Iniciación a SQL

Manua l de iniciación a SQL Server 2019

Miguel Marañón Quero

2023

Contenido

[Teoría 2](#_Toc131010808)

[Estructura Física 2](#_Toc131010809)

[Nombres de archivos 2](#_Toc131010810)

[Crecimiento de la base de datos 2](#_Toc131010811)

[Estructura lógica 2](#_Toc131010812)

[Bases de datos del sistema 3](#_Toc131010813)

[Tipos de datos 3](#_Toc131010814)

[Valores identidad 4](#_Toc131010815)

[Consultas 5](#_Toc131010816)

[Subconsultas 6](#_Toc131010817)

[Conversión de Parámetros 7](#_Toc131010818)

[Agrupaciones 9](#_Toc131010819)

[Tablas temporales 11](#_Toc131010820)

[Uniones y combinaciones 11](#_Toc131010821)

[Inserciones y modificaciones 13](#_Toc131010822)

[Transacciones 14](#_Toc131010823)

[TABLAS FINALES 15](#_Toc131010824)

# Teoría

Una base de datos es un sistema donde se almacena información, no solo asegura el almacenamiento y la recuperación de datos, también asegura su integridad. Para usar SQL es muy importante optimizar el rendimiento, a la hora de ejecutar consultas siempre se debe emplear la menor información y el menor número de procesos posibles para evitar la saturación del sistema.

La lógica del funcionamiento de una base de datos pertenece también a la misma, definiendo como procedimientos almacenados las operaciones más habituales.

## Estructura Física

La estructura física de una base de datos consta de dos elementos principales:

* **Archivo principal de datos**: contiene los datos, índices, vistas, procedimientos almacenados, etc.
* **Archivo de registro de transacciones**: se usa para asegurar la integridad de la base de datos, facilitando la recuperación de la información si fuera necesario.

Además de estos, puede haber más tipos de archivos y archivos secundarios, pero lo mínimo son los dos ya mencionados.

## Nombres de archivos

Como norma general, los nombres de los archivos de una base de datos están definidos por el nombre de esta, por ejemplo: creamos la base de datos Juego.

* **Archivo principal de datos:** Juego.MDF
* **Archivo de registro**: Juego\_log.LDF

Los archivos secundarios pueden tomar el nombre que quieran y tendrán una extensión .NDF

## Crecimiento de la base de datos

Como no es fácil prever cual va a ser el tamaño de una base de datos, normalmente no se les asigna un espacio fijo, el propio gestor de bases de datos aumenta el tamaño establecido cuando llega a su límite.

## Estructura lógica

Las bases de datos funcionan a través de la estructura lógica de cara al usuario, de tal forma que sea fácil de usar. Hay en total 4 elementos para tener en cuenta con otros añadidos a parte:

* **Tablas:** Es la forma más básica de almacenamiento de información. Está compuesta por registros (filas) y campos (columnas).
* **Índice**: Cuando una tabla es demasiado grande, su funcionamiento es menos óptimo a la hora de recuperar información, los índices aceleran este proceso además de facilitar el ordenamiento de datos al registrarlos. Cuando se manipula un registro que también pertenezca a un índice, también se manipula el contenido del propio índice.
* **Vistas:** Son consultas ya almacenadas en la propia base de datos para ciertos procedimientos que son muy repetitivos.
* **Procedimientos almacenados**: Se definen con Transact SQL, implementan dentro de la base de datos procedimientos para la manipulación de esta, son mucho más rápidos que las sentencias enviadas desde la aplicación porque no tiene que ejecutar código y está ya compilado.
* **Otros elementos lógicos adicionales**: diagramas, reglas funciones definidas, etc. Son algunos elementos lógicos adicionales que genera el propio usuario de forma manual, no son tan vitales en el funcionamiento de la base de datos, pero aportan mucha utilidad. Desde un punto de vista de seguridad, las funciones son muy útiles porque pueden asignarse o denegarse a los usuarios.

## Bases de datos del sistema

En SQL Server existen dos tipos de bases de datos, las que vienen implementadas con el sistema (master, model, tempdb, msdby resource) y las que crea el propio usuario.

