# 1. EL LENGUAJE SQL

SQL es un lenguaje que permite organizar, gestionar y recuperar datos almacenados en una base de datos relacional

El nombre **SQL** es una abreviación *de Structured Query Language* ( *Lenguaje estructurado de consulta)*.

El nombre lenguaje estructurado de consultas es realmente un término algo equivocado. SQL es mucho más que una herramienta de consulta, es un lenguaje completo de control e interacción con un SGBD. SQL se utiliza para controlar **todas las funciones que suministra un SGBD** a sus usuarios, incluyendo:

⇒ **Definición de datos**. SQL permite que un usuario defina la estructura y la organización de los datos almacenados, así como las relaciones existentes entre ellos.

⇒ **Recuperación de datos**. SQL permite a un usuario o programa recuperar y utilizar los datos almacenados en una base de datos.

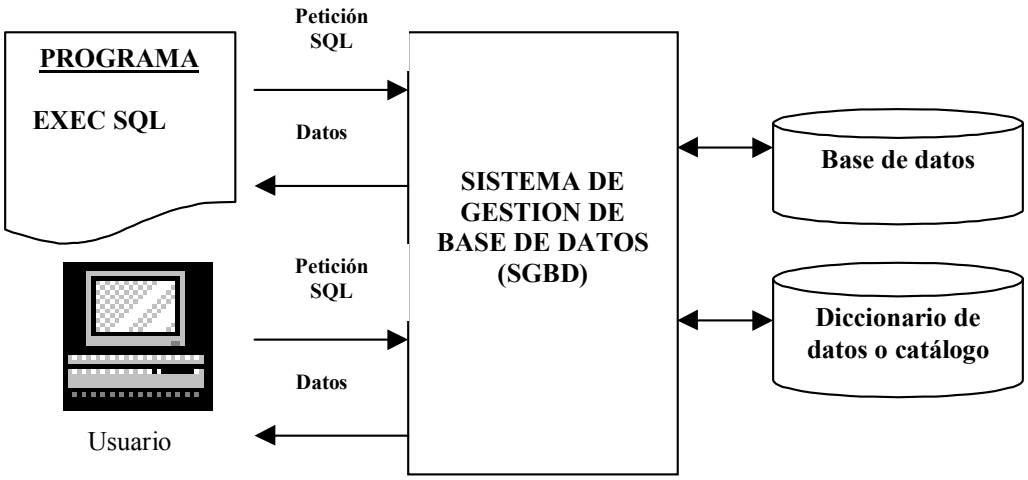
⇒ **Manipulación de datos**. SQL permite a un usuario a programa actualizar la base de datos, añadiendo datos nuevos, borrando los viejos y modificando los almacenados previamente.

⇒ **Control de acceso**. SQL puede ser utilizado para restringir la capacidad de un usuario para recuperar, añadir y modificar datos, protegiendo los datos almacenados contra accesos no autorizados.

⇒ **Compartición de datos**. SQL es utilizado para coordinar la compartición de datos entre usuarios concurrentes, asegurando que no haya interferencias entre ellos.

⇒ **Integridad de datos**. SQL define restricciones de integridad en la base de datos, protegiéndola de alteraciones debidas a actualizaciones inconsistentes o fallos del sistema.

# 2. COMO SE USA EL SQL.



El SGBD (Sistema Gestor de Bases de Datos) es el programa encargado de manejar y gestionar los datos de la base Cuando es necesario recuperar datos de una base de datos, la petición se realiza utilizando SQL. El SGBD procesa la petición SQL, recoge los datos solicitados y los devuelve a quien los solicitó. Este proceso petición de datos a la base de datos y posterior recepción de resultados se llama **consulta (Query)**.

El SGBD permite que los usuarios y programadores vean los datos en estructuras lógicas independientes de las estructuras físicas en las que se encuentran almacenados (esto libera de los detalles relativos al manejo de punteros, ficheros, etc.. ). En el caso de un SGBD RELACIONAL las estructura lógicas que ve el programador o el usuario son TABLAS.

Un SGBD RELACIONAL debe proporcionar un lenguaje para operar con los datos estructurados en las tablas. Este lenguaje es el SQL

***TODAS LAS OPERACIONES DE CONTROL E INTERACCIÓN CON LA BASE DE DATOS SE REALIZAN EN SQL***

Las sentencias que se construyan en este lenguaje se envían al SGBD, que las analiza y traduce a operaciones sobre los ficheros físicos. Una vez ejecutadas estas operaciones, el gestor devuelve al programa o usuario el resultado. El SGBD se encarga igualmente de responder a una eventual petición por parte de múltiples usuarios, coordinándolas para asegurar la integridad de los datos.

