

PRÁCTICA: 4. LENGUAJE DE CONSULTAS (SELECT) - II

<u>1. SOBRE LA PRÁCTICA.....</u>	<u>2</u>
1.1. OBJETIVOS	2
1.2. MATERIAL	2
1.3. CONTENIDOS	2
1.4. BIBLIOGRAFÍA:	2
<u>2. OPERADORES PARA COLUMNAS DE TIPO NO NUMÉRICO</u>	<u>3</u>
2.1. DATOS DE TIPO CARÁCTER.....	3
2.2. DATOS DE TIPO FECHA.....	4
<u>3. COMPOSICIÓN DE TABLAS: CONSULTAS A VARIAS TABLAS.</u>	<u>5</u>
3.1. BASE DE DATOS DE EJEMPLO.	6
3.2. COMPOSICIÓN Y PRODUCTO CARTESIANO	8
3.3. EQUICOMPOSICIÓN.....	9
3.4. THETACOMPOSICIÓN.....	10
3.5. COMPOSICIÓN MÚLTIPLE.....	11
3.6. MÁS EJEMPLOS ILUSTRATIVOS.	12
3.7. OTROS TIPOS DE COMPOSICIONES	13
3.7.1. INNER JOIN VS OUTER JOIN	13
3.7.2. CROSS JOIN, NATURAL JOIN Y CLÁUSULAS USING Y ON.	16
<u>4. OPERADOR UNION Y UNION ALL</u>	<u>17</u>
4.1. EJEMPLOS.....	18
<u>5. OPERADOR INTERSECT.....</u>	<u>18</u>
5.1. EJEMPLOS.....	18
<u>6. OPERADOR MINUS.....</u>	<u>19</u>
6.1. EJEMPLOS.....	19
<u>7. EJERCICIOS</u>	<u>20</u>

1. Sobre la práctica

1.1. Objetivos

Conocer el uso básico de la sentencia SELECT de SQL para realizar consultas que afecten a varias tablas de la BD.

En esta práctica se estudiarán las funciones asociadas a datos de tipo carácter (char, varchar, varchar(2)) y a datos de tipo fecha (date).

También se estudiará la combinación de tablas para hacer consultas multitaslas

Finalmente se estudiarán operadores para unir varias consultas: union, intersect y minus.

1.2. Material

ORACLE9 versión para WINDOWS XP

1.3. Contenidos

- Operadores de tipo no numérico
 - De tipo Carácter
 - De tipo Fecha
- Composición de Tablas: Consultas a varias tablas
 - Composición y Producto Cartesiano
 - Equicomposición
 - *Inner Join / Natural Join*
 - *Join using / join on*
 - *Outer Join: Left/ Right / Full Outer Join*
- Operador *Union* y *Union All*
- Operador *Insersect*
- Operador *Minus*
- Ejercicios

1.4. Bibliografía:

GUÍA DE SQL
JAMES R. GROFF, PAUL N. WEINBERG
MCGRAW-HILL, 1998

ORACLE 9I: GUIA DE APRENDIZAJE
ABBEY, MICHAEL Y COREY, MIKE Y ABRAMSON, IAN
MCGRAW-HILL / INTERAMERICANA DE ESPAÑA, S.A.

2. Operadores para columnas de tipo no numérico

Aunque, en general, los operadores que más se utilizan son los que se aplican a columnas definidas de tipo numérico, en una tabla también pueden existir columnas de otros tipos de datos que tendrán sus operadores específicos, aplicables a la sentencia SELECT.

2.1. Datos de tipo carácter

Como ya se ha comentado existen tipos de datos carácter utilizables en la creación de una tabla: char, varchar, y varchar(2), que se utilizan para representar todas las letras, los números y los caracteres especiales.

Asociados a estos tipos de datos existen toda una serie de funciones que se les pueden aplicar.

Suponiendo que se tiene una tabla usuarios, con el siguiente contenido:

Nombre	DNI
pepe lópez	123
pepe garcía	456
MANUEL AMO	789

Entre otras, las funciones aplicables y los resultados correspondientes serían:

initcap(char) → pone en mayúscula el primer carácter de cada cadena de caracteres.

