# Introducción a SQL JOINs y UNIONs

Por: Hisakazu Ishibashi – UNI – 2007

# SQL Joins

Antes de empezar definiremos 3 tablas para usarlas como ejemplo en el tutorial.

```
CREATE TABLE a (
    a1 int,
    a2 varchar(2)
)
go
```

```
+----+------+
| a1 | a2   |
+----+------+
| 10 | u    |
| 20 | v    |
| 30 | w    |
| 40 | x    |
| 50 | y    |
| 60 | z    |
+----+------+
```

```
CREATE TABLE b (
    b1 int,
    b2 varchar(2)
)
go
```

```
+----+------+
| b1 | b2   |
+----+------+
| 10 | p    |
| 20 | q    |
+----+------+
```

# SQL Joins

Por último, pero no menos.

```
CREATE TABLE c (
    c1 int,
    c2 varchar(2)
)
go
```

```
+----+------+
| c1  | c2   |
+----+------+
| 90  |  m   |
|100 |  n    |
|110 |  o    |
+----+------+
```

Es probable que las técnicas descriptas a continuación en este artículo no funcionen en otros RDBMS; si ese el caso UD debería consultar la documentación proveída con el sistema para obtener la sintaxis correcta.

# SQL Joins

Estas tablas fueron creadas de la forma más sencilla de manera a que se tenga la menor dificultad posible para entender los ejemplos.

En la siguiente etapa, después de explicar lo básico sobre los **JOINS**, se reemplazarán estas tablas por otras más semejantes a las usadas en realidad, para poder comprender algunas de las aplicaciones más comunes de **JOINS.**

# SQL Joins – inner join

En clases anteriores, al realizar consultas de varias tablas hemos ejecutados *joins* de manera implícita, lo cual se demostrará a continuación.

**SELECT * FROM a,b;**

El resultado es →

```
+----+------+----+------+
| a1 | a2   | b1 | b2   |
+----+------+----+------+
| 10 | u    | 10 | p    |
| 20 | v    | 10 | p    |
| 30 | w    | 10 | p    |
| 40 | x    | 10 | p    |
| 50 | y    | 10 | p    |
| 60 | z    | 10 | p    |
| 10 | u    | 20 | q    |
| 20 | v    | 20 | q    |
| 30 | w    | 20 | q    |
| 40 | x    | 20 | q    |
| 50 | y    | 20 | q    |
| 60 | z    | 20 | q    |
+----+------+----+------+
12 rows in set (0.00 sec)
```

# SQL Joins – inner join

La consulta anterior es nuestro primer ***join,*** el cual es conocido como ***cross join*** y el numero de filas en el resultado es igual al producto del numero de filas de cada una de las tablas usadas para el ***join***. Se puede verificar esto al notar que la tabla "**a**" tiene 6 filas y la tabla "**b**" tiene 2 filas, por lo tanto el resultado tiene 12 filas.

Existen 2 tipos de join – *el "**inner join**" y el "**outer-join**"*. El **inner join** es el más común – *el ejemplo anterior es un **inner join*** – y también el más simétrico, ya que requiere una igualdad (match) en cada una de las tabla que forma parte del join. Las filas que no son igualadas son descartadas del resultado final.

Como podrán imaginarse, un ***cross join*** puede tener serias implicaciones en la performance de un servidor de BD. En este ejemplo solo tenemos como resultado 12 filas, pero no es lo mismo si las tablas tuvieran 100 o más filas.

# SQL Joins – inner join

Ya que un *cross join* puede producir un resultado enorme, la forma que usamos para filtrar los resultados es al añadir un ***where*** a nuestra consulta.

