Bases de datos relacionales

Conceptos básicos

Facultad Regional Rosario
Universidad Tecnológica Nacional

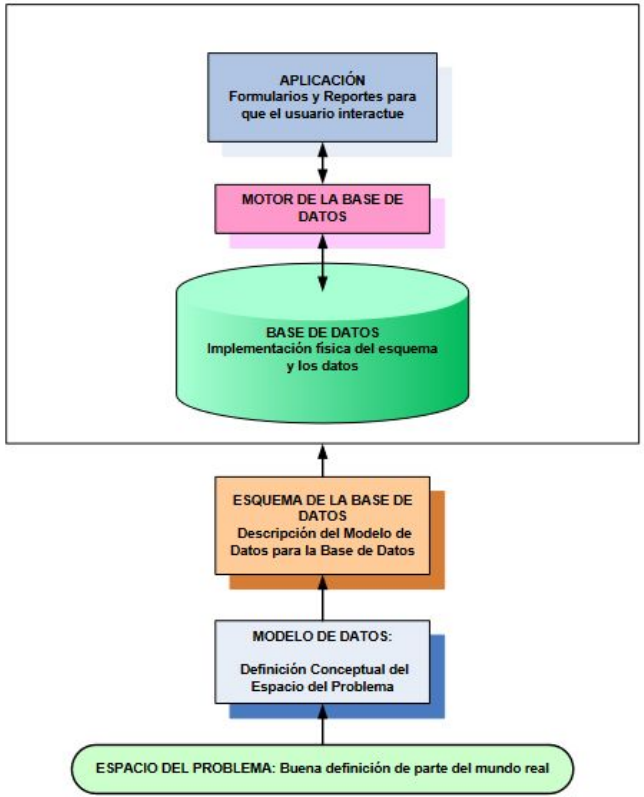
¿Qué es una bd relacional?

Una base de datos relacional es una implementación física del modelo relacional que es una forma de describir aspectos del mundo real de acuerdo con un conjunto de reglas que fueron inicialmente propuestas por Dr. E.F. Codd.

Una Base de Datos relacional almacena los datos en **relaciones** que el usuario percibe como **tablas**. Cada relación está compuesta por **tuplas (o registros)** y estos por **atributos (o campos)**.

No interesa el orden físico de los registros o campos en una tabla y **cada registro está identificado unívocamente por una clave**. Estas son las características que permiten existir a una base de datos relacional independiente del modo en que está almacenada.

Terminología usual



Motor de base de datos: Refiere a una arquitectura física en particular. Son los elementos que se encargarán de almacenar, gestionar y recuperar toda la información de una tabla. Ejemplos de motores son Oracle Server, SQL Server, InnoDB.

La diferencia entre ellos descansa en sus arquitecturas y los problemas que resuelven.

Metodología de diseño de una bd relacional

1. **Definir los requerimientos del sistema**: Idealmente, cada proyecto debe comenzar con una clara definición de qué es lo que se trata de conseguir, por qué se debe conseguir y cómo se considerará si se ha alcanzado o no el éxito esperado.
2. **Reconocer el proceso al que están afectados los datos**: Los usuarios no almacenan datos con el único objetivo de almacenarlos sino que requieren utilizarlos. Entender el proceso por el que atraviesan los datos es crucial.
3. **Construir el modelo conceptual de datos**: es quien define la utilización de los datos en el sistema. Esto incluye, no sólo el modelo lógico de datos, sino una descripción de cómo el proceso interactúa con los datos.
4. **Preparar el esquema de la base de datos**: Se refiere a transformar el modelo conceptual de datos a términos lógicos y físicos que se administrarán a través del SGBD seleccionado.
5. **Diseñar la interfaz de usuario**: debe ser clara y amigable, para que el usuario pueda relacionarse con la base de datos sin necesidad de entender ningún proceso detrás.

Creación de Tablas

La estructura de almacenamiento de los datos del modelo relacional son las tablas

```
CREATE TABLE nombre_tabla
( definición_columna
[, definición_columna...]
[, restricciones_tabla]);
```

Donde definición_columna es:

```
nombre_columna {tipo_datos|dominio} [def_defecto] [restric_col]
```

A cada una de las columnas se le asigna un tipo de datos predefinido o bien un dominio definido por el usuario. También podremos dar definiciones por defecto y restricciones de columna. Una vez definidas las columnas, sólo nos quedará dar las restricciones de tabla.