* **Master**: registra toda la información del sistema SQL server, todas las cuentas de inicio de sesión, la existencia del resto de bases de datos y la información de la inicialización del servicio SQL Server en el sistema.
* **Tempdb**: almacena todas las tablas y procedimientos almacenados temporales, como todo lo temporal se elimina al finalizar el servicio, no necesita guardar información en el sistema. Su tamaño crece según sea necesario.
* **Model**: se utiliza como plantilla para todas las bases de datos del sistema, cuando se crea una nueva, se hace copiando el contenido de esta base de datos. Es modificable para poder incluir ciertas características adicionales de forma automática a las bases que hagamos de forma próxima.
* **Msdb**: La base de datos, msdb utiliza el Agente SQL Server para programar alertas y trabajos y para registrar operadores.
* **Resource**: Base de datos de solo lectura que contiene objetos del sistema que se incluyen con SQL Server. Los objetos del sistema persisten físicamente en la base de datos **resource**, pero aparecen lógicamente en el esquema **sys** de cada base de datos.

## Tipos de datos

A la hora de almacenar datos, importa mucho en que formato almacenamos las cosas, hay una organización en cuanto a cómo catalogar cada dato:

* **Numéricos**: familia de los “int”, real, money, float y bit, representan valores numéricos en distintos rangos. Lo mas importante es utilizar **tinyint** para números entre 0 y 255 y **smallint** para números entre -32k y +32k, **int** y **bigint** llegan a rangos de millones y trillones, ocupan mas espacio y son menos útiles.

**Money** puede tomar valores positivos y negativos y aporta 4 decimales, **real** toma valores decimales negativos y positivos, **float** funciona muy parecido a **real** y bit toma valores de 0 o 1.

* **Caracteres**: puede usarse **char** (cadena de tamaño fijo entre 0 y 8000, rellena los huecos con espacios), **varchar** (cadena de tamaño variable, no rellena huecos) **text** (secuencias de caracteres, se utilizan para cadenas de más de 8k caracteres).

Si se le pone una n delante a cada tipo (**nchar**, **nvarchar** y **ntext**) se pueden usar caracteres especiales de otros alfabetos.

* **Fechas**: almacenan fechas y horas, pueden ser **smalldatetime**, **datetime**, **date.**

## Valores identidad

Los valores identidad se definen activando una propiedad en una columna, lo que establece que SQL Server inserte los valores automáticamente, deben ser de tipo numérico.

Solo puede establecerse una identidad por tabla , aunque parecen bastante útiles, pueden acarrear una serie de problemas: el valor asignado solo se puede usar una vez aunque se elimine el registro, por lo que se van quedando huecos sin utilizar a la larga cuando se han eliminado y creado registros nuevos.

* **Índices:** hay dos tipos diferentes de índices: **agrupados** (determina el orden físico de las tablas, solo puede existir un índice de este tipo por tabla) y **no agrupados** (es un objeto separado que apunta a ciertas filas específicas, no determina como se almacenan esas filas)

además de estos índices, también se genera un índice automático cuando se establece una clave primaria en una tabla.

* **Relaciones**: SQL Server representa las relaciones entre las entidades (representadas como tablas) como restricciones de clave extranjera. Sirven para asegurar la integridad referencial.
* **Control de restricciones:** se utilizan para garantizar dos tipos de integridad.
  + **Integridad de dominio:** define el rango de valores que puede tomar un campo, se implementan con restricciones check.
  + **Integridad de identidad:** fuerza la integridad de una entidad, evita que en una sola tabla pueda haber varias entidades con el mismo valor.

# Consultas

Para hacer consultas se necesitan dos funciones básicas: SELECT y FROM. La función SELECT se encarga de recoger los datos de las columnas indicadas, la función FROM le indicará a SELECT de qué tabla tiene que recoger las columnas.