El **catálogo o diccionario de datos** está formado por unas tablas prediseñadas en las que el SGBD almacena las descripciones de todas las tablas incluidas en cada momento en la base de datos. La información del catálogo es actualizada automáticamente por el gestor cuando se añade una nueva tabla o se modifica la descripción de una existente. También se puede utilizar el lenguaje SQL para permitir a los usuarios y programas acceder al catálogo para consultar precisamente dichas descripciones.

Las peticiones sobre los datos se expresan en SQL mediante sentencias que deben escribirse de acuerdo a las reglas sintácticas y semánticas de este lenguaje. Estas peticiones se pueden realizar:

⇒ **De forma interactiva**. Los usuarios escriben órdenes SQL directamente en la pantalla de un terminal interactivo, en la cual se recibe el resultado de la petición expresada en ellas.

⇒ **SQL embebido en un programa**: Los programadores introducen órdenes SQL en sus programas ( escritos en C, php, VisualBasic, etc..) para acceder a los datos de la base de dato. Tanto los progarmas escritos por los usuarios, como las utilidades de base de datos (tales como generadores de informes y formularios etc..) utilizan esta técnica para acceder a la base de datos.

# 3. SUBCONJUNTOS DE ANSI-SQL

Si bien el uso principal de ANSI-SQL es la realización de consultas y actualizaciones sobre datos almacenados en bases relacionales, el lenguaje también permite realizar otras tareas. La clasificación de estas tareas permite hacer lo propio con el lenguaje, en los subconjuntos siguientes:

❖ **Consulta de datos (Data Query Language)**

Contiene aquellas sentencias que permiten obtener datos de las tablas y especificar la forma en la que deseamos que se presenten. La sentencia más relevante de este subconjunto del lenguaje es **SELECT**. Esta sentencia permite extraer datos de una o varias tablas .

❖ **Manipulación de datos (Data Manipulation Language, DML)**

Están incluidas en este grupo las sentencias que permiten modificar, añadir y borrar filas en las tablas de las bases de datos. Entre estas sentencias se incluyen **INSERT**, **UPDATE** Y **DELETE**.

- **INSERT**: Permite añadir una o varias filas a una tabla.
- **UPDATE**: Permite modificar valores de una o más filas de una tabla.
- **DELETE**: Permite borrar una o más filas de una tabla.

❖ **Definición de datos (Data Definition Language, DDL)**

Permite definir nuevos objetos ( tablas, índices, etc.), modificar las definiciones o destruir los objetos existentes mediante las sentencias **CREATE**, **ALTER** Y **DROP**, así como establecer restricciones para los campos de las tablas (**NOT NULL**, **CHECK** y **CONSTRAINT**), y establecer relaciones entre tablas (**PRIMARY KEY**, **FOREIGN KEY** y **REFERENCES**).

Una tabla debe ser definida antes de poder ser utilizada. Otros muchos objetos son manejados por un SGBD, por ejemplo, espacios físicos, índices, etc. Todas estas definiciones se realizan con sentencias SQL.

- **CREATE**: Utilizado para crear nuevas tablas e índices
- **ALTER**: Utilizado para modificar tablas, por ejemplo, añadiendo nuevos campos o modificando las definiciones existentes.
- **DROP**: Utilizado para eliminar tablas o índices de la base de datos.

❖ **Control de datos (Data Control Language, DCL)**

Permite especificar restricciones de seguridad de los datos: por ejemplo, los datos sean accesibles a unos usuarios o a otros no, las acciones que un determinado usuario pueda realizar sobre los datos (consultar, modificar, borrar etc.).

Así, las sentencias **GRANT** Y **REVOKE** permiten conceder y retirar la autorización para el acceso a un usuario a una tabla.

Un usuario no podrá consultar o actualizar datos de una tabla si previamente no posee permisos para ello.

❖ **Procesamiento de transacciones (Transaction-processing Language, TPL)**

Estas sentencias permiten controlar que una serie de comandos pertenecientes al DML, es decir, que manipulan datos, sean ejecutadas de manera coherente. Sentencias incluidas en este grupo del lenguaje SQL son **BEGIN TRANSACTION**, **COMMIT** y **ROLLBACK**.

