ADMINISTRACIÓN DE BASE DE DATOS.

Tema Nº4:Lenguaje de Manipulación de Datos DML II.

Indicador de logro Nº4:Aplica los comandos DML y las funciones predefinidas en Oracle.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº4:**

Lenguaje de Manipulación de Datos DML II.

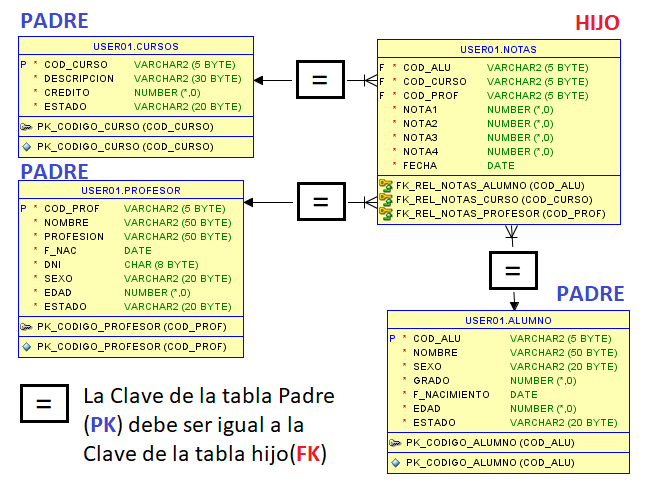
**Subtema 4.1:**

Extrae información realizando consultas avanzadas, tales como: Combinación Interna JOIN / INNER JOIN, NATURAL JOIN. Combinación Externa: LEFT OUTER JOIN, RIGHT OUTER JOIN).

**OBTENER DATOS DE MÚLTIPLES TABLAS**

Es más que habitual necesitar en una consulta datos que se encuentran distribuidos en varias tablas. Las bases de datos relacionales se basan en que los datos se distribuyen en Manual de SQL para Oracle 11g comandos internos SQL e SQL\*Plus 40 tablas que se pueden relacionar mediante un campo. Ese campo es el que permite integrar los datos de las tablas.

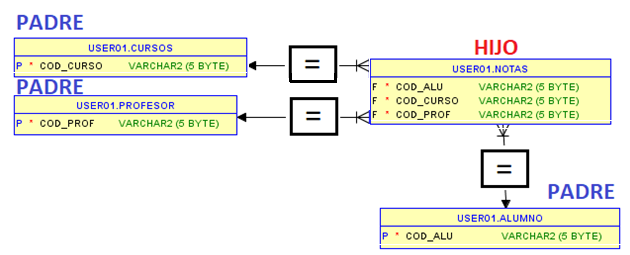
El Script de base de datos Semana04\_DML\_2.sql va generar el siguiente modelo relacional.



Este modelo relacional servirá para realizar las pruebas necesarias, en las tareas operaciones multi tabla.

**ASOCIAR TABLAS, USO EXCLUSIVO DE LA CLAUSULA WHERE. RELACIONES DE TABLAS BASADAS EN REGLA DE IGUALDAD.**

En este proceso solo realizaremos comparaciones entre las llaves de las tablas **padre** e **hijos** de nuestro modelo relacional. Si observamos detenidamente nuestro modelo relacional, vemos que algunas tablas tienen denominación **PADRE** y otros con la denominación **HIJO**, vista de sólo llaves PK y FK.

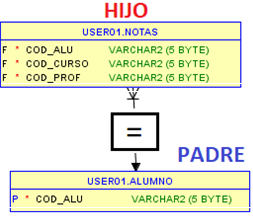


**SE DEBE APLICAR LA REGLA DE IGUALDAD DE LLAVES. PARA LA REALIZACIÓN DE CONSULTAS BASADAS EN VARIAS TABLAS.**

**La interpretación de estas relaciones, es la siguiente:**

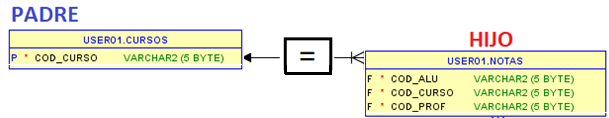
Interpretación de relación entre las tablas **ALUMNO** y **NOTAS**. La llave PK del Padre ALUMNO es igual a la Llave FK del Hijo NOTAS.

**ALUMNO.COD\_ALU = NOTAS.COD\_ALU**



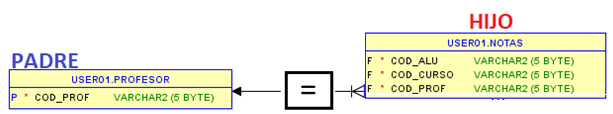
Interpretación de relación entre las tablas **CURSOS** y **NOTAS**. La llave PK del Padre CURSOS es igual a la Llave FK del Hijo NOTAS.