Ejemplo. *select initcap(nombre) from usuarios;*

Salida:
Pepe López
Pepe García
Manuel Amo

lower(char) → pone en minúscula la cadena de caracteres completa

Ejemplo. *select lower(nombre) from usuarios;*

Salida:
pepe lópez
pepe garcía
manuel amo

replace(char, str1, str2) → En la cadena de caracteres char, cada ocurrencia de str1 se reemplaza por str2

Ejemplo: *select replace(nombre, 'pe', 'lu') from usuarios;*

Salida:
lulu lóluz
lulu garcía
MANUEL AMO

substr(char,m,n) → Extrae n caracteres de la cadena de caracteres char, a partir de la posición m.

Ejemplo: *select substr(nombre, 2, 5) from usuarios;*
Salida:
epe l
epe g
ANUEL

length(char) → Longitud de char

Ejemplo: *select length(nombre) from usuarios;*
Salida:
10
11
10

2.2. Datos de tipo fecha

Los datos de tipo fecha en realidad contienen la fecha y la hora. Esto es especialmente importante cuando se comparen dos fechas. El formato por defecto es DD-Mes-AA aunque es posible cambiarlo.

Entre las funciones específicas para fechas tenemos:

sysdate → Fecha y hora actual del sistema.

last_day → último día del mes actual

add_months(d,n) → suma o resta n meses a partir de la fecha d

Ejemplo: *select add_months('12-MAR-99',2) from usuarios;*

months_between(f,s) → diferencia en meses entre la fecha f y la fecha s

next_day(d,day) → Fecha del día especificado (lunes, martes, ...en inglés) de la semana siguiente a d.

También es posible definir el formato deseado para las fechas:

Y → devuelve el último dígito del año

YY → devuelve los dos últimos dígitos del año

YYY → devuelve los tres últimos dígitos del año

Ejemplo: *select to_char (sysdate, 'YY') from usuarios*

SYEAR, YEAR → año, utilizando signo para las fechas a.C.

Q → trimestre del año

MM → mes

RM → mes en números romanos

Month → Nombre del mes (9 caracteres)

WW → semana del año

W → semana del mes

DDD → día del año

DD → día del mes

D → día de la semana

DY → Abreviatura del nombre del día

HH o **HH12** → Hora del día

HH24 → hora del día utilizando las 24

MI → minutos

SS → segundos

Por último hay que resaltar el hecho de que es posible sumar a las fechas valores de la forma:

select (fechaventa+10) from ventas;

indica a Oracle que sume 10 días al valor contenido en la columna fechaventa .Esta suma tiene en cuenta posibles desbordamiento, devolviendo el mes siguiente del actual en caso de resultar necesario.

3. Composición de tablas: consultas a varias tablas.

La mayor parte de las ocasiones, todos los datos no estarán en la misma tabla, sino que por diseño de la base de datos, habrán quedado repartidos en diferentes tablas.

En esas ocasiones, para hacer las consultas correspondientes es necesario combinar todas las tablas en las que están los datos. A continuación se presenta una base de datos de ejemplo que va a servir para ilustrar todas las posibles combinaciones de tablas en diferentes consultas. Las tablas mostradas son el resultado de las correspondientes consultas SQL realizadas en ISQLPLUS. Debajo de cada tabla, se anexa en cursiva la consulta realizada para que el alumno se vaya familiarizando con la sintaxis.

3.1. Base de datos de Ejemplo.

Sean las siguientes tablas:

TABLE_NAME
ALUMNO
DEPARTAMENTO
PROFESOR

(select table_name from user_tables)

Siendo:

Departamento:

Nombre	¿Nulo?	Tipo
ID	NOT NULL	CHAR(3)
NOMBRE		VARCHAR2(30)
NRO_PROFESORES		NUMBER(38)

(desc departamento)

Profesor:

Nombre	¿Nulo?	Tipo
NRO_REG	NOT NULL	NUMBER(38)
NOMBRE		VARCHAR2(10)
APELLIDOS		VARCHAR2(20)
DEPARTAMENTO		CHAR(3)

(desc Profesor)

Alumno:

Nombre	¿Nulo?	Tipo
NRO_EXP	NOT NULL	NUMBER(38)
NOMBRE		VARCHAR2(10)
APELLIDOS		VARCHAR2(20)
CURSO		NUMBER(38)
ESPECIALIDAD		VARCHAR2(10)
TUTOR		NUMBER(38)

(Desc Alumno)

La columna etiquetada como C indica el tipo de restricción: C es de tipo *check*, P de tipo *Primary Key* y R es de tipo *Foreign Key* (la columna R_CONSTRAINT_NAME indica la restricción de tipo PK a la que referencia cada foreign key especificada). Estas observaciones deberían ser tenida en cuenta por los alumnos para reproducir el correspondiente esquema.