❖ **Control de cursores (Cursor-control Language, CCL)**

Este subconjunto del lenguaje se ocupa de operar sobre filas individuales de una tabla resultado que consta de varios registros. Sentencias de este grupo

Son **DECLARE CURSOR**, **FETCH INTO**, **UPDATE WHERE CURRENT**.

| Sentencia | Descripción |
|---|---|
| *Manipulación de datos* | |
| SELECT | Recupera datos de la base de datos |
| INSERT | Añade nuevas filas de datos a la base de datos |
| DELETE | Suprime filas de datos de la base de datos |
| UPDATE | Modifica datos existentes en la base de datos |
| *Definición de datos* | |
| CREATE TABLE | Añade una nueva tabla a la base de datos |
| DROP TABLE* | Suprime una tabla de la base de datos |
| ALTER TABLE* | Modifica la estructura de una tabla existente |
| CREATE VIEW* | Añade una nueva vista a la base de datos |
| DROP VIEW* | Suprime una lista de la base de datos |
| CREATE INDEX* | Construye un índice para una columna |
| DROP INDEX* | Suprime el índice para una columna |
| CREATE SYNONYM* | Define un alias para un nombre de tabla |
| DROP SYNONUM* | Suprime un alias para un nombre de tabla |
| COMMENT* | Define comentarios para una tabla |
| LABEL* | Define el título de una columna |
| *Control de acceso* | |
| GRANT | Concede privilegios de acceso a usuarios |
| REVOKE | Suprime privilegios de acceso a usuarios |
| *Control de transacciones* | |
| COMMIT | Finaliza la transacción actual |
| ROLLBACK | Aborta la transacción actual |
| *SQL programático* | |
| DECLARE | Define un cursor para una consulta |
| EXPLAIN* | Describe el plan de acceso a datos para una consulta |
| OPEN | Abre un cursor para recuperar resultados de consulta |
| FETCH | Recupera una fila de resultados de consulta |
| CLOSE | Cierra un cursor |
| PREPARE* | Prepara una sentencia SQL para ejecución dinámica |
| EXECUTE* | Ejecuta dinámicamente una sentencia SQL |
| DESCRIBE* | Describe una consulta preparada |

* No forma parte del estándar SQL1 ANSI/ISO, pero se encuentra en la mayoría de los productos más populares basados en SQL.

*Principales sentencias en SQL*

# 4. DDL (DATA DEFINITION LANGUAGE)

El SQL puede utilizarse como DDL o Lenguaje de definición de datos, esto es, podemos utilizar SQL para crear, modificar o borrar bases de datos, tablas, vistas o índices.

## CREACIÓN Y BORRADO DE UNA BASE DE DATOS

```
CREATE DATABASE  nbreBaseDatos;
```

Podemos averiguar cuántas bases de datos existen en nuestro sistema usando la sentencia :

```
SHOW DATABASES;
```

Y poner una base de datos en uso con la sentencia siguiente.

```
USE nbreBaseDAtos;
```

Para eliminarla usaremos la sentencia :

```
DROP DATABASE  nbreBaseDatos;
```

## CREACIÓN Y BORRADO DE UNA TABLA.

### BORRAR UNA TABLA.

Para eliminar una tabla existente de una base de datos

```
DROP TABLE tabla
```

Si queremos que la borre sólo si existe, para que  no muestre un mensaje de error de que no encuentra ese objeto, podemos utilizar:

```
DROP TABLE IF EXISTS emple;
```

## CREAR UNA TABLA

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

```
CREATE TABLE Nbr_tabla
      (atributo tipo [NOT NULL],    -- Definición de atributos
      , [DEFAULT valor]             -- Valor por defecto
       , [CKECK condición]          -- Integridad de dominio
                    {Atributo tipo }…
       PRIMARY KEY (atributos_clave),      -- Clave primaria
       FOREIGN KEY (atributos_claveforanea)
       REFERENCES tabla (atributos_otra_tabla))       -- Clave foránea
) ENGINE=tipo_motor DEFAULT CHARSET=juego_caracteres;
```

La definición de atributos se realiza dando el nombre del atributo, su tipo y opcionalmente indicando restricciones:

- NOT NULL :que no acepta valores nulos. Por defecto acepta valores nulos.
- DEFAULT value. Valor que tomará por defecto cuando no se le asigne
- CHECK condición que han de cumplir los valores de la columna. Pueden expresarse mediante IN(val1, .. valn) , operadores >, >=, <, <=, BETWEEN .. AND, OR.