**CURSOS.COD\_CURSO = NOTAS.COD\_CURSO**



Interpretación de relación entre las tablas **PROFESOR** y **NOTAS**. La llave PK del Padre PROFESOR es igual a la Llave FK del Hijo NOTAS.

**PROFESOR.COD\_PROF = NOTAS.COD\_PROF**



**IMPORTANTE:**

La asociación de las tablas es posible usando la cláusula **WHERE**. Solo con comparar llaves se puede establecer una relación.

**Para realizar los ejercicios de consultas Multi-Tabla, debe abrir el Archivo Script de base de datos SEMANA04.sql. Este contiene las tablas y datos necesarios para tal proceso.**

**Ejemplo 1:**

Mostrar los cursos y calificaciones que tiene asignado el alumno JUAN PEREZ SANCHEZ.

Para dar solución a este ejemplo, analicemos que los campos solicitados en la consulta pertenecen a diferentes tablas, para ser exactos: ALUMNO, CURSOS, NOTAS y PROFESOR, para que esta consulta funcione, dichas tablas tienen que ser nombradas en la cláusula FROM, y en la cláusula WHERE comparar las llaves entre estas tablas, es decir **PADRE.LLAVE\_PK = HIJO.LLAVE\_FK**. use este ejemplo para desarrollar otros casos.

**SOLUCION:**

**SELECT** CURSOS.DESCRIPCION, NOTAS.NOTA1, NOTAS.NOTA2,

NOTAS.NOTA3, NOTAS.NOTA4

**FROM** ALUMNO, CURSOS, NOTAS, PROFESOR

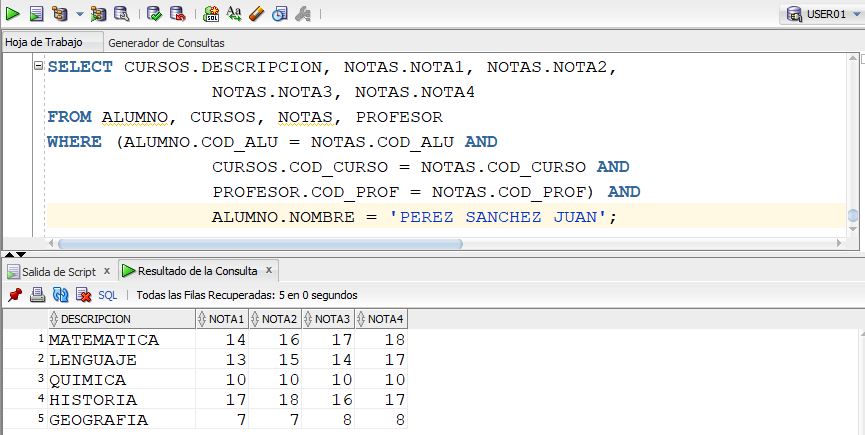
**WHERE (ALUMNO**.COD\_ALU **=** **NOTAS**.COD\_ALU **AND**

**CURSOS**.COD\_CURSO **=** **NOTAS**.COD\_CURSO **AND**

**PROFESOR**.COD\_PROF **=** **NOTAS**.COD\_PROF**)** **AND**

**ALUMNO**.NOMBRE = 'PEREZ SANCHEZ JUAN';

**Solución desde SQL developer:**



**IMPORTANTE:**

**SI LAS TABLAS A USAR ESTAN RELACIONADAS, ENTONCES EN LA CONSULTA DEBEN ESTAR RELACIONADAS TAMBIÉN.**

**JOIN - ON**

Permite establecer relaciones cuya condición se establece manualmente (**ON**), lo que permite realizar asociaciones más complejas o bien asociaciones cuyos campos en las tablas no necesariamente tienen el mismo nombre:

**Ejemplo 2:**

Mostrar los cursos y calificaciones que tiene asignado el alumno PEREZ SANCHEZ JUAN.