Luego existe una función añadida para filtrar los datos llamada WHERE, sirve para añadir una condición, normalmente sigue una de las siguientes estructuras:

* WHERE edad = 19
* WHERE nombre = ‘Samira’
* WHERE NIF LIKE ‘S-%’
* WHERE Telefono IS NULL

También se puede utilizar BETWEEN - AND como comparador de dos atributos o dos parámetros, como por ejemplo fechas o números:

* WHERE edad BETWEEN 19 and 32
* WHERE fecha BETWEEN ‘01/01/2022’ AND ‘31/12/2022’

Sea la tabla Personaje, queremos los nombres con NIF que empiece por “S-“

SELECT Nombre FROM Personaje WHERE NIF LIKE ‘S-%’

|  |  |  |
| --- | --- | --- |
| **Nombre** | **NIF** | **Telefono** |
| Samson | S-1111 | 611 111 111 |
| Samira | S-2222 | 622 222 222 |
| Hornet | N-3333 | *NULL* |

A la hora de hacer consultas también podemos utilizar varias condiciones al mismo tiempo (con AND o OR) y elegir todos los parámetros a la vez (poniendo \*)

Cuando usamos AND estamos exigiendo que ambas condiciones se cumplan, cuando usamos OR estamos exigiendo que se cumplan una de las dos o ambas, sin ser necesario que sea simultáneo.

También podemos pedir valores nulos igualando un parámetro a NULL.

SELECT \* FROM Personaje WHERE Telefono IS NULL

SELECT \* FROM Personaje WHERE NIF LIKE ‘S-2%’ OR Nombre LIKE ‘S%’

SELECT \* FROM Personaje WHERE NIF LIKE ‘S-1%’ AND Nombre LIKE ‘S%’

La primera solicitud nos devolverá solo la fila de **Samson**, mientras que la segunda nos devolverá a **Samson** y **Samira**. La última nos devolverá a **Hornet** porque es la única con un teléfono nulo.

## Subconsultas

Una vez tenemos claro el funcionamiento de las consultas, podemos pasar a las **Subconsultas.**

Una subconsulta ocurre cuando realizamos una consulta y de ella queremos sacar ciertos parámetros, utilizan como nexo entre la consulta principal y la subconsulta la función IN (SELECT …), También se puede negar con un NOT delante para solicitar que se consulten aquellos que no se han reflejado en la subconsulta. Imagina que tenemos dos tablas. Una para **Personajes** y otra para **Regiones**. Adicionalmente creamos una tabla llamada **Origen**.

SELECT Nombre FROM Personajes WHERE NIF IN (SELECT RegionID FROM Origen WHERE RegionID = ‘B-1’)

|  |  |  |
| --- | --- | --- |
| **Personajes** | | |
| **Nombre** | **NIF** | **Telefono** |
| Samson | S-1111 | 611 111 111 |
| Samira | S-2222 | 622 222 222 |
| Hornet | N-3333 | *NULL* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Regiones** | |  | **Origen** | |
| **Region** | **ID** |  | **PersonajeNIF** | **RegionID** |
| Basement | B-1 |  | S-1111 | B-1 |
| Runaterra | R-1 |  | S-2222 | R-1 |
| Hallownest | H-3 |  | N-3333 | H-3 |

De esta forma estamos solicitando a una consulta principal que saque el nombre de todos los personajes con un **NIF** que esté incluido en la subconsulta.

La subconsulta estará escogiendo todos los **NIF** que correspondan a la región con ID  
**B-1**, que en la tabla de Regiones tiene el nombre **Basement**. Por lo que si analizamos el código el resultado será un nombre: **Samson**.

Esto se debe a que de la tabla **Origen** se obtiene un solo **RegionID** correspondiente a   
**B-1,** el cual se va a corresponder con **PersonajeNIF** con valor S-1111 ese **PersonajeNIF** se compara con la tabla **Personajes** y se analiza a que **Nombre** corresponde, en este caso al de **Samson**.

Si quisiéramos comparar entre las tablas **Personajes** y **Regiones**, tenemos que pasar por la tabla **Origen**, porque no hay ningún campo común a ambas tablas, para esto existen los **JOIN** que se explicarán más adelante.

## Conversión de Parámetros

A la hora de trabajar con algunos parámetros, no podemos sacarlos así como así, puede haber conflicto con el sistema gestor de bases de datos y no podríamos realizar la consulta que estaríamos deseando. Éstas son las situaciones que se nos pueden presentar