La opción **`PRIMARY KEY`** define la clave primaria de la tabla, y **`FOREIGN KEY`** especifica la clave foránea, **indicando la tabla que referencia**. Si los atributos clave de la tabla referenciada no tienen el mismo nombre que los atributos que forman la clave foránea, deben especificarse los nombres de los atributos clave.

Por ejemplo: Creación de la tabla de empleado

**Script1**

```
CREATE TABLE  `compania_ir`.`emple` (
  `NSS` varchar(6) NOT NULL,
  `Nombre` varchar(15) default NULL,
  `Apel1` varchar(15) default 'López',
  `Apel2` varchar(15) default NULL,
  `Sexo` varchar(2) DEFAULT 'F'  CHECK (sexo in ('M', 'F')) ,
  `Dirección` varchar(30) default NULL,
  `Fechanac` datetime default NULL,
  `Salario` int(10) DEFAULT 100000 check (salario>30000),
  `Numdept` tinyint(3) unsigned default NULL,
  `NSSsup` varchar(6) default NULL,
  PRIMARY KEY  (`NSS`),
 FOREIGN KEY (`Numdept`) REFERENCES `departamento` (`Numdep`),
 FOREIGN KEY (`NSSsup`) REFERENCES `empleado` (`NSS`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Como se ve en este ejemplo, `Numdept` es una clave foránea que referencia a la columna Numdep de `DEPARTAMENTO`. NSSsup es una clave foránea que referencia a la columna NSS de la propia tabla Emple.

Si comparamos esta sentencia con la que genera MySQL Query Browser, podemos observar algunas diferencias.

**Script2, generado por MySQL Query Browser**

```
DROP TABLE IF EXISTS `compania_ir`.`empleado`;
CREATE TABLE  `compania_ir`.`empleado` (
  `NSS` varchar(6) NOT NULL,
  `Nombre` varchar(15) default NULL,
  `Apel1` varchar(15) default NULL,
  `Apel2` varchar(15) default NULL,
  `Sexo` varchar(2) default NULL,
  `Dirección` varchar(30) default NULL,
```

```
  `Fechanac` datetime default NULL,
  `Salario` int(10) default NULL,
  `Numdept` tinyint(3) unsigned default NULL,
  `NSSsup` varchar(6) default NULL,
  PRIMARY KEY  (`NSS`),
  KEY `FK_empleado_1` (`Numdept`),
  KEY `FK_empleado_2` (`NSSsup`),
  CONSTRAINT  `FK_empleado_1`  FOREIGN  KEY  (`Numdept`)  REFERENCES
`departamento` (`Numdep`),
  CONSTRAINT `FK_empleado_2` FOREIGN KEY (`NSSsup`) REFERENCES `empleado`
(`NSS`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Hay dos cláusulas KEY, que crean dos índices por las columnas indicadas entre paréntesis. Estos índices se generan automáticamente en MySQL Browser al crear las claves foráneas. En el siguiente epígrafe se indican las sentencias para crearlos y borrarlos independientemente.

Las claves foráneas van precedidas por CONSTRAINT nombre_restricción, para dar un nombre a la clave foránea.

Una cláusula **CONSTRAINT** establece varias limitaciones en un campo y se puede utilizar para establecer la clave principal, una relación con otra tabla (clave foránea) y para establecer índices

## CONSTRAINT (CLÁUSULA)

Se puede utilizar la cláusula CONSTRAINT en las instrucciones ALTER TABLE y CREATE TABLE para crear o eliminar limitaciones.

