**Clase 08 - Tablas SQL**

**Propiedades**

Entendemos por **propiedades** a las columnas, tipos de datos, claves e índices de una tabla.

Y **cuando utilizamos un SGBD** como **Mysql Workbench**, podemos observar toda esta información mucho más cómodamente.

**Árbol de navegación**

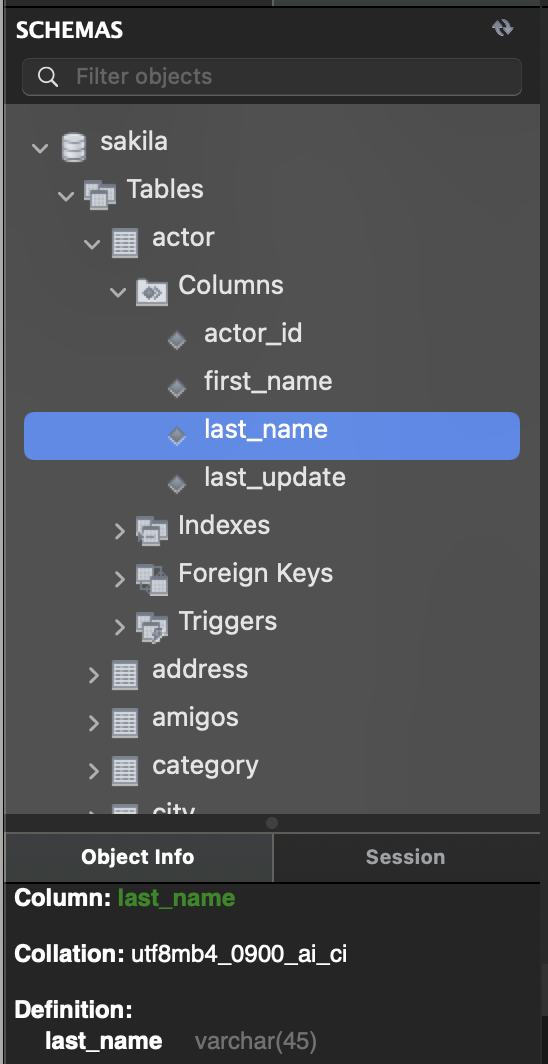
En el panel Schemas de Mysql Workbench encontrarán un árbol de características desplegable que permite acceder a las **propiedades** de una tabla, donde encontrarán:

* Columns
* Indexes
* Foreign Keys
* Triggers

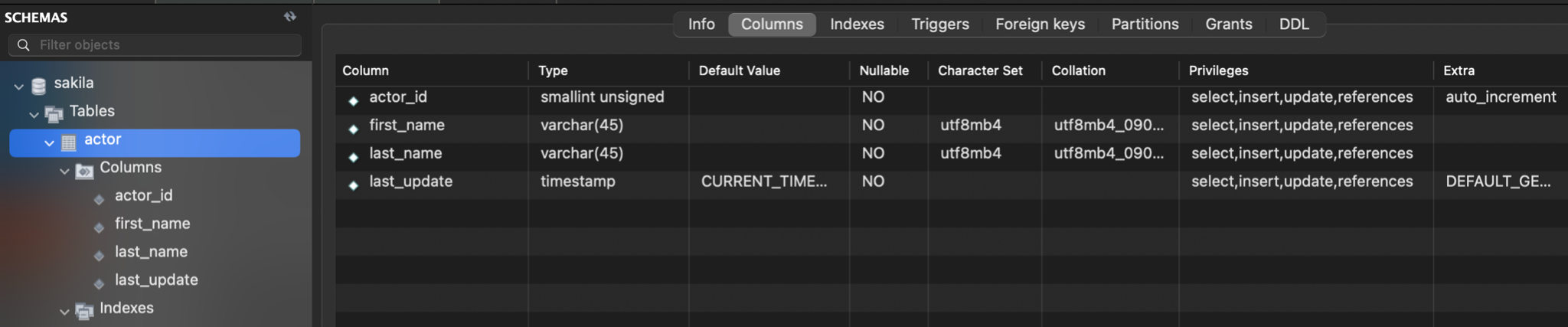
**Columns**

En el nodo **Columns** podrán apreciar, al desplegarlo, todas las columnas de la tabla seleccionada.

Luego, pulsando sobre cualquiera de ellas verán en el panel inferior, **Object Info**, la información de las propiedades que se le asignaron.



**Table inspector**

****

También, a través del menú contextual del mouse, pulsando sobre la tabla en cuestión, accedemos a **Table Inspector** donde encontramos el apartado **Columns** con posibilidad de ver el detalle completo de sus valores.

