido es de '-838: 59: 59' a '838: 59: 59'
- **YEAR ():** Un año en formato de dos o cuatro dígitos. Nota: Valores permitidos en formato de cuatro dígitos: de 1901 a 2155. Valores permitidos en formato de dos dígitos: 70 a 69, que representan los años de 1970 a 2069

Cadena de caracteres:

- **CHAR (tamaño):** Tiene una cadena de longitud fija (puede contener letras, números y caracteres especiales). El tamaño fijo se especifica entre paréntesis. Puede almacenar hasta 255 caracteres
- **VARCHAR (tamaño):** Tiene una cadena de longitud variable (puede contener letras, números y caracteres especiales). El tamaño máximo se especifica entre paréntesis. Puede almacenar hasta 255 caracteres. Nota:

si agrega un valor mayor que 255, se convertirá en un tipo de texto
- **TINYTEXT:** Tiene una cadena con una longitud máxima de 255 caracteres
- **TEXTO:** Tiene una cadena con una longitud máxima de 65.535 caracteres
- **BLOB:** Para BLOB (Objetos grandes binarios). Almacena hasta 65.535 bytes de datos
- **MEDIUMTEXT:** Tiene una cadena con una longitud máxima de 16,777,215 caracteres
- **MEDIUMBLOB:** Para BLOB (Objetos grandes binarios). Tiene capacidad para 16.777.215 bytes de datos
- **LONGTEXT:** Tiene una cadena con una longitud máxima de 4.294.967.295 caracteres
- **LONGBLOB:** Para BLOB (Objetos grandes binarios). Tiene capacidad para 4.294.967.295 bytes de datos

Enum y set:

- **Enum (x, y, z, etc.):** Permite ingresar una lista de valores posibles. Puede enumerar hasta 65535 valores en una lista ENUM. Si se inserta un valor que no está en la lista, se insertará un valor en blanco. **Nota:** los valores se ordenan en el orden en que los ingresas. Ingrese los valores posibles en este formato: ENUM ('X', 'Y', 'Z')
- **Set:** Similar a ENUM, excepto que SET puede contener hasta 64 elementos de lista y puede almacenar más de una opción

Binarios:

- **bit:** Entero que puede ser 0, 1 o NULL

Optimización

Como ya se dijo antes, y suele ser común en los lenguajes de acceso a bases de datos de alto nivel, SQL es un lenguaje declarativo. O sea, que especifica qué es lo que se quiere y no cómo conseguirlo, por lo que una sentencia no establece explícitamente un orden de ejecución.

El orden de ejecución interno de una sentencia puede afectar seriamente a la eficiencia del SGBD, por lo que se hace necesario que este lleve a cabo una optimización antes de su ejecución. Muchas veces, el uso de índices acelera una instrucción de consulta, pero ralentiza la actualización de los datos. Dependiendo del uso de la aplicación, se priorizará el acceso indexado o una rápida actualización de la información. La optimización difiere sensiblemente en cada motor de base de datos y depende de muchos factores.

Los sistemas de bases de datos modernos poseen un componente llamado optimizador de consultas. Este realiza un detallado análisis de los posibles planes de ejecución de una consulta SQL y elige aquel que sea más eficiente para llevar adelante la misma.

Existe una ampliación de SQL conocida como FSQL (Fuzzy SQL, SQL difuso) que permite el acceso a bases de datos difusas, usando la lógica difusa. Este lenguaje ha sido implementado a nivel experimental y está evolucionando rápidamente.

Lenguaje de definición de datos (DDL)

El lenguaje de definición de datos (en inglés *Data Definition Language*, o *DDL*), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos de la base de datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE.[7]

CREATE (Crear)

Este comando permite crear objetos de datos, como nuevas bases de datos, tablas, vistas y procedimientos almacenados.

Ejemplo (crear una tabla)

```
CREATE TABLE clientes;
```

ALTER (Alterar)

Este comando permite modificar la estructura de una tabla u objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

Ejemplo (agregar columna a una tabla)

```
ALTER TABLE alumnos ADD edad INT UNSIGNED;
```

DROP (Eliminar)

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

Ejemplo

```
DROP TABLE alumnos;
```

TRUNCATE (Truncar)

Este comando solo aplica a tablas y su función es borrar el contenido completo de la tabla especificada. La ventaja sobre el comando DELETE, es que si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande. La desventaja es que TRUNCATE solo sirve cuando se quiere eliminar absolutamente todos los registros, ya que no se permite la cláusula WHERE. Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando TRUNCATE borra la tabla y la vuelve a crear y no ejecuta ninguna transacción.