TABLE_NAME	CONSTRAINT_NAME	C	SEARCH_CONDITION	R_CONSTRAINT_NAME
ALUMNO	NRO_EXP_NOT_NULL	C	"NRO_EXP" IS NOT NULL	
ALUMNO	PK_ALUMNO	P		
ALUMNO	FK_ALUMNO_A_PROFESOR	R		PK_PROFESOR
DEPARTAMENTO	ID_NOT_NULL	C	"ID" IS NOT NULL	
DEPARTAMENTO	PK_DEPARTAMENTO	P		
PROFESOR	NRO_RE_NOT_NULL	C	"NRO_REG" IS NOT NULL	
PROFESOR	PK_PROFESOR	P		
PROFESOR	FK_DEPARTAMENTO_A_PROFESOR	R		PK_DEPARTAMENTO

(select table_name, Constraint_name, constraint_type, search_condition, r_constraint_name from user_constraints;)

Donde se insertan los siguientes datos:

NRO_REG	NOMBRE	APELLIDOS	DEP
11	Maria	Martínez	1
12	Pepe	Martínez	1

```
insert into profesor values(11, 'maria', 'martínez', 1);
insert into profesor values(12, 'pepe', 'martínez', 1);
select * from profesor;
```

NRO_EXP	NOMBRE	APELLIDOS	CURSO	ESPECIALIDAD	TUTOR
1	Luis	Fernández	1	LSI	11
2	Maria	Ruiz	1	LSI	11
3	Carmen	López	1	LSI	11
4	Elena	Martin	1	LSI	12

```
insert into alumno values (1, 'luis', 'fernández', 1, 'lsi', 11);
insert into alumno values (2, 'maria', 'ruiz', 1, 'lsi', 11);
insert into alumno values (3, 'carmen', 'lópez', 1, 'lsi', 11);
insert into alumno values (4, 'elena', 'martin', 1, 'lsi', 12);
select * from alumno;
```

ID	NOMBRE	NRO_PROFESORES
1	tsi	13

```
insert into departamento values(1, 'tsi', 13)
select * from departamento;
```

3.2. Composición y Producto Cartesiano

La composición (join) de tablas se produce cuando se combinan datos de dos o más tablas. Es un caso particular de la operación del álgebra relacional llamada **producto cartesiano**. Para obtener un producto cartesiano se debe poner en la cláusula *from* el nombre de las tablas que se quieran combinar sin poner ninguna condición (o bien que la que se ponga sea inválida) de igualdad en la cláusula *where*.

Por ejemplo, la siguiente tabla muestra el producto cartesiano de alumno y departamento.

NRO_EX P	NOMBRE	APELLIDOS	CURSO	ESPECIALIDAD	TUTOR	NRO_REG	NOMBRE	APELLIDOS	DEP
1	Luis	Fernández	1	LSI	11	11	Maria	Martínez	1
2	Maria	Ruiz	1	LSI	11	11	Maria	Martínez	1
3	Carmen	López	1	LSI	11	11	Maria	Martínez	1
4	Elena	Martin	1	LSI	12	11	Maria	Martínez	1
1	Luis	Fernández	1	LSI	11	12	Pepe	Martínez	1
2	Maria	Ruiz	1	LSI	11	12	Pepe	Martínez	1
3	Carmen	López	1	LSI	11	12	Pepe	Martínez	1
4	Elena	Martin	1	LSI	12	12	Pepe	Martínez	1

(Select * from alumno, profesor)

El número total de filas que se obtendrían en la consulta sería el producto del número de filas que tiene cada una de ellas. Como puede verse, esta consulta origina 8 filas como resultado del producto de las 4 que tiene la tabla alumno por las dos que tiene la tabla Profesor.

Si se quiere limitar el producto cartesiano es preciso incorporar condiciones a la cláusula *where*. Para poder realizar la composición **deben existir columnas comunes a las tablas**, de forma que al componer dos tablas A y B con las columnas comunes X e Y se satisface una condición del tipo $A.X=B.Y$ o similar.

