CONSULTAS RELACIONADAS

Consultas de más de una tabla

Es posible consultar datos desde varias tablas en la misma sentencia **SELECT**.
Esto permite realizar otras dos operaciones de álgebra relacional: el **producto cartesiano** y la **composición interna**.

Producto cartesiano
Se obtiene mencionando las dos tablas en una consulta sin ninguna restricción en la cláusula **WHERE**.
El **producto cartesiano** de dos tablas son todas las combinaciones de todas las filas de las dos tablas. Usando una sentencia **SELECT** se deben listar todos los campos de ambas tablas. Los nombres de las tablas se indican en la cláusula **FROM** separados con comas.

Composición interna
La **composición interna** se trata de un **producto cartesiano restringido** en donde las tuplas (conjunto de nombres de atributos relacionados) que se emparejan deben cumplir una determinada condición.

Sintaxis

```
SELECT * FROM Productos, Marcas
WHERE Productos.Marca = Marcas.idMarca;
```

Tabla: Productos								Tabla: Marcas	
idProducto	Nombre	Precio	Marca	Categoria	Presentacion	Stock	Disponible	idMarca	Nombre
1	iPhone 6	499.99	1	Smartphone	16GB	500	SI	1	Apple
2	iPad Pro	599.99	1	Smartphone	128GB	300	SI	2	Samsung
3	Nexus 7	299.99	4	Smartphone	32GB	250	NO	3	Huawei
4	Galaxy S7	459.99	2	Smartphone	64GB	200	SI	4	LG
5	Impresora T23	489.99	8	Impresoras	Color	180	NO	5	Motorola
6	Impresora T33	399	8	Impresoras	Color	200	NO	6	Google
7	Lavarropa 7000	1679	4	Lavarropas	Automático	100	SI	7	HP
8	Camara Digital 760	649	9	Fotografía	Sin detalle	150	NO	8	Epson
9	Notebook CQ40-300	2999	7	Notebooks	Intel Core i3	100	SI	9	Kodak

Analizando el panorama, se ve que en la tabla **Productos** el campo **idProducto** es Clave Primaria y por ende no puede tener valores repetidos. Pero en el campo **Marca** el valor numérico hace referencia a qué Marca pertenece el producto. Una marca puede tener ningún producto asociado, uno ó muchos.

Sintaxis

```
SELECT * FROM Productos, Marcas;
```

idProducto	Nombre	Precio	Marca	Categoria	Presentacion	Stock	Disponible	idMarca	Nombre
1	iPhone 6	499.99	1	2	16GB	500	SI	1	Apple
1	iPhone 6	499.99	1	2	16GB	500	SI	2	Samsung
1	iPhone 6	499.99	1	2	16GB	500	SI	3	Huawei
1	iPhone 6	499.99	1	2	16GB	500	SI	4	LG
1	iPhone 6	499.99	1	2	16GB	500	SI	5	Motorola
1	iPhone 6	499.99	1	2	16GB	500	SI	6	Google
1	iPhone 6	499.99	1	2	16GB	500	SI	7	HP
1	iPhone 6	499.99	1	2	16GB	500	SI	8	Epson
1	iPhone 6	499.99	1	2	16GB	500	SI	9	Kodak

Así se obtiene la combinación de todos los registros de la primera tabla con todos los registros de la segunda tabla. Vale aclarar que la concordancia lógica de los datos jugará un rol importante a la hora de ejecutar este tipo de consultas

Modelo de Entidad - Relación

Introducción

Cuando se utiliza una base de datos para gestionar información se está plasmando una parte del mundo real en una serie de tablas, registros y campos ubicados en un dispositivo; creándose un modelo parcial de la realidad. Antes de crear físicamente estas tablas se debe realizar un modelo de datos.
El modelo entidad-relación (E-R) es uno de los varios modelos conceptuales existentes para el diseño de bases de datos.
Se suele cometer el error de ir creando nuevas tablas a medida que se van necesitando, haciendo así el modelo de datos y la construcción física de las tablas simultáneamente.
El modelo de datos más extendido es el denominado ENTIDAD/RELACIÓN (E/R). En el modelo E/R se parte de una situación real a partir de la cual se definen entidades y relaciones entre dichas entidades.