* Diferencias en fechas (DATEDIFF): La función DATEDIFF necesita tres parámetros, el **primero** es la cantidad (días, meses o años) con el que se quiere comparar, el **segundo** la fecha de inicio (por ejemplo, la fecha que se almacena en una tabla) y la tercera es la **fecha** final (por ejemplo, la fecha que se almacena en otra tabla). Para obtener la fecha actual podemos usar el parámetro GETDATE().
* Números con frases delante (CONVERT): La función CONVERT necesita tres parámetros, el **primero** es el tipo de dato al que se quiere convertir (generalmente es CHAR), el **segundo** es el atributo del que se quiere sacar el dato, el **tercero** es opcional, cuando se dé la situación de que se quiera convertir una fecha para ponerla en otro formato también se puede usar esta función, aquí iría un número que representa un estándar, por ejemplo 103.
* Operaciones (Parámetro +/\*- número): Se pondría de la siguiente forma, **Edad** \* 10 (queremos la edad multiplicada por 10)
* Sacar fragmentos de frases (SUBSTRING): La función SUBSTRING necesita tres parámetros, el **primero** es el atributo del que se quiere sacar el dato, el **segundo** es el carácter del que se quiere partir, el **tercero** es la cantidad de caracteres que se quieren coger.
* Sacar longitud de frases (LEN): Se introduce de la siguiente forma, LEN(Nombre), sacaría el número de caracteres en el parámetro **nombre.**
* Convertir las letras en mayúsculas (UPPER): Se introduce de la siguiente forma, UPPER(Nombre), pondría el **nombre** en mayúscula.
* Convertir las letras en minúsculas (LOWER): Se introduce de la siguiente forma, LOWER(Nombre), pondría el **nombre** en minúscula.
* Eliminar espacios en blancos (RTRIM): se introduce de la siguiente forma, RTRIM(teléfono), pondría el **telefono** sin espacios.
* Redondear (ROUND): redondea el atributo que se le introduce, se escribe de la siguiente forma, ROUND(atributo, x) donde x es el número de decimales.

Estos son algunos ejemplos de cómo serían, imaginamos que a la tabla Personaje añadimos la edad y la fecha en la que se añadió al juego:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre** | **NIF** | **teléfono** | **Edad** | **Fecha** |
| Samson | S-1111 | 611 111 111 | 5 | 07/12/2013 |
| Samira | S-2222 | 622 222 222 | 22 | 01/09/2020 |
| Hornet | N-3333 | *NULL* | *NULL* | 10/08/2017 |

SELECT DATEDIFF(MONTH, Fecha, GETDATE()) FROM Personajes WHERE Nombre = ‘Hornet’

Sacar cuantos meses han pasado desde que **Hornet** llegó a Hallownest.

SELECT ‘El teléfono es’ + CONVERT(CHAR, Teléfono) FROM Personajes WHERE Nombre = ‘Samira’

Sacar el teléfono de **Samira** con la frase ‘el teléfono es’ delante sabiendo que el atributo es de tipo INT, por lo que lo convertimos a CHAR.

SELECT Edad \* 10 FROM Personajes WHERE Nombre = ‘Samson’

Sacar la edad de **Samson** multiplicada por 10.

Sacar los tres primeros caracteres de los nombres de todos los personajes que hay.

SELECT SUBSTRING(Nombre, 1, 3) FROM Personajes

SELECT LEN(Nombre) FROM Personajes WHERE Nombre = ‘Samira’

Sacar la longitud del nombre de **Samira**

SELECT UPPER(Nombre) FROM Personajes WHERE Nombre = ‘Hornet’

Sacar el nombre de **Hornet** en mayúsculas

SELECT LOWER(Nombre) FROM Personajes WHERE NIF = ‘S-1111’

Sacar el nombre de **Samson** en minúsculas

Cuando hagamos todas estas selecciones, a veces no aparecerá nombre en la columna porque se ha manipulado su información y se percibe como si fuera nueva, por lo que podemos poner un AS justo después de la selección para darle nombre a la columna:

SELECT SUBSTRING(UPPER(Nombre), 1, 3) AS ‘primeros 3 caracteres del nombre en mayúsculas’ FROM Personajes

## Agrupaciones

A la hora de manejar variables también nos puede interesar contabilizar el contenido que tengan o incluso ordenarlas en el momento en el que se imprimen tras consultarlas. Para esto utilizamos una serie de comandos entre los que destacan COUNT y DISTINCT. También se pueden incluir algunas funciones como SUM(), ORDER BY y GROUP BY.