Tipos de datos

Tipos de datos predefinidos	
Tipos de datos	**Descripción**
CHARACTER (longitud)	Cadenas de caracteres de longitud fija.
CHARACTER VARYING (longitud)	Cadenas de caracteres de longitud variable.
CHARACTER LARGE OBJECT	Cadena de caracteres de longitud variable hasta el máximo definido por la implementación de la BD
BIT (longitud)	Cadenas de bits de longitud fija.
BOOLEANO (bit(1))	V o F
BINARY LARGE OBJECT	Cadena de bit de longitud variable hasta el max permitido por la implementación. (permite guardar imágenes)
BIT VARYING (longitudb)	Cadenas de bits de longitud variables.
NUMERIC (precisión, escala)	Número decimales con tantos dígitos Como indique la precisión y tantos decimales como
DECIMAL (precisión, escala)	Número decimales con tantos dígitos Como indique la precisión y tantos decimales como
INTEGER	Números enteros.
SMALLINT	Números enteros pequeños.
REAL	Números con coma flotante con precisión predefinida.
FLOAT (precisión)	Números con coma flotante con la precisión especificada.
DOUBLE PRECISION	Números con coma flotante con más precisión predefinida que la del tipo REAL.
DATE	Fechas. Están compuestas de: YEAR año, MONTH mes, DAY día.
TIME	Horas. Están compuestas de HOUR hora, MINUT minutos, SECOND segundos y fraccion de Segundos
TIMESTAMP	Fechas y horas. Están compuestas de YEAR año, MONTH mes, DAY día, HOUR hora, MINUT minutos, SECOND

Restricciones de columna

Una vez les hemos dado un nombre a las tablas y se ha definido su dominio, podemos imponer ciertas restricciones que siempre se tendrán que cumplir. Las restricciones que se pueden dar son:

Restricciones de columna	
Restricción	Descripción
NOT NULL	La columna no puede tener valores nulos.
UNIQUE	La columna no puede tener valores repetidos. Es una clave
PRIMARY KEY	La columna no puede tener valores repetidos ni nulos. Es la clave primaria.
REFERENCES Tabla [(columna)]	La columna es la clave foránea de la columna de la tabla especificada.
CHECK (condiciones)	La columna debe cumplir las condiciones especificadas.

Restricciones de tabla

Una vez que definido una tabla y hemos impuesto ciertas restricciones para cada una de las columnas, podemos aplicar restricciones sobre toda la tabla, que siempre se deberán cumplir. Las restricciones que se pueden dar son las siguientes

Restricciones de tabla	
Restric	Descripción
UNIQUE (columna [, columna. . .])	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa.
PRIMARY KEY (columna [, columna. . .])	El conjunto de las columnas especificadas no puede tener valores nulos ni repetidos. Es una clave primaria.

Restricciones de tabla

Restricciones de tabla	
Restric	Descripción
UNIQUE (columna [, columna. . .])	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa.
PRIMARY KEY (columna [, columna. . .])	El conjunto de las columnas especificadas no puede tener valores nulos ni repetidos. Es una clave primaria.
FOREIGN KEY (columna [, columna. . .]) REFERENCES tabla [(columna2 [, columna2. . .])]	El conjunto de las columnas especificadas es una clave foránea que referencia la clave primaria formada por el conjunto de las columnas2 de la tabla dada. Si las columnas y las columnas2 se denominan exactamente igual, entonces no sería necesario poner columnas2.
CHECK (condiciones)	La tabla debe cumplir las condiciones especificadas.

Ejemplos de creación de tablas

```
CREATE TABLE `clientes' (
        'tipo_doc' varchar(4) NOT NULL
        `nro_doc' int(11) NOT NULL
        'nom_ape' varchar(50) NOT NULL,
        'tel' VARCHAR(2) default NULL,
        `dir`varchar(50) default NULL
)
CREATE TABLE 'empleados' (
        'cuil`varchar(20) NOT NULL PRIMARY KEY ,
        'nom_ape' varchar(50) NOT NULL,
        sexo char(1) CHECK VALUE IN ("F","M","X")
)
```