Sintaxis:

```
CONSTRAINT nombre
{PRIMARY KEY (claveprimaria) |
UNIQUE (unico1,unico2...) |
NOT NULL (campo1,campo2,.... )|
FOREIGN KEY (clave foránea) REFERENCES tablaexterna (campoexterno)}
```

La cláusula CONSTRAINT consta de las siguientes partes:

| **Parte** | **Descripción** |
|---|---|
| nombre | El nombre de la limitación que se va a crear. |
| Clave principal | El nombre del campo o de los campos que se van a designar como clave principal. |
| único1, único2 | El nombre del campo o de los campos que se van contener valores únicos para todas sus filas |
| Campo1, campo2 | El nombre del campo o de los campos que se van a limitar a valores |

| | |
|---|---|
| | no Null. |
| Clave foránea | El nombre del campo o de los campos de la clave externa a los que se hace referencia en otra tabla. |
| tablaexterna | El nombre de la tabla que se va hacer referencia cuando se defina la clave foránea. |
| campoexterno1, campoexterno2 | El nombre del campo o de los campos en tablaexterna que se va a referenciar . Puede omitir esta cláusula si el campo al que se hace referencia es la clave principal de tablaexterna. |

## EJEMPLO DE MODIFICACIÓN DE LA DEFINICIÓN DE UNA TABLA.

### ALTER TABLE Y CONSTRAINT.

El siguiente script crearía las claves foráneas de las tablas de la BD Compania, con actualización en cascada y borrado no permitido para las claves primarias que tengan claves foráneas.

```
-- Departamento - Empleado
ALTER TABLE departamento ADD CONSTRAINT FK_depar_1 FOREIGN KEY
(NSSgerente)
    REFERENCES empleado (NSS)
    ON DELETE RESTRICT
    ON UPDATE CASCADE;


-- Empleado - departamento
ALTER TABLE empleado ADD CONSTRAINT FK_empleado_1 FOREIGN KEY
FK_empleado_1 (NUMDEPT)
    REFERENCES departamento (Numdep)
    ON DELETE RESTRICT
    ON UPDATE CASCADE;

-- Empleado - Empleado

ALTER TABLE empleado ADD CONSTRAINT FK_supervisor FOREIGN KEY  (NSSsup)
    REFERENCES empleado (nss)
    ON DELETE RESTRICT
    ON UPDATE CASCADE;


-- EmpleadoProyecto - Proyecto
ALTER TABLE empleadoproyecto ADD CONSTRAINT FK_empleadoproyecto_2 FOREIGN
KEY FK_empleadoproyecto_2 (NUMPROY)
    REFERENCES proyecto (Numproy)
    ON DELETE RESTRICT
    ON UPDATE CASCADE;
```

```
-- EmpleadoProyecto - Empleado
ALTER TABLE empleadoproyecto ADD CONSTRAINT FK_con_empleado FOREIGN KEY
(NSS)
    REFERENCES  empleado (NSS)
    ON DELETE RESTRICT
    ON UPDATE CASCADE;


-- Proyecto - departamento
ALTER TABLE proyecto ADD CONSTRAINT FK_proyecto_1 FOREIGN KEY (Numdep)
    REFERENCES departamento (Numdep)
    ON DELETE RESTRICT
    ON UPDATE cascade;


-- si quiero eliminar alguna clave foránea, por ejemplo, esta última, --
haría lo siguiente, teniendo en cuenta que el nombre de es el que ---
sigue a la palabra CONSTRAINT

ALTER TABLE proyecto DROP FOREIGN KEY FK_proyecto_1;
```

## CREACIÓN Y BORRADO DE UN ÍNDICE

Además de crearse simultáneamente con la tabla a la que van asociados, también se puede utilizar la instrucción **CREATE INDEX** para crear índices adicionales.

```
CREATE [tipoindice] INDEX nombre_indice ON tabla( columna1, columna2, ….)
```

La siguiente sentencia crea un índice para la columna nombrebombon en la tabla Bombones.

```
CREATE INDEX NomBom on bombones(nombrebombon)
```

La siguiente sentencia crea un índice para varias columnas en la tabla Bombones.

```
CREATE INDEX Ingredien ON bombones(tipochocolate, tiponuez, tiporelleno);
```

Se puede indicar el nímero de caracteres de las columnas del indice (especialmente útil si son columnas de muchos caracteres, para no crear índices muy grandes)

La sentencia anterior podría escribirse entonces como sigue

```
CREATE INDEX Ingredien ON bombones(tipochocolate(4), tiponuez(3),
tiporelleno(4));
```

Los índices pueden ser, entre otros, de los siguientes tipos:

- PRIMARY: sólo puede existir uno por tabla, correspondiente a la clave primaria
- UNIQUE: para que no se admita valores repetidos en la columna o conjunto de columnas referidas. Las claves candidatas que no sean primarias deberán tener un índice de este tipo.