Ejemplo

```
 TRUNCATE TABLE nombre_tabla;
```

Lenguaje de manipulación de datos DML (Data Manipulation Language)

El lenguaje de manipulación de datos (*Data Manipulation Language*, o *DML* en inglés) es el que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.[7] En SQL incluye órdenes para seleccionar, insertar, actualizar y borrar los datos. Existen cuatro operaciones básicas: SELECT, INSERT, UPDATE y DELETE.

SELECT (Seleccionar)

La sentencia **SELECT** nos permite consultar los datos almacenados en una tabla de la base de datos.

Forma básica

```
SELECT [{ALL|DISTINCT}]
    <nombre_campo>[, <nombre_campo>...]

FROM {<nombre_tabla>|<nombre_vista>}[,
    {<nombre_tabla>|<nombre_vista>}...]

[WHERE <condición> [{AND|OR} <condición>...]]

[GROUP BY <nombre_campo>[, <nombre_campo>...]]
```

```
[HAVING <condición> [{AND|OR} <condición>...]]

[ORDER BY {<nombre_campo>|<indice_campo>} [{ASC|DESC}][,
    {<nombre_campo>|<indice_campo>} [{ASC|DESC}]]];
```

	Palabra clave que indica que la sentencia de SQL que queremos ejecutar es una consulta. Selecciona tanto los campos que se enumeran como todos los registros que cumplan con la condición de la parte WHERE. Cuando los atributos se toman de distintas tablas en la parte FROM, también realiza la reunión (join). Por eso se dice que es un lenguaje ortogonal.
SELECT	Cuando se pone la palabra clave **ALL** Indica que queremos seleccionar todos los valores, es decir que genere un multiconjunto o bolsa en lugar de un conjunto. Es el valor por defecto y no suele especificarse casi nunca. Cuando se pone la palabra clave **DISTINCT** Indica que queremos seleccionar solo los valores distintos. El resultado es un conjunto en lugar de un multiconjunto o bolsa.
FROM	Indica la tabla (o tablas) desde la que queremos recuperar los datos. En el caso de que exista más de una tabla se denomina a la consulta "consulta combinada" o reunión "join". En las consultas combinadas es necesario aplicar una condición de combinación a través de una cláusula **WHERE**.
WHERE	Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Admite los operadores lógicos **AND** y **OR** además de los relacionales y otros.
GROUP BY	Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.
HAVING	Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de **WHERE** pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a **GROUP BY** y la condición debe estar referida a los campos contenidos en ella.
ORDER BY	Presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con **ASC** (orden ascendente) y **DESC** (orden descendente). El valor predeterminado es **ASC.**

Ejemplo:

Para formular una consulta a la tabla coches y recuperar los campos matrícula, marca, modelo, color, número_kilómetros, num_plazas debemos ejecutar la siguiente consulta. Los datos serán devueltos ordenados por marca y por modelo en orden ascendente, de menor a mayor. La palabra clave **FROM** indica que los datos serán recuperados de la tabla Coches.

```
SELECT
    matricula,
    marca,
    modelo,
    color,
    numero_kilometros,
    num_plazas
FROM
    coches
ORDER BY
    marca,
    modelo;
```

Ejemplo de consulta simplificada a través de un comodín de campos (*):

El uso del asterisco indica que queremos que la consulta devuelva todos los campos que existen en la tabla y los datos serán devueltos ordenados por marca y por modelo.

```
SELECT *
FROM
    coches
ORDER BY
    marca,
    modelo;
```

Cláusula WHERE (Donde)

La cláusula *WHERE* es la instrucción que nos permite filtrar el resultado de una sentencia **SELECT**. Habitualmente no deseamos obtener toda la información existente en la tabla, sino que queremos obtener solo la información que nos resulte útil en ese momento. La cláusula **WHERE** filtra los datos antes de ser devueltos por la consulta. Cuando en la Cláusula WHERE queremos incluir un tipo texto, debemos incluir el valor entre comillas simples.