Ejemplos de creación de tablas

```
CREATE TABLE `procesos_realizados` (
        `nro_servicio` int(11) NOT NULL,
        `orden` int(11) NOT NULL,
        `cod_proceso` int(11) NOT NULL,
        `fecha_inicio` date NOT NULL,
        `hora_inicio` time NOT NULL,
        `cuil_empleado` varchar(20) NOT NULL,
        `fecha_fin` date default NULL,
        `hora_fin` time default NULL,
        `resultado_proceso` varchar(20) default NULL,
PRIMARY KEY (`nro_servicio`,`orden`,`fecha_inicio`,`hora_inicio`),
KEY `procesos_realizados_fk1` (`cod_proceso`),
KEY `procesos_realizados_fk2` (`cuil_empleado`),
CONSTRAINT `procesos_realizados_fk` FOREIGN KEY (`nro_servicio`, `orden`) REFERENCES `tratamiento_limpieza`
(`nro_servicio`, `orden`)
ON UPDATE CASCADE,
CONSTRAINT `procesos_realizados_fk1` FOREIGN KEY (`cod_proceso`) REFERENCES `procesos` (`cod_proceso`) ON
UPDATE CASCADE,
CONSTRAINT `procesos_realizados_fk2` FOREIGN KEY (`cuil_empleado`) REFERENCES `empleados` (`cuil`) ON UPDATE
CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Modificación y borrado de tablas

Para modificar una tabla es preciso utilizar la sentencia ALTER TABLE:

ALTER TABLE nombre_tabla {acción_modificar_columna|
acción_modif_restricción_tabla};

Donde:

acción_modificar_columna:
{ADD [COLUMN] columna def_columna |
ALTER [COLUMN] columna {SET def_defecto|DROP DEFAULT}|
DROP [COLUMN] columna {RESTRICT|CASCADE}}

acción_modif_restricción_tabla:
{ADD restricción|
DROP CONSTRAINT restricción
RESTRICT|CASCADE}}

Modificación y borrado de tablas

Modificar una tabla puede implicar:

- Añadir una columna (ADD columna).
- Modificar las definiciones por defecto de la columna (ALTER columna).
- Borrar la columna (DROP columna).
- Añadir alguna nueva restricción de tabla (ADDCONSTRAINT restricción).
- Borrar alguna restricción de tabla (DROPCONSTRAINT restricción).

Para borrar una tabla es preciso utilizar la sentencia DROP TABLE:

```
DROP TABLE nombre_tabla {RESTRICT|CASCADE};
```

Donde:
RESTRICT, la tabla no se borrará si está referenciada, por ejemplo, por alguna vista.
CASCADE, todo lo que referencie a la tabla se borrará con ésta.

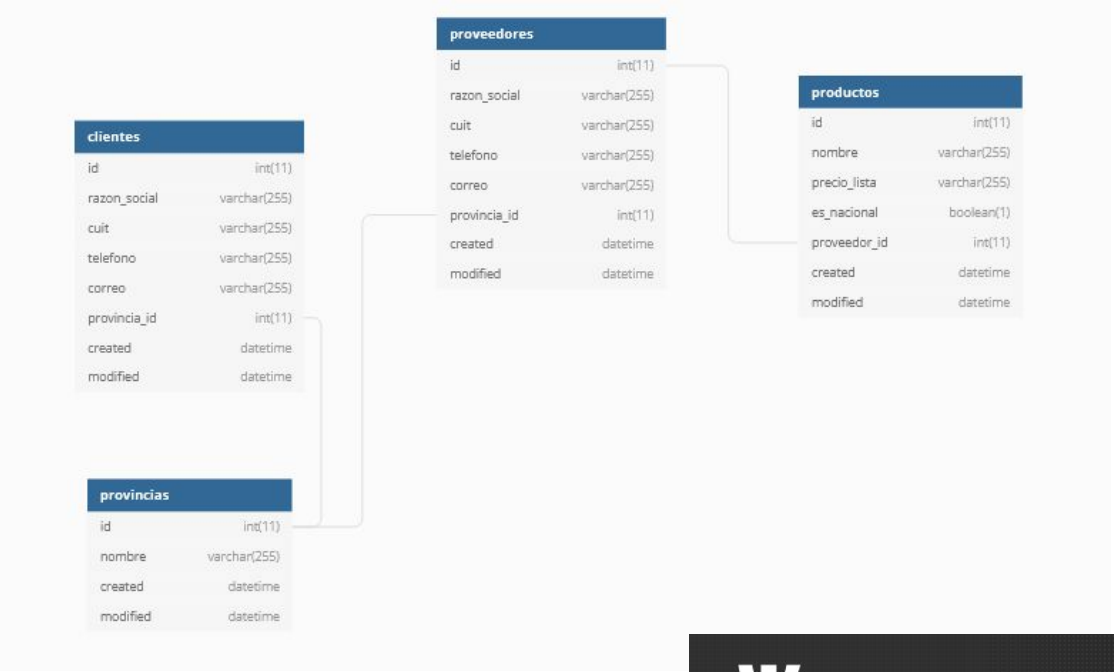
Eliminar contenido de una tabla

TRUNCATE TABLE nombre_tabla;

Las operaciones de truncado destruyen y recrean la tabla, que es mucho más rápido que borrar registros uno a uno.
Los datos AUTO_INCREMENT empiezan a contar desde el principio.

Ejercicios

Crear las siguientes tablas



Creación de vistas

Las vistas son tablas ficticias, denominadas derivadas.
Se construyen a partir de tablas reales almacenadas en la base de datos
La no existencia real de las vistas hace que puedan ser actualizables o no.
Simplifican las consultas generando independencia de los datos.
Podemos clasificar a las vistas en

- Simples: Poseen una sola tabla, no contiene funciones, no contiene grupos.
- Compuestas: Poseen una o más tablas, contienen funciones y/o funciones de grupo.

Creación de vistas

CREATE VIEW nombre_vista [(lista_columnas)]AS (consulta)
[WITH CHECK OPTION];

Donde consulta es una cláusula SELECT que hace referencia a la tabla de donde deriva la vista.

DROP VIEW nombre_vista (RESTRICT|CASCADE);

Donde:
RESTRICT, la vista no se borrará si está referenciada,por ejemplo, por otra vista.
CASCADE, todo lo que referencie a la vista se borrará con ésta.

Operaciones básicas

Comandos que se aplican a una tabla

- INSERT: para agregar filas a una tabla,
- UPDATE: para modificar filas de una tabla,
- DELETE: para borrar filas de una tabla

Comandos que pueden aplicarse a una o más tablas

- SELECT FROM

Inserción de registros

INSERT INTO nombre_tabla [(columnas)]
    {VALUES({v1|DEFAULT|NULL}, ...,
    {vn/DEFAULT/NULL})

INSERT INTO nombre_tabla[(columnas)]
    {clausula SELECT};

En el caso de utilizar SELECT se produce una tabla temporal. El SELECT se evalúa antes que inicie la operación INSERT.

Modificación y borrado de registros

```
UPDATE nombre_tabla
SET columna = {expresión | DEFAULT | NULL}
[, columna = {expr | DEFAULT | NULL} ...]
[, columna = {expr | DEFAULT | NULL} ...]
WHERE condiciones;

DELETE FROM nombre_tabla
[WHERE condiciones];
```

Consultas simples

```
SELECT [DISTINCT]
*|
nombre_columna_a_seleccionar1 [[AS] alias_col1],
nombre_columna_a_seleccionar 2[[AS] alias_col2]...
nombre_columna_a_seleccionar n[[AS] alias_coln]
nombre_columna_a_seleccionar n[[AS] alias_coln]

FROM tabla_a_consultar [[AS] alias_tabla]

WHERE condiciones;
```

Ejemplo de consulta simple

Listar los números de CUIT y la Razón Social de los Clientes y Proveedores que tiene registrada la Ferretería.

```
SELECT per.`Cuit`, per.`Razon_Social`
SELECT per.`Cuit`, per.`Razon_Social`
FROM personas per
```

per es un ALIAS de la tabla personas

Ejemplo de consulta simple

Listar las localidades que pertenezcan a la provincia de Santa Fe (cuyo código es 1)

```
SELECT loc.`Cod_Postal`, loc.`ciudad`
FROM localidades loc
WHERE loc.`Id_Pcia` = 1
```

Palabras y símbolos para consultas

Las expresiones de valores

- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (%)

Conectores Lógicos

- AND
- OR
- NOT

Palabras y símbolos para consultas

Predicados

- Comparación: (=, <>, <, >, <=, =>)
- Entre (...BETWEEN...AND...)
- IN, (NOT IN)
- LIKE
- NULL
- Cuantificador (ALL, SOME, ANY)
- EXISTS, (NOT EXISTS)

Subconsultas

Una subconsulta es una consulta incluida dentro de una cláusula WHERE o HAVING de otra consulta.
En ocasiones, para expresar ciertas condiciones no hay otra alternativa que obtener el valor que buscamos otra alternativa que obtener el valor que buscamos como resultado de una consulta.

SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
WHERE columna operador_comparación subconsulta;

Ejemplo de subconsulta

Listar las localidades(Código Postal y Ciudad) de aquellas Localidades donde no haya Clientes ni Proveedores registrados.