Entidad

Una entidad es cualquier "objeto" discreto sobre el que se tiene información. Cada ejemplar de una entidad se denomina **instancia.** Las entidades son modeladas en la base de datos como tablas.

INTEGRIDAD REFERENCIAL

La integridad referencial es un mecanismo que garantiza la integridad de datos en tablas relacionadas, ya que la misma evita la existencia de los llamados registros huérfanos (registros hijos sin su correspondiente registro padre). Para establecer la integridad referencial es necesario crear en una tabla hija una clave externa o foránea que esté relacionada a una clave primaria de la tabla padre. Es posible establecer el comportamiento de los registros en la tabla hija cuando se producen actualizaciones de datos en la clave primaria de la tabla padre o se eliminan registros en la tabla padre a través de la definición de operaciones en cascada: ON UPDATE, ON DELETE.

Clave Foránea (FOREIGN KEY)

La Clave Foránea referencia a la clave primaria de una tabla. Esta puede referenciar a la clave primaria de la misma tabla o de otra. Ante una consulta SQL se valida la legitimidad de los datos almacenados en una clave foránea y se fuerza la integridad referencial.

La Clave Foránea debe tener el mismo tipo de datos que el campo al cual hace referencia.

Sintaxis

```
CREATE TABLE Facturas(Letra char NOT NULL, Numero int,
id_articulo int UNSIGNED NOT NULL, -- Coincide en tipo/longitud con el campo al que será relacionado
PRIMARY KEY (Letra, Numero),
);

CREATE TABLE Articulos(id_articulo INT UNSIGNED NOT NULL AUTO_INCREMENT,
Nombre VARCHAR(30) NOT NULL,
PRIMARY KEY (id_articulo) -- Al definir una Primary Key, ésta podrá tener como enlace una Foreign Key
);

ALTER TABLE Facturas ADD FOREIGN KEY(id_articulo) REFERENCES Articulos(id_articulo) ON DELETE CASCADE ON
UPDATE CASCADE;
```

Conceptos Claves

Super llave
Es un conjunto de uno o más atributo que "juntos" identifican de manera única a una entidad.

Es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar de forma única un registro en el conjunto de registros. Es un conjunto de atributos mediante los cuales es posible reconocer a un registro. Este tipo de llaves contiene comúnmente atributos ajenos; es decir, atributos que no son indispensables para llevar a cabo el reconocimiento del registro.

Clave candidata
Llave candidata: es una súper llave mínima. Una tabla puede tener varias llaves candidatas, pero solo una es elegida como llave primaria.

Relación
Una relación describe cierta interdependencia (de cualquier tipo) entre una o más entidades. Esta no tiene sentido sin las entidades que relaciona. Las relaciones son definidas con claves primarias y claves foráneas y mantienen la integridad referencial.

Cardinalidad de las Relaciones
Una relación describe cierta interdependencia (de cualquier tipo) entre una o más entidades. Las relaciones pueden ser:

- **de uno a uno**: una instancia de la entidad A se relaciona con una y solamente una de la entidad B.
- **de uno a muchos:** cada instancia de la entidad A se relaciona con varias instancias de la entidad B.
- **de muchos a muchos**: cualquier instancia de la entidad A se relaciona con cualquier instancia de la entidad

Atributos
Las entidades tienen **atributos**. Un atributo de una entidad representa alguna propiedad que nos interesa almacenar En el modelo de Bases de Datos, los atributos son almacenados como columnas o campos de una tabla.

Consideraciones en el Planeamiento del Diseño Lógico de la Base de Datos

- Determinar el negocio y las necesidades del usuario.
- Considerar cuáles son los problemas que hay que resolver y las tareas que los usuarios deberán completar
- Crear Bases de Datos Normalizadas.
- Evitar el almacenamiento de información duplicada, inconsistencias en la base de datos, anomalías y problemas de pérdida de la información.