**SELECT** CURSOS.DESCRIPCION, NOTAS.NOTA1, NOTAS.NOTA2,

NOTAS.NOTA3, NOTAS.NOTA4

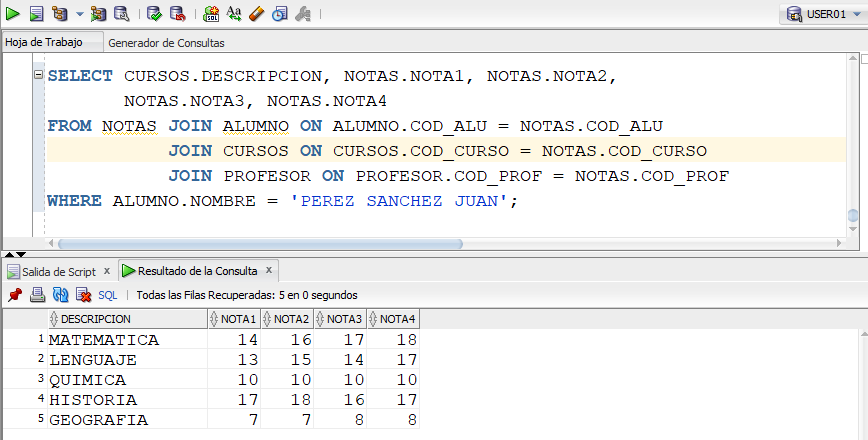
**FROM** **NOTAS** JOIN **ALUMNO** **ON** **ALUMNO**.COD\_ALU **=** **NOTAS**.COD\_ALU

JOIN **CURSOS** **ON** **CURSOS**.COD\_CURSO **=** **NOTAS**.COD\_CURSO

JOIN **PROFESOR** **ON** **PROFESOR**.COD\_PROF **=** **NOTAS**.COD\_PROF

**WHERE** ALUMNO.NOMBRE = 'PEREZ SANCHEZ JUAN';

**Solución desde SQL developer:**



**NATURAL JOIN**

Establece una relación de igualdad entre las tablas a través de los campos que tengan el mismo nombre en ambas tablas:

**Ejemplo 3:**

Mostrar los cursos, calificaciones y promedios que tiene asignado el alumno PEREZ SANCHEZ JUAN. Use alias para cada tabla.

**SELECT** C.DESCRIPCION, N.NOTA1, N.NOTA2, N.NOTA3, N.NOTA4,

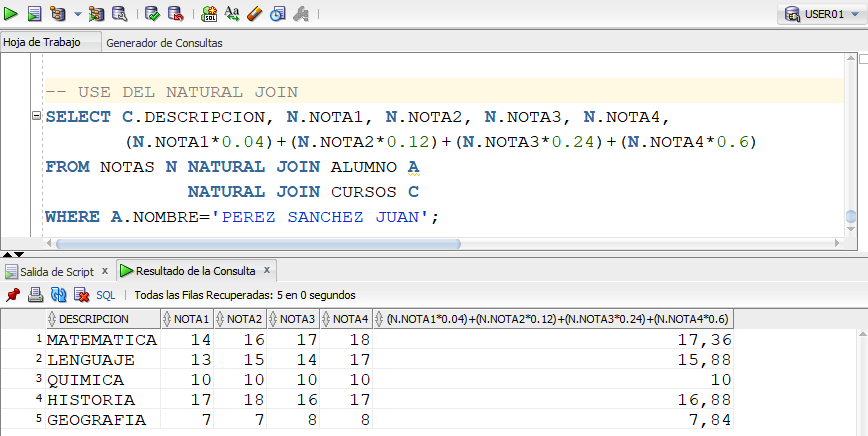
(N.NOTA1\*0.04) + (N.NOTA2\*0.12) + (N.NOTA3\*0.24) + (N.NOTA4\*0.6)

**FROM** **NOTAS** N **NATURAL JOIN** **ALUMNO** A

**NATURAL JOIN** **CURSOS** C

**WHERE** A.NOMBRE='PEREZ SANCHEZ JUAN';

**Solución desde SQL developer:**



**INNER JOIN:**

Otra opción es utilizar la cláusula **INNER JOIN**. Su sintaxis es idéntica a la de una consulta **SELECT** habitual, con la particularidad de que en la cláusula **FROM** sólo aparece una tabla o vista, añadiéndose el resto de tablas a través de cláusulas **INNER JOIN**.

**Ejemplo 4:**

Visualice a los docentes que han impartido clases de MATEMATICA.