Aquí vamos a añadir dos tablas: **Jugador** y **Amistad**. También añadiremos tres nuevos personajes a la tabla de **Personajes.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **AR** |  | **ID\_Jugador** | **NIF\_personaje** | **NivAmistad** |
| Guilirecs | 58 |  | EskolUndead | S-1111 | 8 |
| LilDalia | 56 |  | EskolUndead | S-2222 | 10 |
| EskolUndead | 55 |  | LilDalia | H-2356 | 10 |
|  |  |  | LilDalia | B-1111 | 7 |
|  |  |  | Guilirecs | N-3333 | 10 |
|  |  |  | Guilirecs | Q-3333 | 9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre** | **NIF** | **Telefono** | **Edad** | **Fecha** |
| Samson | S-1111 | 611 111 111 | 5 | 07/12/2013 |
| Samira | S-2222 | 622 222 222 | 22 | 01/09/2020 |
| Raiden | H-2356 | *NULL* | 21 | 10/08/2021 |
| Brand | B-1111 | 621 111 111 | 25 | 9/11/2015 |
| Hornet | N-3333 | 623 222 222 | *NULL* | 10/08/2017 |
| Childe | C-3333 | *NULL* | 19 | 28/09/2020 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Region** | **ID** |  | **PersonajeNIF** | **RegionID** |
| Basement | B-1 |  | S-1111 | B-1 |
| Runaterra | R-1 |  | S-2222 | R-1 |
| Hallownest | N-3 |  | H-2356 | T-1 |
| Teyvat | T-1 |  | B-1111 | R-1 |
|  |  |  | N-3333 | N-3 |

* COUNT: Se encarga de contar la cantidad de datos que hay en una tabla, distinguiendo los parámetros repetidos o no. Por ejemplo, si en la tabla **Origen** tenemos 6 registros de los cuales 2 tienen el valor **R-1,** otros 2 **T-1** y demás. En la columna **Origen**, podemos usar COUNT(**Origen**) y la consulta contaría 6 registros, pero si usamos COUNT(DISTINCT **Origen**) la tabla contaría 4 porque detectaría que **B-1**, **R-1**, **T-1** y **N-3**, estando repetidas **R-1** y **T-1**.
* DISTINCT: Tiene una función muy parecida cuando se usa de forma aislada. Cuando la usamos con COUNT distingue los registros repetidos para sacar una suma de los que son distintos, mientras que cuando se usa sola lo que hace es imprimir los registros sin repetirlos. Por ejemplo, si se escribe SELECT DISTINCT **RegionID** FROM **Origen**, nos devolverá una tabla con los valores **B-1**, **R-1**, **T-1** y **N-3**.
* SUM(): Se utiliza para contar el valor total de una columna que guarda valores de varios registros con la misma identificación. Imagina que queremos sumar toda la **amistad** que tiene **Guilirecs** con sus **personajes**, tendríamos que usar SELECT SUM(**NivAmistad**) FROM **Amistad** WHERE **ID\_Jugador** = ‘**Guilirecs**’.
* ORDER BY: Se utiliza para ordenar los parámetros cuando los muestra una consulta, se utiliza los parámetros ASC o DESC para solicitar que sea un orden ascendente o descendente. por ejemplo, ORDER BY **edad** ASC.
* GROUP BY: Esta función coge todos los resultados y los imprime en la tabla agrupándolos en función de lo que se le ordene tomar como referencia. Para poner condiciones hay que agregar el parámetro HAVING, que tiene una función muy parecida a WHERE a la hora de trabajar, pero se usan en contextos distintos.

Estos son algunos ejemplos de cómo funcionarían las agrupaciones ya mencionadas:

Cuenta la cantidad de **Personajes** que hay en total en la tabla **Personajes**.

SELECT COUNT(\*) as Num\_personajes FROM Personajes

SELECT DISTINCT Region FROM Regiones

Muestra todas las **Regiones** sin ser repetidas, solo puede aparecer cada región una vez.

SELECT SUM(NivAmistad) FROM amistad WHERE ID\_Jugador = ‘EskolUndead’

Suma la amistad total que tiene **EskolUndead** con sus personajes.

SELECT \* FROM Amistad ORDER BY NivAmistad DESC

Selecciona todos los registros de la tabla **amistad** ordenándolos por nivel de amistad descendente.