**TABLAS TRANSACCIONALES**

**Concepto**

**Mysql soporta diversos motores de almacenamiento** y estos, a su vez, soportan diferentes tipos de tablas.

**Algunos trabajan con** **tablas transaccionales**, mientras que la mayoría de los otros, no.

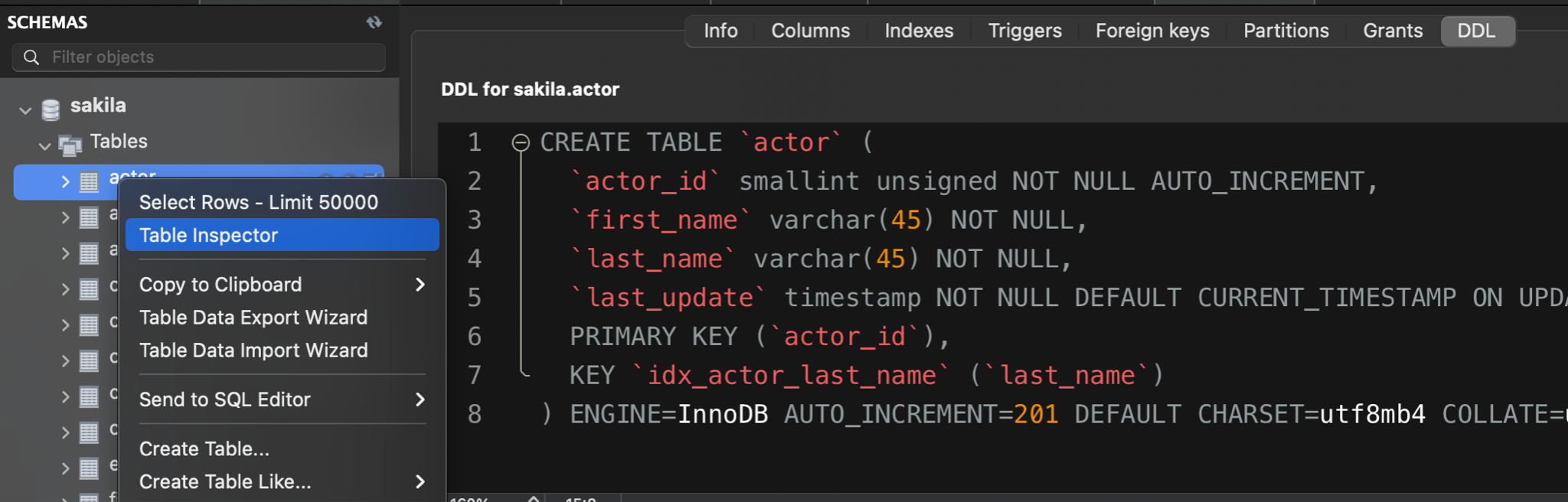
Entre estos motores, podemos encontrar:

* MyISAM
* MEMORY
* InnoDB y BDB
* EXAMPLE
* NDB Cluster
* ARCHIVE
* CSV
* FEDERATED

De todos ellos, **InnoDB** y **BDB** son motores transaccionales siendo el primero el más elegido en Mysql, gracias a que InnoDB soporta bloqueos y capacidades de **COMMIT** y **ROLLBACK** además de **Recuperación ante Fallas**. InnoDB también permite **bloqueo a nivel de filas**, mejorando su rendimiento y **soportando múltiples usuarios en simultáneo**.

También fue diseñado para procesar de manera ágil **grandes volúmenes de datos**.

Puedes encontrar referencia a este, visualizando cualquier tabla de nuestra base de datos SAKILA, a través de la opción **Table Inspector**. Allí verás la opción por defecto **ENGINE=InnoDB**.



**TABLAS DE HECHO**

**Concepto**

Este tipo de tablas se ocupan de capturar información de un único hecho de negocios.

Su principal objetivo es generar un evento de medición específico de la información clave que se debe analizar.

Son utilizadas para el modelo analítico de un sistema de información de base de datos.

El segmento de **Analítica de Datos** basado en **Business Intelligence**, se apoya en tablas de hecho y dimensionales para obtener las métricas de reportes que los usuarios buscan.

**Tipos de columnas**

En las tablas de hecho existen tres tipos de columnas:

* aditivas
* semi-aditivas
* no-aditivas

**Aplicación**

Su principal beneficio surge cuando debemos obtener métricas para armar, por ejemplo, reportes utilizando [**Power BI**](https://powerbi.microsoft.com/es-es/), Tableau u otra herramienta similar.

Nuestra bb.dd. podría incluir tablas de hecho sin relación alguna con el **Schema**, albergando solo los “*números importantes*”.

**Ejemplo**

Imaginemos que existe una tabla de **comprobantes de venta**, y queremos solo conocer el dinero total generado por ventas de un período y el total de comprobantes.