**SELECT * FROM a,b WHERE a1>30;**

Una variante del *cross-join* es el ***equi-join****,* donde ciertos campos en las tablas asociadas son igualados unos a otros. En este caso el resultado final solo incluirá las filas de las tablas asociadas en las cuales se tiene una coincidencia para los campos especificados. Esto ya es familiar para nosotros, ya que es la característica de las consultas que hemos aprendido hasta ahora. Ejemplo:

**SELECT * FROM a,b WHERE a1 = b1;**

# SQL Joins – inner join

La forma explícita de:

**SELECT * FROM a,b WHERE a1 = b1;**

es:

**SELECT * FROM a inner join b on a1 = b1;**

# SQL Joins – inner join

Natural join

Un **natural join** es una especialización de los equi-join. El predicado del **join** está dado implícitamente, comparando las columnas que tienen el mismo nombre en ambas tablas asociadas. El resultado contiene solo una columna para cada par de columnas que tienen el mismo nombre.

Para este ejemplo suponemos que tenemos una tabla "d", con los campos a1 y n, entonces:

**SELECT * FROM a NATURAL JOIN d;** nos daría el siguiente resultado:

```
+----+------+-----+
| a1 | a2   | n   |
+----+------+-----+
| 90 |  m   | n1 |
|100 |  n   | n2 |
|110 |  o   | n3 |
+----+------+-----+
```

El uso de NATURAL JOIN es ambiguo, y podria ocasionar problemas si cambia el schema de la BD. Por ejemplo, si se cambia el nombre de una columna, cambia la semántica del NATURAL JOIN. Es por eso que usualmente es mejor declarar de manera explícita la condición de búsqueda del Join.

# SQL Joins – outer join

Como se ha visto en diapositivas anteriores, los inner join son simétricos; osea que para que una fila sea incluida en el resultado, la fila debe tener una coincidencia en todas las tablas asociadas. Outer join, en cambio, son asimétricos, todas las filas de *un lado* de la asociación son incluidas en el resultado final, sin importar si tuvieron coincidencias en el *otro lado.*

Esto puede parecer un poco complicado pero con un ejemplo se lo puede ver más claramente. Consideremos la siguiente consulta

**SELECT * FROM a LEFT [OUTER] JOIN b ON a1=b1;**

El resultado es: →

```
+----+------+--------+--------+
| a1 | a2   | b1     | b2     |
+----+------+--------+--------+
| 10 | u    | 10     | p      |
| 20 | v    | 20     | q      |
| 30 | w    | NULL | NULL |
| 40 | x    | NULL | NULL |
| 50 | y    | NULL | NULL |
| 60 | z    | NULL | NULL |
+----+------+--------+--------+
6 rows in set (0.06 sec)
```

# SQL Joins – outer join

De la misma forma que LEFT OUTER JOIN, también tenemos RIGHT OUTER JOIN, que hace exactamente lo mismo que LEFT JOIN pero al revés.

**SELECT * FROM a RIGHT JOIN c ON a1=c1;**

# SQL Joins – Ejercicios

Después de la teoría es necesario un poco de práctica para repasar la teoría. Considerar las siguientes tablas que tienen datos para un pequeña biblioteca.

```
+----+-------------------------------------------+
 | id  | title                                    |
+----+-------------------------------------------+
 | 1   | Mystic River                             |
 | 2   | Catch-22                                 |
 | 3   | Harry Potter and the Goblet of Fire      |
 | 4   | The Last Detective                       |
 | 5   | Murder on the Orient Express             |
+----+-------------------------------------------+
```

```
+-----+-------+
 |  id  | name|
+-----+-------+
 |100 | Joe   |
 | 101| John |
 | 103| Dave |
 | 109| Harry|
 | 110| Mark |
 | 113| Jack  |
 +-----+-------+
```

La 1era tabla se llama “books” y contiene la lista completa de libros. Cada libro tiene un único ID. La 2da tabla se llama “members” y contiene la lista de todos los socios.

# SQL Joins – Ejercicios

Una tercera tabla contiene el registro de los préstamos (loans) de libros

| member_id | book_id |
|---|---|
| 101 | 1 |
| 103 | 3 |
| 105 | 4 |

Realizar las siguientes consultas:

1) Lista de todos los miembros
2) Lista de todos los libros en el catálogo
3) Ver que libro tiene actualmente el miembro Dave
4) Ver que miembros no tienen prestado ningún libro
5) Lista de todos los libros en stock
6) Lista de todos los titulos prestados actualmente con el nombre de los socios que lo prestaron

# SQL Joins – self joins

Además de los inner y outer joins, SQL también permite un tercer tipo de join, conocido como *self join.* Típicamente este tipo de join es usado para extraer datos de una tabla cuyos registros están relacionados unos a otros.