3.3. Equicomposición.

Es el caso más habitual, en el cual la condición es una igualdad entre dos columnas que son la clave primaria de una de las tablas y la clave ajena de la otra tabla que apunta a la mencionada clave primaria. A este tipo de consulta se le llama Equicomposición, Equijoin o Inner Join

En el caso de la consulta entre el profesor y el alumno, podría ser *Profesor.Nro_reg* la clave primaria de Profesor y *Alumno.Tutor* la clave ajena de alumno que apunta a *Profesor.Nro_reg*.

Así la consulta

```
select alumno.Nombre as "NomAlumno", alumno.Apellidos as "ApeAlumno",
Profesor.Nombre as "NomProfesor", Profesor.Apellidos as "ApeProfesor"
from alumno , profesor
where alumno.tutor = profesor.nro_reg
```

daría el siguiente resultado:

NomAlumno	ApeAlumno	NomProfeso	ApeProfesor
Luis	Fernández	Maria	Martínez
Maria	Ruiz	Maria	Martínez
Carmen	López	Maria	Martínez
Elena	Martin	Pepe	Martínez

Es importante tener en cuenta la interpretación de la pregunta que se hace para elaborar la consulta. En este caso se podría decir: “*dime el nombre y los apellidos de los tutores de cada uno de los alumnos*”. Para mostrar el resultado obsérvese que se han utilizado alias para las columnas con el objetivo de dejar claro a quién pertenece cada uno de los datos.

Mediante la introducción de más condiciones convenientemente escritas usando los operadores lógicos, es posible seleccionar como si fuese una consulta de una única tabla alguna de las tuplas.

Por ejemplo, si quisiésemos saber quién es el tutor de “Elena Martín”, habría que añadir a la consulta anterior “*and alumno.Nombre = 'Elena' and alumno.apellidos='Martin';*” quedando la consulta:

```
select alumno.Nombre as "NomAlumno", alumno.Apellidos as "ApeAlumno",
Profesor.Nombre as "NomProfesor", Profesor.Apellidos as "ApeProfesor"
from alumno , profesor
where alumno.tutor = profesor.nro_reg
and alumno.Nombre = 'Elena' and alumno.apellidos='Martin';
```

NomAlumno	ApeAlumno	NomProfeso	ApeProfesor
Elena	Martin	Pepe	Martínez

A partir de Oracle 9i se introduce lo que se llama *Natural Join* que es un tipo de equicomposición que automáticamente usa para hacer en una *inner join* las columnas en cada tabla que tienen el mismo nombre, eliminando la necesidad de hacer la comparación de igualdad y permitiendo así hacer consultas más sencillas. En nuestro ejemplo podría hacerse si la columna *tutor* se llamase *nro_reg* como en profesor.

3.4. Thetacomposición.

Similar a la anterior, pero el operador relacional que realiza la comparación es distinto de la igualdad:

$$A.X \theta B.Y \quad | \quad \theta \in \{>, <, \diamond, >=, <=\}$$

Con esto se podrían obtener todos los valores que no cumplen una condición.

Por ejemplo, mientras que por la equicomposición se puede saber quién es el tutor de “*Elena Martín*”, por la thetacomposición se podría llegar a saber qué otros profesores no son sus tutores. Para ello en la cláusula *where* la comparación entre clave primaria / clave ajena se haría con ‘!=’ (thetacomposición) en lugar de con ‘=’ (equicomposición). Así la consulta sería.

```
select alumno.Nombre as "NomAlumno", alumno.Apellidos as "ApeAlumno",
Profesor.Nombre as "NomProfesor", Profesor.Apellidos as "ApeProfesor"
from alumno , profesor
where alumno.tutor != profesor.nro_reg
and alumno.Nombre = 'Elena' and alumno.apellidos='Martin';
```

Siendo el resultado que María Martínez no es tutora de Elena Martín (observéese que en la tabla profesores se tienen datos sólo de dos profesores)

NomAlumno	ApeAlumno	NomProfeso	ApeProfesor
Elena	Martin	María	Martínez

3.5. Composición múltiple.

Es la generalización a más de dos tablas. Por ejemplo, para tres tablas la condición será del tipo:

$(A.X = B.Y1) \text{ AND } (B.Y2 = C.Z)$

En el caso de que se quisiese hacer una composición (bien equi bien theta) de tres columnas, habría que relacionar dos a dos las tablas a través de los pares (clave primaria, clave ajena) correspondientes. Por ejemplo si se quisiera saber el nombre del departamento al que está adscrito el tutor de la alumna Elena Martín, habría que combinar la información de la tabla alumno, ya que allí es donde están los datos de la alumna, y la de profesor, ya que ahí están los datos del profesor. La combinación se haría a través del campo Alumno.Tutor/ Profesor.nro_reg. Hasta aquí podríamos llegar a saber cuál es el identificador del departamento (Profesor.Departamento) al que pertenece el tutor, pero no el nombre de dicho Departamento. Este dato está en la tabla Departamento. Por tanto habría que combinar las tablas Profesor y Departamento a través de Profesor.Departamento (clave ajena en Profesor) / Departamento.Id (clave primaria en Departamento). La combinación se haría con el operador lógico **and**. Y si además hubiera más condiciones como es este caso, pues se combinaría con los operadores lógicos que semánticamente mejor representen lo que se pretende.

La consulta en este caso quedaría como sigue:

```
select Departamento.Nombre
from alumno, profesor, departamento
where (alumno.tutor = profesor.nro_reg and profesor.departamento =
departamento.id)
and alumno.Nombre = 'Elena' and alumno.apellidos='Martin';
```

Y el resultado sería:

NOMBRE
tsi

En general podría decirse que para combinar n tablas se necesita n-1 relaciones a través de pares clave primaria / claves ajena.

3.6. Más Ejemplos ilustrativos.

Otro ejemplo de **equicomposición** (Nombre y apellidos de alumnos tutorizados por profesores del departamento1) sería:

```
SELECT alumno.nombre, alumno.apellidos
FROM alumno, profesor
WHERE tutor=profesor.nro_reg
AND departamento =1;
```

Cuya salida sería:

NOMBRE	APELLIDOS
Luis	Fernández
Maria	Ruiz
Carmen	López
Elena	Martin

De **thetacomposición** (Nombre de alumnos del mismo curso y especialidad tutorizados por diferentes profesores) sería (además es un ejemplo de autocomposición):

```
SELECT a.nombre, b.nombre
FROM alumno a, alumno b
WHERE a.tutor<>b.tutor
AND a.curso=b.curso
AND a.especialidad=b.especialidad
AND a.nro_exp<b.nro_exp;
```

Cuya salida sería:

NOMBRE	NOMBRE
Luis	Elena
Maria	Elena
Carmen	Elena

Y de composición múltiple (Nombre y apellidos de los alumnos autorizados por profesores del departamento TSI):

```
SELECT alumno.nombre, alumno.apellidos
FROM profesor, alumno, departamento
WHERE tutor=nro_reg
AND departamento=id
AND departamento.nombre = 'tsi';
```

Cuya salida sería:

NOMBRE	APELLIDOS
Luis	Fernández
Maria	Ruiz
Carmen	López
Elena	Martin

3.7. Otros tipos de Composiciones

3.7.1. Inner Join vs Outer Join

Todas las consultas descritas en los subapartados anteriores muestran tuplas que satisfacen el resultado de la condición puesta para hacer la composición. A este tipo de consultas se le llama *Inner Join*.

En ocasiones puede ser necesario mostrar también aquellas tuplas que no satisfacen el resultado de la condición. En ese caso se habla de *outer join* (no debe confundirse con la thetacomposición).

Para representarlo considérese que se introduce en la tabla profesores una nueva tupla:

```
insert into profesor (nro_reg, nombre, apellidos) values (13, 'Antonio','López');
```

Obsérvese que no se ha proporcionado un valor para el atributo *Departamento*

*Select * from profesores order by nro_reg* devolvería esta tabla:

NRO_REG	NOMBRE	APELLIDOS	DEP
11	Maria	Martínez	1
12	Pepe	Martínez	1
13	Antonio	López	

Una consulta de producto cartesiano sobre las tablas *departamento* y *profesor* devolvería estos 3 (3 de profesores por 1 de departamentos) valores:

ID	NOMBRE	NRO_PROFESORES	NRO_REG	NOMBRE	APELLIDOS	DEP
1	tsi	13	13	Antonio	López	
1	tsi	13	11	Maria	Martínez	1
1	tsi	13	12	Pepe	Martínez	1

Si se hiciese una consulta como las vistas hasta ahora (por ejemplo muestre todos los profesores que tiene un departamento) quedaría (equicomposición):

```
select * from departamento, profesor
where departamento.id = profesor.departamento
```

ID	NOMBRE	NRO_PROFESORES	NRO_REG	NOMBRE	APELLIDOS	DEP
1	tsi	13	11	Maria	Martínez	1
1	tsi	13	12	Pepe	Martínez	1

O todos los departamentos que no tienen profesores (thetacomposición):

```
select * from departamento, profesor
where departamento.id != profesor.departamento
```

que en nuestro caso daría una respuesta vacía porque en la base de datos sólo hay información de un departamento 'tsi' y tiene los dos profesores mostrados anteriormente.