Esta información podría reflejarse directamente en una **tabla de hecho**.

**TABLAS DIMENSIONALES**

Ya tenemos las **tablas de hecho**, que concentran la información más importante de un negocio.

Veamos ahora cómo las **tablas dimensionales** se ocupan de nutrir y complementar el detalle de la información que hacen a las primeras.

**Concepto**

Las **tablas dimensionales** brindan información uniforme con datos asociados a otras entidades.

A diferencia de las **tablas de hecho**, estas portarán datos adicionales relacionados con la información de las tablas de hecho.

Volviendo al ejemplo de **total de facturas y montos** que vimos con las tablas de hecho, las dimensionales podrían alojar el resto de la información que complemente a dichos totales.

**Información**

Por ejemplo, la información que podrán alojar estas tablas, abarca desde los datos básicos del cliente comprador, hasta:

* datos demográficos detallados.
* datos de perfil de compra.
* preferencias del cliente.
* Información de promociones.
* etcétera.

**Beneficios**

Toda la información adicional y detallada permitirá tener **métricas efectivas** y generar **aportes de valor** a las campañas publicitarias y/o de Marketing.

Además, por supuesto, mantener una topología del tipo estrella, entre todo el cúmulo de información, permite aplicar la normalización de bases de datos de la forma más óptima.

**CLAVES**

**Implementación de las diferentes claves índices**

**Claves e índices**

En la **Clase 02** realizamos una aproximación teórica de la importancia de los índices y claves en las tablas de datos. Profundicemos ahora un poco más la funcionalidad que éstos aportan.

En el **ecosistema de bb.dd.** los índices son fundamentales, tanto para la normalización de la información, como también para su posterior búsqueda y relación entre tablas de almacenamiento.

Se los considera como un grupo de datos que vinculan a una o varias columnas de diferentes tablas, creando entre éstas una relación de contenido.

**Beneficio**

Podemos destacar la importancia de éstos para acelerar las consultas de información, algo que se vuelve mucho más efectivo cuando una tabla con datos crece considerablemente.

**Índice**

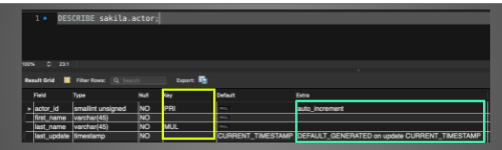
En el momento en el que definimos una tabla, es cuando el índice debe ser especificado correctamente.

Si bien las tablas pueden modificarse, aplicar un índice desde el principio es la mejor opción para todo el **schema**.

Podemos recurrir a la sentencia **DESCRIBE**, además de **Table Inspector**, para acceder a la información de los índices de una tabla.

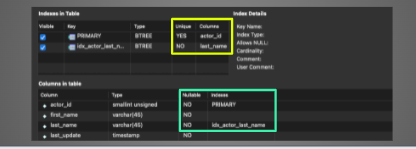
**Describe “tabla”**

Con **DESCRIBE** accedemos a la información propia de la construcción de dicha tabla, y veremos la columna **Key** con la información de los campos con índice, más la columna **Extra**, con la información adicional de éstos.

****

**Table Inspector**

En el apartado **Indexes** de la tabla, veremos el o los índices definidos en ésta. Accediendo, por ejemplo, a **sakila.actor**, veremos que la misma posee dos índices: uno primario y único, y el otro múltiple.

****

**Índices**

La columna **Key** muestra si el campo está indexado, a través de los indicadores:

* **PRI**: indica que el campo es una clave primaria.
* **UNI**: indica que el campo es un índice UNIQUE.
* **MUL**: indica si se permiten múltiples ocurrencias.

**CLAVES PRIMARIAS**

**Primary Key**

Este tipo de clave fue creado para poder **establecer consultas realmente veloces** sobre los datos almacenados en una tabla.

Siempre es recomendable aplicarlo sobre campos o columnas cuyos valores sean numéricos.

**Primary Key: único**

Otra característica de las claves primarias, es que **cada uno de los registros que se almacene en la tabla, deberá ser único**.

Por consiguiente, no podrá repetirse su valor en ningún otro registro que allí se guarde.

**Primary Key: AUTOINCREMENT**

Para evitar la duplicidad de datos en estas columnas, debemos agregar la propiedad **AUTOINCREMENT** al definirlas.

Nos evitará tener que implementar un proceso de validación previo por cada nuevo dato a agregar.

Genera un número incremental consecutivo en el campo definido como **Primary Key**. Inicia su numeración en **1**, y va de forma incremental por cada nuevo registro añadido.

Aún cuando eliminamos un registro de la tabla, al agregar el siguiente, SQL respetará el número consecutivo que le correspondía, tal como si el anterior aún existiese.