Para entender mejor considerar la tabla en la siguiente diapositiva, la cual guarda datos para representar una jerarquía de menús.

# SQL Joins – self joins

| id | label | parent |
|---|---|---|
| 1 | Services | 0 |
| 2 | Company | 0 |
| 3 | Media Center | 0 |
| 4 | Your Account | 0 |
| 5 | Community | 0 |
| 6 | For Content Publishers | 1 |
| 7 | For Small Businesses | 1 |
| 8 | Background | 2 |
| 9 | Clients | 2 |
| 10 | Addresses | 2 |
| 11 | Jobs | 2 |
| 12 | News | 2 |
| 13 | Press Releases | 3 |
| 14 | Media Kit | 3 |
| 15 | Log In | 4 |
| 16 | Columns | 5 |
| 17 | Colophon | 16 |
| 18 | Cut | 16 |
| 19 | Boombox | 16 |

# SQL Joins – self joins

Supongamos que necesitamos listar cada *nodo del árbol* con el nombre de su nodo padre. Es aquí en donde usamos *self join* ya que todos los datos que necesitamos relacionar se encuentran en la misma tabla.

La solución para el problema es entonces:

```
SELECT a.label AS parent_label, b.label AS child_label FROM menu AS a,
menu AS b WHERE a.id = b.parent;
```

Como se puede observar utilizamos los alias para crear una copia virtual de la misma tabla y relacionar sus atributos.

# SQL Joins – full outer join

Por último tenemos los *full outer join* los cuales combinan el resultado de ambos el left y el right outer join.  El resultado contendrá todos los registros de ambas tablas, y llenará con NULLs las coincidencias no encontradas en ambos *lados.*

**SELECT * FROM a FULL OUTER JOIN b ON a1 = b1**

# UNION y UNION ALL

El operador UNION combina el resultado de 2 consultas en un solo resultado. Las 2 tablas deben coincidir en los nombre de campos y en los tipos de datos para que puedan ser unidas. Cualquier registro duplicado es eliminado, si es que no se usa UNION ALL.

UNION puede ser muy útil en aplicaciones *data warehouse(ver enlaces recomendados)* en donde las tablas no están perfectamente normalizadas, la mayoría de las veces por cuestiones de performance.

Ventas 2005

| **person** | **amount** |
|---|---|
| Joe | 1000 |
| Alex | 2000 |
| Bob | 5000 |

Ventas 2006

| **person** | **amount** |
|---|---|
| Joe | 2000 |
| Alex | 2000 |
| Zach | 35000 |

# UNION y UNION ALL

Como ejemplo tenemos las 2 tablas a continuación.

Ventas2005

| person | amount |
|---|---|
| Joe | 1000 |
| Alex | 2000 |
| Bob | 5000 |

Ventas2006

| person | amount |
|---|---|
| Joe | 2000 |
| Alex | 2000 |
| Zach | 35000 |

```
SELECT *
FROM
Ventas2005
UNION
SELECT *
FROM
Ventas2006
```

| person | amount |
|---|---|
| Joe | 1000 |
| Alex | 2000 |
| Bob | 5000 |
| Joe | 2000 |
| Zach | 35000 |

```
SELECT *
FROM
Ventas2005
UNION ALL
SELECT *
FROM
Ventas2006
```

| person | amount |
|---|---|
| Joe | 1000 |
| Joe | 2000 |
| Alex | 2000 |
| Alex | 2000 |
| Bob | 5000 |
| Zach | 35000 |

# Bibliografía y Enlaces recomendados


- http://www.devshed.com/c/a/MySQL/Understanding-SQL-Joins/
- http://en.wikipedia.org/wiki/SQL#SQL_keywords
- http://en.wikipedia.org/wiki/Join_%28SQL%29
- http://en.wikipedia.org/wiki/Data_warehouse