En cambio si quisiésemos saber qué profesores no tienen departamento podríamos hacerlo de dos formas:

1. Preguntando directamente los que tienen el campo a null, lo que nos daría poca información para comparar:

```
select * from profesor where departamento is null;
```

NRO_REG	NOMBRE	APELLIDOS	DEP
13	Antonio	López	

O bien

2. viendo todos los profesores y todos los departamentos que hay, mediante una composición del tipo *outer join* :

```
select * from departamento, profesor
where departamento.id (+) = profesor.departamento
```

ID	NOMBRE	NRO_PROFESORES	NRO_REG	NOMBRE	APELLIDOS	DEP
			13	Antonio	López	
1	tsi	13	11	Maria	Martínez	1
1	tsi	13	12	Pepe	Martínez	1

Para hacer un *outer join* hay que colocar el operador '+' encerrado entre paréntesis del lado que no se tengan datos. Tiene el efecto de crear una o más filas de la tabla que no tiene información para que pueda ser completada mediante el *update* correspondiente. Este operador sólo puede aparecer del lado de la expresión de la que no se tiene información (en nuestro caso no tenemos información del departamento)

Además en Oracle 9i esta sentencia sería equivalente a:

```
select * from departamento right outer join profesor on departamento.id = profesor.departamento
```

o bien

```
select * from profesor left outer join departamento on departamento.id = profesor.departamento
```

Si se hubiera puesto la consulta con el operador (+) del otro lado se hubiera producido este resultado:

```
select * from departamento, profesor
where departamento.id = profesor.departamento (+)
```

```
select * from profesor right outer join departamento on departamento.id = profesor.departamento
```

```
select * from departamento left outer join profesor on departamento.id = profesor.departamento
```

ID	NOMBRE	NRO_PROFESORES	NRO_REG	NOMBRE	APELLIDOS	DEP
1	tsi	13	11	Maria	Martínez	1
1	tsi	13	12	Pepe	Martínez	1

Una limitación de este operador es que no se puede usar con el operador IN o cuando se utiliza el operador OR.

En cuanto a la sintaxis de *outer join* se podría haber puesto *full outer join* para que comprobara tanto la *right outer join* como la *left outer join*.

3.7.2. Cross Join, Natural Join y Cláusulas Using y On.

Cross Join sirve para obtener el producto cartesiano de dos tablas. Este operador estaba inicialmente definido en SQL 1999. Es equivalente a no poner cláusula *where*.

Así *select * from profesor cross join alumno* devolverá el mismo resultado que *select * from profesor, alumno*

Para poder hacer un *natural join* es preciso que las dos tablas que se combinan tengan el mismo nombre y el mismo tipo de datos para el atributo por el que se hace la composición. En caso de no tenerlo hay que recurrir a poner una cláusula *where Tabla1.AtributoPK = Tabla2.AtributoFK*, como se ha visto en los ejemplos anteriores.

Si en nuestro caso, el atributo *tutor* de la tabla *alumno* se hubiera llamado *nro_reg* como el atributo de la clave primaria de *profesor*, entonces se podría haber hecho:

```
Select * from profesor natural join alumno o bien
Select * from profesor, alumno where profesor.nro_reg = alumno.nro_reg;
```

Pero si ocurriera que aun teniendo el mismo nombre, las dos columnas no tienen el mismo tipo de dato, habría que recurrir a la cláusula *using* que puede ser usada para especificar aquellas columnas que participarán en la equicomposición siempre y cuando las columnas que participen en la composición no tengan un calificador (nombre de tabla o alias):

Para comprobarlo vamos a cambiar el nombre a la columna *tutor* de *alumno* para que se llame como la clave primaria de *profesor* *nro_reg*:

```
ALTER TABLE alumno
RENAME COLUMN tutor to nro_reg;
```

Nombre	¿Nulo?	Tipo
NRO_EXP	NOT NULL	NUMBER(38)
NOMBRE		VARCHAR2(10)
APELLIDOS		VARCHAR2(20)
CURSO		NUMBER(38)
ESPECIALIDAD		VARCHAR2(10)
NRO_REG		NUMBER(38)

Y ahora se comprueba como funciona la cláusula *using*:

```
Select * from alumno join profesor using (nro_reg)
```

No se podría hacer:

```
Select * from alumno a join profesor p using (nro_reg)
```

Porque se le han añadido a alumno el calificador 'a' y a profesor el calificador 'p'

Otra posible combinación es usar la cláusula *join... on* en su lugar.