```
SELECT loc.`Cod_Postal`, loc.`ciudad`
FROM localidades loc
WHERE loc.cod_postal
NOT IN (SELECT per.`Cod_Postal` FROM personas per)
```

Combinar tablas

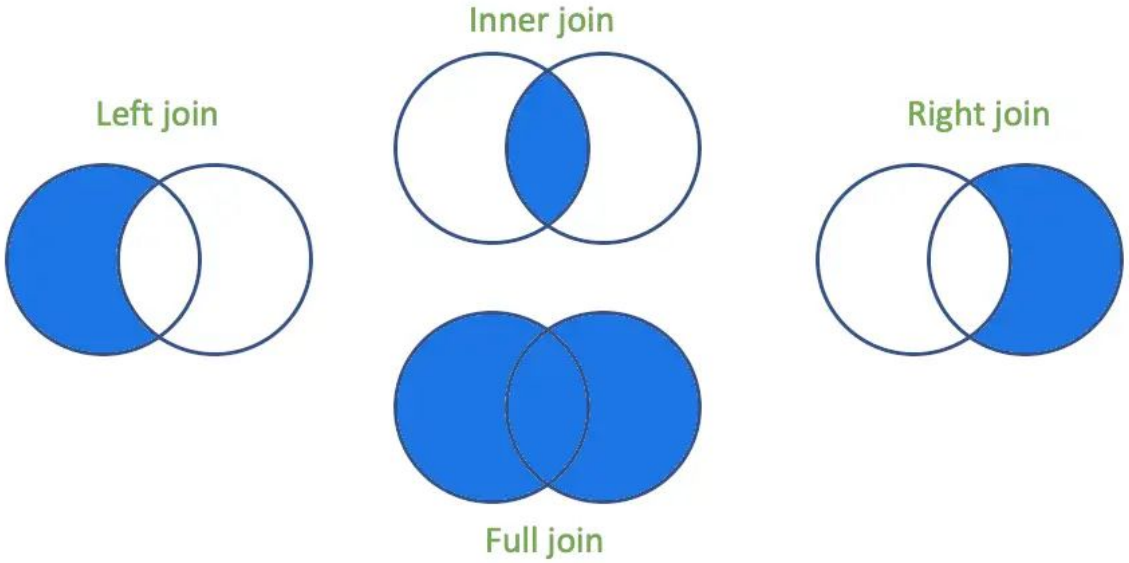
Cuando requerimos datos de tablas relacionadas necesitaremos enlazar las mismas. Podemos hacerlo a través del uso de:

- PRODUCTO CARTESIANO
- INNER JOIN
- LEFT JOIN - RIGHT JOIN
- SELF JOIN

Ejemplo Producto cartesiano

```
SELECT loc.`Cod_Postal`, loc.`ciudad`, pro.`Nom_pcia`
FROM `localidades` loc, provincias pro
WHERE loc.`Id_Pcia` = pro.`Id_Pcia`
```

Teoría de conjuntos para los JOINS



Inner Join

INNER JOIN es una combinación por equivalencia, conocida también como unión interna.
Las combinaciones equivalentes son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en atributos comunes a ambas tablas.

Hay que tener en cuenta que esto puede hacer que perdamos alguna fila interesante de alguna de las dos tablas; por ejemplo, porque se encuentra en NULL en el momento de hacer la combinación.

Ejemplo Inner Join

Listar los números de CUIT, Razón Social, Dirección, código postal, Nombre de Localidad y Nombre de provincia de los Clientes y Proveedores que tiene registrada la Ferretería y pertenecen a la provincia de Santa Fe (cuyo código es 1)