Ejemplos:

En nuestro ejemplo, se desea consultar un coche en concreto, para esto se agregó una cláusula **WHERE**. Esta cláusula especifica una o varias condiciones que deben cumplirse para que la sentencia **SELECT** devuelva los datos. En este caso la consulta devolverá solo los datos del coche con matrícula para que la consulta devuelva solo los datos del coche con matrícula `MF-234-ZD` o bien la matrícula `FK-938-ZL` . Se puede utilizar la cláusula **WHERE** solamente, o en combinación con tantas condiciones como queramos.

```
SELECT
    matricula,
    marca,
    modelo,
    color,
    numero_kilometros,
    num_plazas
FROM
    coches
WHERE
    matricula = 'MF-234-ZD'
    OR matricula = 'FK-938-ZL';
```

Una condición WHERE puede ser negada a través del Operador Lógico NOT. La Siguiente consulta devolverá todos los datos de la tabla Coches, menos el que tenga la Matrícula `MF-234-ZD`.

```
SELECT
    matricula,
    marca,
    modelo,
    color,
    numero_kilometros,
    num_plazas
FROM
    coches
WHERE
    NOT matricula = 'MF-234-ZD';
```

La siguiente consulta utiliza la condicional **DISTINCT**, la cual nos devolverá la tabla generada al seleccionar únicamente los campos marca y modelo de cada registro, eliminando los renglones repetidos. Dicho de otra manera, el conjunto de modelos de cada marca que hay en coches.

```
SELECT DISTINCT marca, modelo FROM coches;
```

Si se omitese **DISTINCT** o se usase **ALL** la tabla generada tendría renglones repetidos porque pueden haber vaios coches de la misma marca y modelo pero con distinta matricula. En ese caso el resultado sería un multiconjunto donde cada renglón con los campos *marca y modelo* en la respuesta corresponde a alguna matricula de la tabla *coches* que no se muestra en la respuesta.

Cláusula ORDER BY (Ordernar por)

La cláusula *ORDER BY* es la instrucción que nos permite especificar el orden en el que serán devueltos los datos. Podemos especificar el orden de forma ascendente o descendente a través de las palabras clave **ASC** y **DESC**. El orden depende del tipo de datos que estén definidos en la columna, de forma que un campo numérico será ordenado como tal, y un alfanumérico se ordenará de la A a la Z, aunque su contenido sea numérico. El valor predeterminado es ASC si no se especifica al hacer la consulta.

Ejemplos:

```
SELECT
    matricula,
    marca,
```

```
    modelo,
    color,
    numero_kilometros,
    num_plazas
FROM
    coches
ORDER BY
    marca ASC,
    modelo DESC;
```

Este ejemplo, selecciona todos los campos matricula, marca, modelo, color, numero_kilometros y num_plazas de la tabla coches, ordenándolos por los campos marca y modelo, marca en forma ascendente y modelo en forma descendente.

```
SELECT
    matricula,
    marca,
    modelo,
    color,
    numero_kilometros,
    num_plazas
FROM
    coches
ORDER BY 2;
```

Este ejemplo, selecciona todos los campos matrícula, marca, modelo, color, numero_kilometros y num_plazas de la tabla coches, ordenándolos por el campo *marca*, ya que aparece en segundo lugar dentro de la lista de campos que componen la **SELECT.**

Subconsultas

Una subconsulta es una sentencia SELECT que está embebida en una cláusula de otra sentencia SQL. También pueden utilizarse subconsultas en los comandos INSERT, UPDATE, DELETE y en la cláusula FROM.[8]

Las subconsultas pueden resultar útiles si necesitas seleccionar filas de una tabla con una condición que depende de los datos de la propia tabla o de otra tabla.

La subconsulta (consulta interna), se ejecuta antes de la consulta principal; el resultado de la subconsulta es utilizado por la consulta principal (consulta externa).

```
SELECT c.matricula, c.modelo
FROM   coches AS c
WHERE  c.matricula IN
    (
        SELECT m.matricula
        FROM   multas AS m
        WHERE  m.importe > 100
    );
```

En este ejemplo, se seleccionan las matrículas y los modelos de los coches cuyas multas superan los u$s 100.

INSERT (Insertar)

Una sentencia *INSERT* de SQL agrega uno o más registros a una (y solo una) tabla en una base de datos relacional.

Forma básica

```
INSERT INTO
    tabla(columnaA, [columnaB, ...])
VALUES
    ('valor1', ['valor2', ...]);

-- O también se puede utilizar como:
INSERT INTO tabla VALUES ('valor1', 'valor2');
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia `INSERT` deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

Ejemplo

```
INSERT INTO agenda_telefonica (nombre, numero)
VALUES ('Roberto Jeldrez', 4886850);
```

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

```
INSERT INTO nombre_tabla VALUES ('valor1', ['valor2', ...]);
```

Ejemplo (asumiendo que 'nombre' y 'número' son las únicas columnas de la tabla 'agenda_telefonica'):

```
INSERT INTO agenda_telefonica
VALUES ('Johnny Aguilar', 080473968);
```

Formas avanzadas

Una característica de SQL (desde SQL-92) es el uso de *constructores de filas* para insertar múltiples filas a la vez, con una sola sentencia SQL:

```
INSERT INTO
    tabla(columna1[, columna2, ...])
VALUES
    ('valor1A', ['valor1B', ...]),
    ('value2A', ['value2B', ...]), ...;
```

Esta característica es soportada por DB2, PostgreSQL (desde la versión 8.2), MySQL, y H2.

Ejemplo (asumiendo que nombre y número son las únicas columnas en la tabla agenda_telefonica):

```
INSERT INTO
    agenda_telefonica
VALUES
    ('Roberto Fernández', '4886850'),
    ('Alejandro Sosa', '4556550');
```

Que podía haber sido realizado por las sentencias

```
INSERT INTO agenda_telefonica VALUES ('Roberto Fernández', '4886850');
INSERT INTO agenda_telefonica VALUES ('Alejandro Sosa', '4556550');
```

Notar que las sentencias separadas pueden tener semántica diferente (especialmente con respecto a los triggers), y puede tener diferente rendimiento que la sentencia de inserción múltiple.

Para insertar varias filas en MS SQL puede utilizar esa construcción:

```
INSERT INTO phone_book
SELECT 'John Doe', '555-1212'
UNION ALL
SELECT 'Peter Doe', '555-2323';
```

Tenga en cuenta que no se trata de una sentencia SQL válida de acuerdo con el estándar SQL (SQL: 2003), debido a la cláusula subselect incompleta.

Para hacer lo mismo en Oracle se usa la Tabla DUAL, siempre que se trate de solo una simple fila:

```
INSERT INTO phone_book
SELECT 'John Doe', '555-1212' FROM DUAL
```

```
UNION ALL
SELECT 'Peter Doe','555-2323' FROM DUAL
```

Una implementación conforme al estándar de esta lógica se muestra el siguiente ejemplo, o como se muestra arriba (no aplica en Oracle):

```
INSERT INTO phone_book
SELECT 'John Doe', '555-1212' FROM LATERAL ( VALUES (1) ) AS t(c)
UNION ALL
SELECT 'Peter Doe','555-2323' FROM LATERAL ( VALUES (1) ) AS t(c)
```

Copia de filas de otras tablas

Un INSERT también puede utilizarse para recuperar datos de otros, modificarla si es necesario e insertarla directamente en la tabla. Todo esto se hace en una sola sentencia SQL que no implica ningún procesamiento intermedio en la aplicación cliente. Un SUBSELECT se utiliza en lugar de la cláusula VALUES. El SUBSELECT puede contener la sentencia JOIN, llamadas a funciones, y puede incluso consultar en la misma TABLA los datos que se inserta. Lógicamente, el SELECT se evalúa antes que la operación INSERT esté iniciada. Un ejemplo se da a continuación.

```
INSERT INTO phone_book2

SELECT *
FROM phone_book
WHERE name IN ('John Doe', 'Peter Doe');
```

Una variación es necesaria cuando algunos de los datos de la tabla fuente se está insertando en la nueva tabla, pero no todo el registro. (O cuando los esquemas de las tablas no son iguales.)

```
INSERT INTO phone_book2 ([name], [phoneNumber])

SELECT [name], [phoneNumber]
FROM phone_book
WHERE name IN ('John Doe', 'Peter Doe');
```

El SELECT produce una tabla (temporal), y el esquema de la tabla temporal debe coincidir con el esquema de la tabla donde los datos son insertados.

UPDATE (Actualizar)

Una sentencia *UPDATE* de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Ejemplo

```
UPDATE My_table SET field1 = 'updated value' WHERE field2 = 'N';
```

DELETE (Borrar)

Una sentencia *DELETE* de SQL borra uno o más registros existentes en una tabla.

Forma básica

```
DELETE FROM tabla WHERE columna1 = 'valor1';
```

Ejemplo

```
DELETE FROM mi_tabla WHERE columna2 = 'N';
```

Recuperación de clave

Los diseñadores de base de datos que usan una clave suplente como la clave principal para cada tabla, se ejecutará en el ocasional escenario en el que es necesario recuperar automáticamente la base de datos, generando una clave primaria de una sentencia SQL INSERT para su uso en otras sentencias SQL. La mayoría de los sistemas no permiten sentencias SQL INSERT para retornar fila de datos. Por lo tanto, se hace necesario aplicar una solución en tales escenarios.

Implementaciones comunes incluyen:

- Utilizando un procedimiento almacenado específico de base de datos que genera la clave suplente, realice la operación INSERT, y finalmente devuelve la clave generada.
- Utilizando una sentencia SELECT especifica de base de datos, sobre una tabla temporal que contiene la última fila insertada. DB2 implementa esta característica de la siguiente manera:

```
SELECT *
FROM NEW TABLE (
    INSERT INTO phone_book
    VALUES ('Cristobal Jeldrez','0426.817.10.30')
) AS t
```

- Utilizando una sentencia SELECT después de la sentencia INSERT con función específica de base de datos, que devuelve la clave primaria generada por el registro insertado más recientemente.
- Utilizando una combinación única de elementos del original SQL INSERT en una posterior sentencia SELECT.
- Utilizando un GUID en la sentencia SQL INSERT y la recupera en una sentencia SELECT.
- Utilizando la función de PHP mysql_insert_id() de MySQL después de la sentencia INSERT.
- Utilizando un INSERT con la cláusula RETURNING para Oracle, que solo se puede utilizar dentro de un PL/SQL bloque, en el caso de PostgreSQL se puede usar también tanto con SQL como con PL/SQL.

```
INSERT INTO phone_book VALUES ('Cristobal Jeldrez', '0426.817.10.30')
RETURNING phone_book_id INTO v_pb_id
```

- En el caso de MS SQL se puede utilizar la siguiente instrucción:

```
Set NoCount On;
INSERT INTO phone_book VALUES ('Cristobal Jeldrez', '0426.817.10.30');
Select @@Identity as id
```

Disparadores

Los disparadores, también conocidos como desencadenantes (*triggers* en inglés) son definidos sobre la tabla en la que opera la sentencia INSERT, y son evaluados en el contexto de la operación. Los desencadenantes BEFORE INSERT permiten la modificación de los valores que se insertarán en la tabla. Los desencadenantes AFTER INSERT no puede modificar los datos de ahora en adelante, pero se puede utilizar para iniciar acciones en otras tablas, por ejemplo para aplicar mecanismos de auditoría Excel.

Sistemas de gestión de base de datos

Los sistemas de gestión de base de datos con soporte SQL más utilizados son, por orden alfabético:

- DB2
- Firebird
- HSQL
- Informix
- InterBase
- MariaDB
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- PervasiveSQL
- SQLite
- Sybase ASE

Interoperabilidad

El lenguaje de consultas de los diferentes sistemas de gestión de bases de datos son incompatibles entre ellos y no necesariamente siguen completamente el estándar. En particular, la sintaxis de fecha y tiempo, la concatenación de cadenas, nulas, y la comparación de textos en cuanto al tratamiento de mayúsculas y minúsculas varían de un proveedor a otro. Una excepción particular es PostgreSQL, que se esfuerza por lograr el cumplimiento del estándar.[9]

Las implementaciones populares de SQL omiten comúnmente soporte para funciones básicas de SQL estándar, como la de los tipos de dato `DATE` o `TIME`. Es el caso del manejador de bases de datos de Oracle (cuyo tipo `DATE` se comporta como `DATETIME`, y carece de un tipo `TIME`)[10] y MS SQL Server (antes de la versión de 2008). Como resultado, el código SQL rara vez puede ser portado entre los sistemas de base de datos sin modificaciones.

Hay varias razones para esta falta de portabilidad entre sistemas de bases de datos:

- La complejidad y el tamaño del estándar SQL conlleva a que la mayoría de las implementaciones de SQL no sean compatibles con la norma completa.
- La norma no especifica el comportamiento de la base de datos en varias áreas importantes (por ejemplo, índices, almacenamiento de archivos, etc.), dejando a las implementaciones decidir cómo comportarse.
- El estándar SQL especifica con precisión la sintaxis que un sistema de base de datos conforme debe implementar. Sin embargo, no está tan bien definida la especificación en el estándar de la semántica de las construcciones del lenguaje, lo que lleva a ambigüedad.
- Muchos proveedores de bases de datos tienen grandes bases de clientes existentes, por lo que introducir cambios para adaptarse el estándar podría producir incompatibilidades en las instalaciones de los usuarios y el proveedor puede no estar dispuesto a abandonar la compatibilidad con versiones anteriores.
- Hay poco incentivo comercial para que un proveedor facilite a los usuarios el cambiar de proveedor de bases de datos.
- Los usuarios que evalúan el software de base de datos tienden a valorar más otros factores tales como el rendimiento más alto en sus prioridades sobre las conformidad al estándar.

El estándar ODBC (Open Database Connectivity) permite acceder a la información desde cualquier aplicación independientemente del sistema de gestión de base de datos (DBMS) en el que esté almacenada la información, desacoplando así la aplicación de la base de datos.

Véase también

- AQL
- FSQL
- Lenguaje de definición de datos
- Modelo de base de datos