Se puede comprobar que estas tres sentencias:

$$\begin{array}{l} \text{select *} \\ \text{from alumno join profesor} \\ \text{on alumno.nro_reg} \\ \text{profesor.nro_reg} \end{array} = \begin{array}{l} \text{select *} \\ \text{from alumno join profesor} \\ \text{using (nro_reg)} \end{array} = \begin{array}{l} \text{select *} \\ \text{from alumno, profesor} \\ \text{where alumno.nro_reg} \\ \text{profesor.nro_reg} \end{array}$$

Van a dar este mismo resultado:

NRO_EX P	NOMBR E	APELLIDO S	CURS O	ESPECIALID	NRO_RE G	NRO_RE G	NOMBR E	APELLIDO S	DEP
1	Luis	Fernández	1	LSI	11	11	Maria	Martínez	1
2	Maria	Ruiz	1	LSI	11	11	Maria	Martínez	1
3	Carmen	López	1	LSI	11	11	Maria	Martínez	1
4	Elena	Martin	1	LSI	12	12	Pepe	Martínez	1

4. Operador UNION y UNION ALL

Al hacer la unión de dos sentencias *select*, se devolverán los resultados de cada una de las dos sentencias *select*, eliminando los duplicados.

Para poder hacer la unión es necesario que ambas sentencias *select* se hagan sobre el mismo número de columnas y con el mismo tipo. El nombre de las columnas no tienen que ser iguales.

Existe una variante, el operador **UNION ALL** que no elimina duplicados (en este caso no se puede usar la restricción **DISTINCT**).

4.1. Ejemplos

Trabajando con las tablas descritas en la sección anterior:

Podemos hacer la siguiente selección

```
SELECT nombre, apellidos FROM alumno
UNION
SELECT nombre, apellidos FROM profesor;
```

Cuya salida sería:

NOMBRE	APELLIDOS
Carmen	López
Elena	Martin
Luis	Fernández
Maria	Martínez
Maria	Ruiz
Pepe	Martínez

5. Operador INTERSECT

Al hacer la intersección de dos sentencias select, se devolverán los resultados comunes a las dos sentencias select.

Para poder hacer la interseccion es necesario que ambas sentencias select se hagan sobre el mismo número de columnas y con el mismo tipo. El nombre de las columnas no tienen porque ser iguales.

5.1. Ejemplos

```
SELECT nombre FROM alumno
INTERSECT
SELECT nombre FROM profesor;
```

Cuya salida sería:

NOMBRE
Maria

6. Operador MINUS

El operador MINUS devuelve las filas de la primera sentencia select que no están en la segunda sentencia select.

Para poder aplicar el operador MINUS es necesario que ambas sentencias select se hagan sobre el mismo número de columnas y con el mismo tipo. El nombre de las columnas no tiene porque ser iguales.

6.1. Ejemplos

```
SELECT nombre FROM alumno  
MINUS  
SELECT nombre FROM profesor;
```

Cuya salida sería:

NOMBRE
Carmen
Elena
Luis

7. EJERCICIOS

1. Devolver el nombre y los apellidos de todos los profesores (con las primeras letras en mayúsculas) que pertenezcan al departamento de informática y cuya dedicación sea completa ('TC')
2. Devolver el nombre y los apellidos de los profesores que den clase en locales con capacidad mayor que 50 personas. Devolver el resultado en letras minúsculas.
3. Seleccionar la periodicidad de las asignaturas impartidas por los profesores que pertenezcan a cualquier área del departamento de informática ('INF')
4. Seleccionar las asignaturas y el código del profesor que las imparte que cumplan que sus horas de teoría no son más que el 10% de las horas de práctica.
5. Seleccionar los nombres de los profesores a tiempo completo o de las asignaturas anuales
6. ¿Existe algún código de área que sea igual que el código de algún local?
7. Obtener las siglas de aquellas asignaturas de las que no se imparte docencia