Así, para crear un índice único para la columna nombrebombon, emplearíamos la sentencia:

```
CREATE UNIQUE INDEX NomBom on bombones(nombrebombon)
```

Si en la tabla existen nombres de bombón repetidos, no nos permitirá crear el índice.

Para borrar un índice emplearé la siguiente sentencia:

ALTER TABLE *tabla* DROP INDEX *indice*

```
ALTER TABLE bombones DROP INDEX nombom;
```

# TIPOS DE DATOS EN SQL

Los tipos de datos determinan las características de los datos de las columnas.

Los tipos de datos que puede haber en un campo, se pueden agrupar en tres grandes grupos:

1. Tipos numéricos
2. Tipos de Fecha
3. Tipos de Cadena

**1 Tipos numéricos:**
Existen tipos de datos numéricos, que se pueden dividir en dos grandes grupos, los que están en coma flotante (con decimales) y los que no.

- **TinyInt:** es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255
- **Bit ó Bool:** un número entero que puede ser 0 ó 1
- **SmallInt:** número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.
- **MediumInt:** número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.
- **Integer, Int:** número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295
- **BigInt:** número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

CON DECIMALES

- **Float:** número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.

- **xReal, Double:** número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308

Resumidos en la siguiente tabla:

| Tipo de Campo | Tamaño de Almacenamiento |
|---|---|
| TINYINT | 1 byte |
| SMALLINT | 2 bytes |
| MEDIUMINT | 3 bytes |
| INT | 4 bytes |
| INTEGER | 4 bytes |
| BIGINT | 8 bytes |
| FLOAT(X) | 4 ú 8 bytes |
| FLOAT | 4 bytes |
| DOUBLE | 8 bytes |
| DOUBLE PRECISION | 8 bytes |
| REAL | 8 bytes |

**2 - Tipos fecha:**

A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes esta comprendido entre 0 y 12 y que el día esta comprendido entre 0 y 31.

- **Date:** tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia

- **DateTime:** Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

- **TimeStamp:** Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:

| Tamaño | Formato |
|---|---|
| 14 | AñoMesDiaHoraMinutoSegundo aaaammddhhmmss |
| 12 | AñoMesDiaHoraMinutoSegundo aammddhhmmss |
| 8 | AñoMesDia aaaammdd |
| 6 | AñoMesDia aammdd |
| 4 | AñoMes aamm |
| 2 | Año aa |

- **Time:** almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

- **Year:** almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

| Tipo de Campo | Tamaño de Almacenamiento |
|---|---|
| DATE | 3 bytes |
| DATETIME | 8 bytes |
| TIMESTAMP | 4 bytes |

| | |
|---|---|
| TIME | 3 bytes |
| YEAR | 1 byte |

**3 - Tipos de cadena:**

**Char(n):** almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

**VarChar(n):** almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.

Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Test y los tipo BLOB (Binary large Object)

La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Mientras que el tipo test se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndolas en cuenta.

Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.

**TinyText y TinyBlob:** Columna con una longitud máxima de 255 caracteres.

**Blob y Text:** un texto con un máximo de 65535 caracteres.

**MediumBlob y MediumText:** un texto con un máximo de 16.777.215 caracteres.

**LongBlob y LongText:** un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.

**Enum:** campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos

**Set:** un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores.

| Tipo de campo | Tamaño de Almacenamiento |
|---|---|
| CHAR(n) | n bytes |
| VARCHAR(n) | n +1 bytes |
| TINYBLOB, TINYTEXT | Longitud+1 bytes |
| BLOB, TEXT | Longitud +2 bytes |
| MEDIUMBLOB, MEDIUMTEXT | Longitud +3 bytes |
| LONGBLOB, LONGTEXT | Longitud +4 bytes |
| ENUM('value1','value2',...) | 1 ó dos bytes dependiendo del número de valores |
| SET('value1','value2',...) | 1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores |

Diferencia de almacenamiento entre los tipos Char y VarChar

| Valor | CHAR(4) | Almace namiento | VARCHAR(4) | Almace namiento |
|---|---|---|---|---|
| '' | '' | 4 bytes | '' | 1 byte |
| 'ab' | 'ab ' | 4 bytes | 'ab' | 3 bytes |
| 'abcd' | 'abcd' | 4 bytes | 'abcd' | |
| 'abcdefgh' | 'abcd' | 4 bytes | 'abcd' | 5 bytes |