Conceptos Avanzados – JOIN

Recordemos que en un SELECT, cuya sintaxis básica es:

SELECT campos
FROM tabla
WHERE condición

podemos distinguir tres partes:

1. **campos** donde se enumeran las columnas a devolver
2. **tabla** donde se especifica la ó las tablas donde están los datos
3. **condición** donde se colocan las condiciones que deben cumplirse para restringir las filas del resultado de la consulta

Hasta ahora hemos utilizado sólo una tabla en cada consulta, a continuación veremos cómo se hace para combinar datos que provienen de más de una tabla. Cuando se quiere consultar campos que se encuentran en distintas tablas, es necesario combinar las mismas a partir de la cláusula **JOIN** mediante un campo en común.
La cantidad de JOINS es igual a la cantidad de tablas que participan en la consulta -1.

Sintaxis alternativas de **JOIN**
A continuación, vamos a ver dos maneras posibles de combinar tablas mediante la cláusula JOIN para aquellos casos en los que el campo en común no tiene el mismo nombre. Las sintaxis son las siguientes:

1. SELECT tabla1.campos, tabla2.campos
   FROM tabla1
   JOIN tabla2 ON tabla1.campo1=tabla2.campo1
   WHERE condición
2. SELECT tabla1.campos, tabla2.campos
   FROM tabla1, tabla2
   WHERE tabla1.campo1=tabla2.campo1
   AND condición

- Ambas son válidas, es cuestión de utilizar el código con el cual uno se sienta más cómodo.
- Hay que tener en cuenta que en ambos casos, si los campos por los cuales se combinan las tablas tienen el mismo nombre, hay que escribirlos en el SELECT de la forma “tabla.campo” para especificar a qué tabla pertenece el campo.
- En todos los tipos de JOIN al referirse a la tabla de la izquierda estamos hablando de la tabla especificada en el FROM, y la tabla de la derecha es la tabla definida a continuación de JOIN.
- En las sintaxis 1 y 2, la tabla de la izquierda es tabla1 y la de la derecha, tabla2.

Tipos de JOIN

Los mismos pueden ser:

1. **[INNER] JOIN**
2. **LEFT [OUTER] JOIN**
3. **RIGHT [OUTER] JOIN**
4. **CROSS JOIN**

A continuación, explicaremos cada JOIN con un ejemplo, suponiendo:

Tabla1

Codigo (int)	**Nombre** (varchar(15))
1	A
3	C
8	H

Tabla2

Codigo (int)	**Nombre2** (varchar(15))
3	Tres
5	Cinco
8	Ocho

1. [INNER] JOIN

Este JOIN devuelve solamente los registros que coinciden:

```
SELECT *
FROM tabla1
JOIN tabla2
ON tabla1.codigo = tabla2.codigo;
```

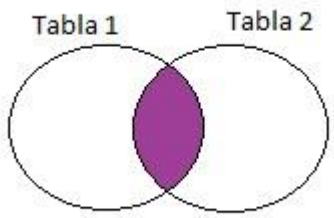
Podemos observar como resultado de la combinación de la tabla1 y la tabla2, los siguientes registros:

3	c	tres
8	h	ocho

Esta consulta también puede escribirse:

```
SELECT *
FROM tabla1
INNER JOIN tabla2
ON tabla1.codigo = tabla2.codigo;
```

Si hacemos una analogía con los diagramas de Venn (teoría de conjuntos), donde cada conjunto es una tabla, podemos expresar el resultado del [INNER] JOIN del siguiente modo:



2. LEFT [OUTER] JOIN

Este JOIN devuelve todos los registros de la tabla de la izquierda y los registros que coinciden de la tabla de la derecha:

```
SELECT *
FROM tabla1
LEFT JOIN tabla2
ON tabla1.codigo = tabla2.codigo;
```

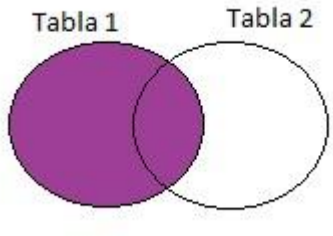
Obteniendo como resultado de la combinación de la tabla1 y la tabla2, los siguientes registros:

1	a	
3	c	tres
8	h	ocho

Esta consulta también puede escribirse:

```
SELECT *
FROM tabla1
LEFT OUTER JOIN tabla2
ON tabla1.codigo = tabla2.codigo;
```

Basándonos nuevamente en los diagramas de Venn, podemos expresar el resultado del LEFT [OUTER] JOIN del siguiente modo:



3. RIGHT [OUTER] JOIN

Este JOIN devuelve todos los registros de la tabla de la derecha y los registros que coinciden de la tabla de la izquierda:

```
SELECT *
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.codigo = tabla2.codigo;
```

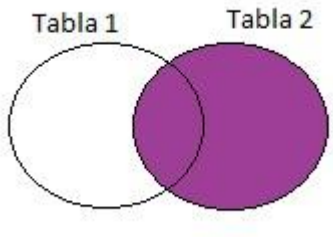
Obteniendo como resultado de la combinación de la tabla1 y la tabla2, los siguientes registros:

3	c	tres
5		cinco
8	h	ocho

Esta consulta también puede escribirse:

```
SELECT *
FROM tabla1
RIGHT OUTER JOIN tabla2
ON tabla1.codigo = tabla1.codigo;
```

Si seguimos con la analogía de los diagramas de Venn, podemos expresar el resultado del RIGHT [OUTER] JOIN del siguiente modo:



4. CROSS JOIN

Combina cada registro de la tabla de la izquierda con cada registro de la tabla de la derecha, sin hacer coincidir un campo en particular:

```
SELECT * FROM tabla1 CROSS JOIN tabla2;
```

Obteniendo como resultado de la combinación de la tabla1 y la tabla2, los siguientes registros:

1	a	3	tres

1	a	5	cinco
1	a	8	ocho
3	c	3	tres
3	c	5	cinco
3	c	8	ocho
8	h	3	tres
8	h	5	cinco
8	h	8	ocho

Este tipo JOIN brinda la posibilidad de cruzar todos los registros con todos (producto cartesiano) y resulta imposible de dibujar con un diagrama de Venn. Se puede observar que representa un JOIN muy poco eficiente de ejecutar en tablas grandes.

Combinación de Consultas - UNION

Para comparar los resultados de varias consultas y combinarlos en un nuevo resultado basado en esa comparación existe (entre otros) el operador UNION.

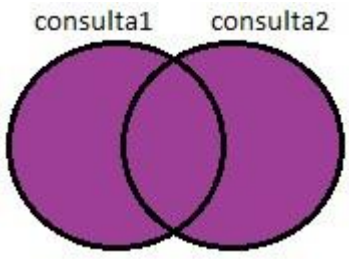
Como vamos a comparar varias consultas, es necesario que en cada resultado exista la misma cantidad de campos y que los campos a comparar tengan tipos de datos compatibles. Cabe aclarar que no es necesario que tengan el mismo nombre.

Debemos diferenciar por un lado, la comparación de los resultados de varias consultas y su combinación y por otro, la combinación de datos de varias tablas en una consulta: para esto último usamos JOIN estableciendo una relación por uno o varios campos con otras tablas.
Las dos consultas se pueden hacer sobre la misma tabla, pero podrían ser cualquier consulta siempre y cuando se respete que la salida contenga la misma cantidad de campos y que los tipos de datos de los campos sean compatibles para la comparación, en general esto implica campos numéricos con campos numéricos y campos de texto con campos de texto.
La sintaxis para combinar dos consultas mediante la cláusula UNION es la siguiente (Los corchetes ([]) indican que el elemento es opcional):

consulta1 UNION [ALL] consulta2

Esta sintaxis, agrega el resultado de la consulta2 al resultado de la consulta1. Los registros duplicados se eliminan. Para conservar los registros duplicados se usa UNION ALL en lugar de UNION.

Gráficamente, podemos ver que el resultado está formado por todos los registros de la primera y de la segunda consulta:



La primera consulta que interviene se escribe sin el punto y coma (;) al final ya que la consulta completa está formada por las consultas que intervienen y el operador UNION finalizando después de la segunda consulta. Esto es así para todas las combinaciones de consultas.