**CLAVES FORÁNEAS**

**Claves foráneas: Foreign Key**

La clave foránea establece una **limitación referencial entre dos tablas**, identificando un campo en una tabla que se referencia con el campo de otra.

Una oficia como **tabla Padre** (*o referenciada*) mientras que la otra será la **tabla Hija** (*o referendo*).

El campo de la tabla Padre siempre es la **Clave Primaria** y los registros de la tabla Hija no pueden tener valores que no existan en la tabla Padre.

Esto ocurre porque las referencias se crean para relacionar información, siendo ésta una **parte esencial de la Normalización de Base de Datos**.

**Condiciones**

Para utilizar esta vinculación entre dos tablas, debemos:

* Definir **Foreign Key** al crear la tabla desde el principio.
* Las tablas relacionadas deben ser del tipo **InnoDB**.
* La tabla Padre debe crearse primero, con su **Primary Key**.
* La tabla Hija debe referenciar **Foreign Key** a la tabla Padre.
* Ambos campos clave deben ser del mismo tipo de dato.

**Ejemplo**

La Tabla Padre (*Receta*) posee la **Clave Primaria**, y se relaciona con la Tabla Hija (*Ingredientes*) mediante el campo **id\_receta**, siendo ésta la **Clave Foránea**.

¿Se imaginan la tabla *Ingredientes* teniendo campos sin relacionar?



**Restricciones**

En este tipo de relación de datos, debemos prestar atención a eventos clave como son las actualizaciones/eliminaciones de registros o la inserción de nueva información.

**Restricciones ON UPDATE y ON DELETE**

Existen sub-cláusulas denominadas **ON UPDATE** y **ON DELETE** que se deben crear junto a la cláusula Foreign Key. Esto actuará en el momento en que un usuario desee eliminar o actualizar datos de la tabla padre.

Los tipos de respuestas que estas accionarán pueden ser (**CASCADE**, **NO ACTION**, **RESTRICT**)

**Se rechaza cualquier operación** **INSERT** en una **Tabla Hija** si no existe el valor de la clave foránea, en la **Tabla Padre**.

La operación correcta deberá ser: (*insertar un dato en la Tabla Padre, luego insertar el o los datos asociados en la Tabla Hija*).

**CLAVES CANDIDATAS**

**Definición**

Una clave candidata es un conjunto de atributos que identifican de manera unívoca cada tupla (*fila*) de una tabla SQL. La conforma un conjunto de columnas cuyos valores no se repiten en ninguna otra fila de dicha tabla.

En una tabla que almacena datos de **personas**, encontraremos seguramente dos o más con **nombre** y **apellido** coincidente. Si deseamos que el conjunto de datos de un registro no se repita, podemos optar por un tercer dato que conforme la **clave candidata**.

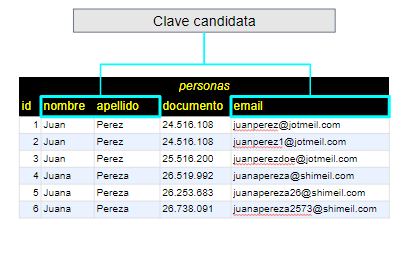
Por ejemplo, combinando **nombre** + **apellido** + **documento**, tendríamos una clave candidata que haga el registro; único.

* *¡Pero se puede duplicar un documento!*

Sí, es verdad. Para esto, podríamos elegir, en su lugar, la dirección de **correo electrónico**, o su **teléfono móvil**.

**Ejemplo**

De esta forma, alcanzaremos a contar con una clave candidata que nos evite la repetición de registros clave dentro de una tabla importante en la base de datos.

****

**CLAVES CONCATENADAS**

**Definición**

Una **clave concatenada**, también conocida como **clave compuesta**, es una clave primaria formada por más de un campo. Se usa en situaciones donde una clave primaria no satisface la necesidad de identificar unívocamente un registro, o situación.

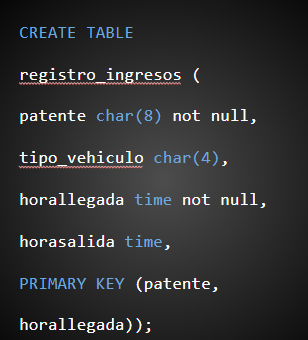
**Implementación**

Un caso óptimo sería un estacionamiento vehicular, donde guardamos nuestro vehículo de forma frecuente.

Si bien la patente podría definir una clave única, si ingresamos varias veces en un día, la tabla de registro de ingreso romperá su exclusividad en el registro.

**SQL**

Su implementación en una tabla SQL se realiza a través del atributo **PRIMARY KEY**, combinando en éste el nombre de cada columna, separado por una coma.

****