```
SELECT per.`Cuit`, per.`Razon_Social`, per.`Direccion`,loc.`ciudad`,pro.`Nom_pcia
FROM personas per
INNER JOIN localidades loc
ON per.`Cod_Postal` = loc.`Cod_Postal`
INNER JOIN provincias pro
ON pro.`Id_Pcia` = loc.`Id_Pcia`
WHERE pro.`Id_Pcia` = 1
```

Ejemplo Inner Join

Si deseáramos obtener los productos que vende la ferretería,
los proveedores a los que los compra y sus precios:

```
SELECT prod.`ID_Producto`, prod.`Nombre_producto`, prop.`Cuit`,
per.`Razon_Social`, prec.`Fecha_Desde`, prec.`Precio`
FROM productos prod
INNER JOIN productos_proveedores prop
ON prop.`ID_Producto` = prod.`ID_Producto`
INNER JOIN personas per
ON per.`Cuit` = prop.`Cuit`
INNER JOIN `precios` prec
ON prop.`Cuit` = prec.`Cuit`
AND prop.`ID_Producto` = prec.`ID_Producto`
```

Left Join

El problema con la consulta anterior es que no aparecen aquellos productos de os proveedores para los que no se tienen precios. Para resolverlo utilizamos LEFT JOIN:

```
SELECT prod.`ID_Producto`, prod.`Nombre_producto `, prop.`Cuit`,
per.`Razon_Social`, prec.`Fecha_Desde`, prec.`Precio`
FROM productos prod
INNER JOIN productos_proveedores prop
ON prop.`ID_Producto` = prod.`ID_Producto`
INNER JOIN personas per
ON per.`Cuit` = prop.`Cuit`
LEFT JOIN `precios` prec
ON prop.`Cuit` = prec.`Cuit`
AND prop.`ID_Producto` = prec.`ID_Producto`
```

Left Join

El LEFT JOIN permite que aparezcan en la consulta todos los registros de la tabla de la izquierda aunque los mismos no tengan ninguna correspondencia con la tabla con la que se combinan, completando los atributos que se muestran de la tabla de la derecha con valores NULL.

Orden de las filas en una consulta

Si se necesita que las filas aparezcan en un orden determinado se deberá utilizar la cláusula ORDER BY en la sentencia SELECT

SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
ORDER BY columna_orden1 [DESC]
[, columna_orden2 [DESC]...
, columna_ordenn [DESC]...] };

Ejemplo de Orden en una consulta

Listar los números de CUIT, Razón Social, Dirección, código postal, Nombre de Localidad y Nombre de la provincia de los Clientes y Proveedores que tiene registrada la Ferretería ordenados por provincia y por localidad

```
SELECT per.`Cuit`, per.`Razon_Social`, per.`Direccion`,loc.`ciudad`, pro.`Nom_pcia`
FROM personas per
INNER JOIN localidades loc
ON per.`Cod_Postal` = loc.`Cod_Postal`
INNER JOIN provincias pro
ON pro.`Id_Pcia` = loc.`Id_Pcia`
ORDER BY pro.`Nom_pcia`, loc.`ciudad`
```

Funciones de agregación

Función	Descripción
COUNT	Nos da el número total de filas seleccionadas
SUM	Suma los valores de una columna
MIN	Nos da el valor mínimo de una columna
MAX	Nos da el valor máximo de una columna
AVG	Calcula el valor promedio de una columna

Ejemplos de Agregación

Cantidad de personas registradas:
SELECT COUNT(*)
FROM personas per

Cantidad de productos total en stock:
SELECT SUM(prod.`Stock`)
FROM productos prod

Máximo número de Pedido registrado:
SELECT MAX(ped.`ID_Pedido`)
FROM pedidos ped

Funciones de Agrupación

Es probable que necesitemos realizar las consultas anteriores pero para determinados valores de un atributo.

Para ello deberemos recurrir a las funciones de AGRUPACIÓN:

GROUP BY nos sirve para agrupar
HAVING especifica condiciones de búsqueda para grupos de filas; lleva a cabo la misma función que cumple la cláusula WHERE para las filas de toda la tabla, pero ésta aplica las condiciones a los grupos obtenidos.

Ejemplos de Agrupación

Cantidad de personas registradas por localidad:

```
SELECT per.`Cod_Postal`, COUNT(*)
FROM personas per
GROUP BY per.`Cod_Postal`
```

Productos de los cuales se haya solicitado más de 10:

```
SELECT det.`Cuit`, det.`ID_Producto`,prod.`Descripc_Prod`, SUM(det.`Cantidad`)
FROM `detalle_pedidos` det
INNER JOIN productos prod
ON prod.`ID_Producto` = det.`ID_Producto`
INNER JOIN personas per
ON per.`Cuit` = det.`Cuit`
GROUP BY det.`Cuit`, det.`ID_Producto`
HAVING SUM(det.`Cantidad`) > 10
```

Optimización de consultas

Las malas prácticas en la formulación de una consulta pueden traducirse en consultas lentas cuyo rendimiento no hace sino empeorar conforme la cantidad de datos que almacenas crece, por poner un ejemplo, lo que era una consulta de 200 ms, se convierte en una de 4 seg cuando la tabla registra cientos de miles de datos.

En el trabajo de optimización de rendimiento no hay nada más importante que la medición de los resultados, medir es el primer paso para mejorar el rendimiento de cualquier cosa. Con esto comprobaremos que los cambios introducidos realmente mejoran el rendimiento.

EXPLAIN nos permite obtener información sobre cómo se llevarán a cabo las consultas.

Índices

Los índices, al igual que en la vida real los índices de los libros, nos sirven para encontrar más rápido aquello que buscamos, por lo tanto podemos decir que nos sirven para agilizar las consultas a las tablas.

Ventajas de los Índices

- Una de las mayores ventajas es que cuando se encuentra un índice **evitamos un “escaneo completo de la tabla”** lo que hace que cuando tenemos grandes cantidades de datos en nuestras tablas, la mejora puede ser muy importante. Con esto evitamos sobrecarga de CPU, sobrecarga de disco y concurrencia.
- Con los índices **evitamos hacer lecturas secuenciales**.
- Los índices nos permiten una **mayor rapidez en la ejecución de las consultas** tipo SELECT ... WHERE ...
- Y por último será una **ventaja para aquellos campos que no tengan datos duplicados**, sin embargo si es un campo con valores que se repiten continuamente (Ej. Masculino/Femenino) no es aconsejable.

Desventajas de los Índices

A pesar de sus grandes ventajas no debemos abusar de ellos puesto que en determinadas situaciones no supondrá una mejora:

- Los índices son una desventaja en aquellas tablas las que se utiliza frecuentemente operaciones de escritura (Insert, Delete, Update), esto es porque **los índices se actualizan cada vez que se modifica una columna**.
- Los índices también suponen una **desventaja en tablas demasiado pequeñas** puesto que no necesitaremos ganar tiempo en las consultas.
- Tampoco son muy aconsejables cuando pretendemos que la tabla sobre la que se aplica **devuelva una gran cantidad de datos en cada consulta**.
- Por último hay que tener en cuenta que **ocupan espacio** y en determinadas ocasiones incluso más espacio que los propios datos.

Desventajas de los Índices

Si tenemos una consulta en la que tenemos claro el uso que vamos a necesitar de la claúsula WHERE es recomendable, por el contrario si tenemos un gran número de registros duplicados y lo que necesitamos la gran mayoría de veces es una lectura secuencial la respuesta es no

Índices - ejemplo

Supongamos que tenemos una tabla de clientes con 15000 registros y debemos importar un archivo con 5000 registros, pero debemos validar que no esté cargado ya para que no se repita.
Nuestro negocio lleva un control interno de los cliente en base a un número de cliente (nro_cliente), asi que podemos usar el campo para validar no cargar 2 veces un mismo cliente.
Al intentar hacer la importación podemos notar que esta lleva mucho tiempo y se debe a que nuestro sistema hace una consulta por cada *nro_cliente*.
Podemos verificar los tiempos y registros recorridos haciendo:

```
EXPLAIN SELECT * FROM `clientes` WHERE nro_cliente > 123;
```

Si agregamos un índice para esa columna, y volvemos a consultar el comando anterior, podemos notar como baja drásticamente los tiempo y cantidad de registros que indexa.

Ejercicios

1. Listar los correos, teléfonos y la Razón Social de los Clientes que tiene registrada la Ferretería.
2. Listar los CUIT de los Clientes que tiene registrados la ferretería en la provincia de Entre Ríos (el id de esa provincia es 4).
3. Obtener los productos que vende la ferretería, los proveedores a los que los compra y sus precios
4. Cantidad de detalles de pedidos por producto de cada proveedor