SELECT \* FROM Regiones ORDER BY Region DESC

Selecciona todos los registros de la tabla **Regiones** ordenándolos por **Region** descendente en orden alfabético.

## Tablas temporales

Para usar tablas temporales, usaremos SELECT seguido de las columnas que queramos incluir, luego escribiremos INTO seguido del nombre de la tabla que queramos crear (para que sea temporal debe tener # delante del nombre), por último, escribimos FROM para seleccionar la tabla de donde se sacan los datos.

SELECT \* FROM #personajesPermanentes

SELECT \* FROM Personajes INTO #personajesPermanentes WHERE NIF = ’Q-3333’

## Uniones y combinaciones

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **AR** |  | **ID\_Jugador** | **NIF\_personaje** | **NivAmistad** |
| Guilirecs | 58 |  | EskolUndead | S-1111 | 8 |
| LilDalia | 56 |  | EskolUndead | S-2222 | 10 |
| EskolUndead | 55 |  | LilDalia | H-2356 | 10 |
|  |  |  | LilDalia | B-1111 | 7 |
|  |  |  | Guilirecs | N-3333 | 10 |
|  |  |  | Guilirecs | Q-3333 | 9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre** | **NIF** | **Telefono** | **Edad** | **Fecha** |
| Samson | S-1111 | 611 111 111 | 5 | 07/12/2013 |
| Samira | S-2222 | 622 222 222 | 22 | 01/09/2020 |
| Raiden | H-2356 | *NULL* | 21 | 10/08/2021 |
| Brand | B-1111 | 621 111 111 | 25 | 9/11/2015 |
| Hornet | N-3333 | 623 222 222 | *NULL* | 10/08/2017 |
| Childe | C-3333 | *NULL* | 19 | 28/09/2020 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Region** | **ID** |  | **PersonajeNIF** | **RegionID** |
| Basement | B-1 |  | S-1111 | B-1 |
| Runaterra | R-1 |  | S-2222 | R-1 |
| Hallownest | N-3 |  | H-2356 | T-1 |
| Teyvat | T-1 |  | B-1111 | R-1 |
|  |  |  | N-3333 | N-3 |

Cuando trabajamos con muchas tablas, a veces se nos va a pedir que busquemos datos que hay en una tabla tomando como referencia un dato de otra tabla. Por ejemplo: Dime los nombres y la región de todos los personajes pertenecientes a **Liyue**. Para esto necesitaremos hacer un INNER JOIN o JOIN a secas (o se puede hacer sin él), lo que tendría dos tipos de sintaxis tal que así:

SELECT Nombre, RegionID FROM Personaje, Origen WHERE Personaje.NIF = Region.NIF AND Region = ‘Runaterra’

SELECT Nombre, RegionID FROM Personaje INNER JOIN Origen ON Personaje.NIF = Region.NIF WHERE Region = ‘Hallownest’

También existen otro tipo de selecciones, como los OUTER JOIN y CROSS JOIN.

* OUTER JOIN: Esta función funciona como el INNER JOIN a diferencia de que rellena las filas que no tengan asignadas ciertos registros, éstos tomarán el valor de *NULL*. Va acompañada de RIGHT o LEFT en función de lo que se quiera rellenar.
* CROSS JOIN: Esta función combina de todas las formas posibles las tablas que se le asignen. No suelen ser muy eficaz más allá de eso.

Por ejemplo, para consultar todos los nombres de sus personajes junto con sus respectivas regiones:

SELECT Nombre, Region FROM Regiones LEFT OUTER JOIN Personaje ON Personaje.NIF = Region.NIF

SELECT Nombre, Region FROM Personaje RIGHT OUTER JOIN Regiones ON Personaje.NIF = Region.NIF

Seleccionar todas las combinaciones entre Jugadores y Personajes.

SELECT Personaje.Nombre, Jugador.ID\_Jugador FROM Personaje CROSS JOIN Jugador

# Inserciones y modificaciones

Las inserciones se utilizan para no tener que agregar registros en las tablas de forma manual, son muy útiles especialmente cuando se trata de **inserciones masivas.** Su función principal es INSERT (INTO de forma opcional) el nombre de la tabla, seguido de las columnas que se quieran llenar entre paréntesis, seguido de VALUES y los valores que se quieran añadir.

INSERT INTO Personaje (Nombre, NIF, edad, fecha) VALUES (‘Lilith’, ‘L-4444’, 5, ‘12/02/2015’)

SELECT \* FROM Personaje

Para realizar inserciones masivas, necesitaremos utilizar otra sintaxis, que se parece bastante a la de un SELECT. Supongamos q ue creamos una tabla vacía nueva llamada **Equipo**. Definimos **ID\_Eq** como índice.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID\_Eq** | **ID\_Jugador** | **NIF\_1** | **NIF\_2** | **NIF\_3** | **NIF\_4** |

De esta forma estaremos añadiendo un equipo para cada jugador, más tarde se podrían crear más equipos para cada jugador sin problemas. También podremos añadir personajes a cada equipo con las **modificaciones**.

INSERT INTO Equipo SELECT (ID\_Jugador) FROM Jugador

Además de poder insertar en tablas, podemos eliminar filas enteras o incluso todo el contenido de la tabla, esto se hace con la función DELETE y se utiliza de la siguiente forma:

De esta forma estaremos eliminando el equipo de **LilDalia** porque es un pesado de mierda y quería aparecer en un ejemplo.

DELETE Equipo WHERE ID\_Jugador = ‘LilDalia’

También tenemos las modificaciones, que sirven para añadir o editar contenido en filas que ya han sido creadas en tablas. Supongamos que queremos añadirle a **LilDalia** un equipo porque resulta que no he borrado su equipo aún, sabiendo la ID\_Eq (vamos a suponer que es 2), Sería de la siguiente forma:

UPDATE Equipo SET (NIF\_1 = S-2222, NIF\_2 = S-1111, NIF\_3 = N-3333, NIF\_4 = L-4444) WHERE ID\_Eq = 2

Aprovechando que Childe no tiene ninguna región aún, vamos a añadirle a la lista:

INSERT INTO Origen VALUES (‘Teyvat’, ‘C-3333’)

Por último, existe otra forma de eliminar tablas con TRUNCATE TABLE, que sirve para borrar la tabla entera de una.

## Transacciones

Las transacciones son acciones que abren un proceso en el cual no se harán cambios sobre la base de datos real mientras esté activo, se pueden ejecutar todo tipo de funciones dentro de una transacción, de la misma forma que si no estas en ella, pero si en cualquier momento cometes un error o algo que comprometa toda la base de datos, puedes cancelar la transacción para que no se aplique ningún cambio.

Para comenzar una transacción se escribe el comando BEGIN TRAN, para terminarla sin que se guarde nada se escribe ROLLBACK. Para confirmar el proceso de una transacción se escribe COMMIT TRAN.

# TABLAS FINALES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **AR** |  | **ID\_Jugador** | **NIF\_personaje** | **NivAmistad** |
| Guilirecs | 58 |  | EskolUndead | S-1111 | 8 |
| LilDalia | 56 |  | EskolUndead | S-2222 | 10 |
| EskolUndead | 55 |  | LilDalia | H-2356 | 10 |
|  |  |  | LilDalia | B-1111 | 7 |
|  |  |  | Guilirecs | N-3333 | 10 |
|  |  |  | Guilirecs | Q-3333 | 9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre** | **NIF** | **Telefono** | **Edad** | **Fecha** |
| Samson | S-1111 | 611 111 111 | 5 | 07/12/2013 |
| Samira | S-2222 | 622 222 222 | 22 | 01/09/2020 |
| Raiden | H-2356 | *NULL* | 21 | 10/08/2021 |
| Brand | B-1111 | 621 111 111 | 25 | 9/11/2015 |
| Hornet | N-3333 | 623 222 222 | *NULL* | 10/08/2017 |
| Childe | C-3333 | *NULL* | 19 | 28/09/2020 |
| Lilith | L-4444 | *NULL* | 5 | 12/02/2015 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Region** | **ID** |  | **PersonajeNIF** | **RegionID** |
| Basement | B-1 |  | S-1111 | B-1 |
| Runaterra | R-1 |  | S-2222 | R-1 |
| Hallownest | N-3 |  | H-2356 | T-1 |
| Teyvat | T-1 |  | B-1111 | R-1 |
|  |  |  | N-3333 | N-3 |
|  |  |  | C-3333 | T-1 |
|  |  |  | L-4444 | B-1 |