**SELECT** **DISTINCT** **PROFESOR**.NOMBRE **FROM** **NOTAS**

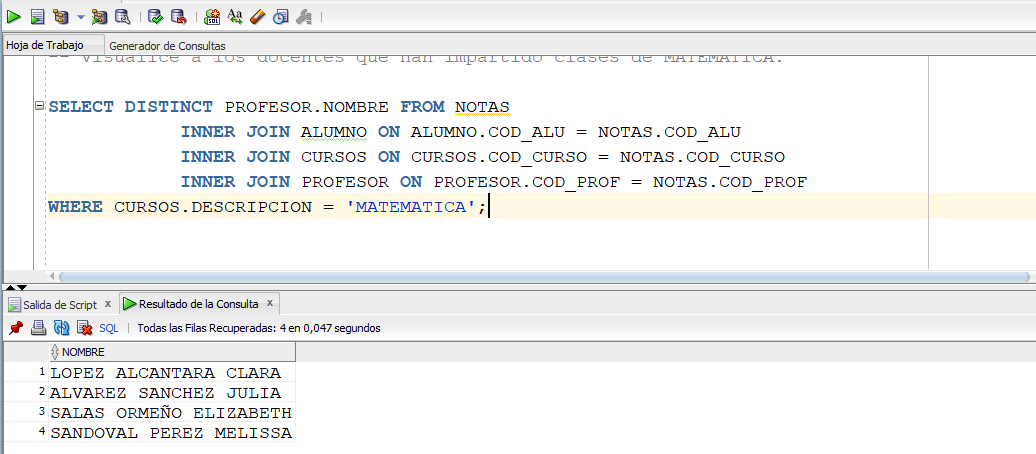
**INNER JOIN** **ALUMNO** **ON** **ALUMNO**.COD\_ALU **=** **NOTAS**.COD\_ALU

**INNER JOIN** **CURSOS** **ON** **CURSOS**.COD\_CURSO **=** **NOTAS**.COD\_CURSO

**INNER JOIN** **PROFESOR** **ON** **PROFESOR**.COD\_PROF **=** **NOTAS**.COD\_PROF

**WHERE** CURSOS.DESCRIPCION = 'MATEMATICA';

**Solución desde SQL developer:**



**JOIN USING.**

Mostrar los cursos, calificaciones y promedios que tiene asignado el alumno PEREZ SANCHEZ JUAN. Use alias para cada tabla.

**Ejemplo 5:**

Muestre las calificaciones con promedio incluido del alumno PEREZ SANCHEZ JUAN

**SELECT** C.DESCRIPCION, N.NOTA1, N.NOTA2, N.NOTA3, N.NOTA4,

(N.NOTA1\*0.04) + (N.NOTA2\*0.12) + (N.NOTA3\*0.24) + (N.NOTA4\*0.6)

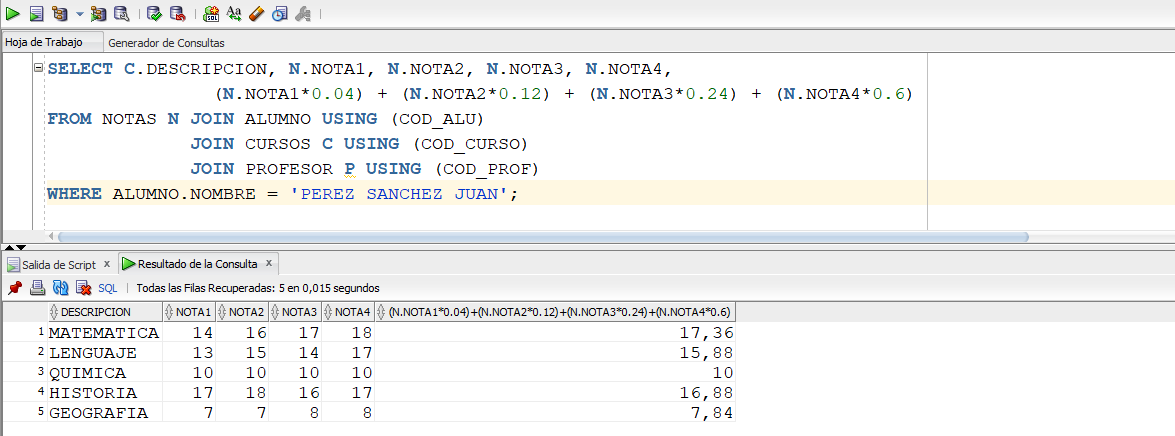
**FROM** **NOTAS** JOIN **ALUMNO** **USING** (COD\_ALU)

JOIN **CURSOS** **USING** (COD\_CURSO)

JOIN **PROFESOR** **USING** (COD\_PROF)

**WHERE** ALUMNO.NOMBRE = 'PEREZ SANCHEZ JUAN';

**Solución desde SQL developer:**



**CROSS JOIN**

Utilizando la opción CROSS JOIN se realiza un producto cruzado entre las tablas indicadas. Las combinaciones cruzadas (CROSS JOIN) muestran todas las combinaciones de todos los registros de las tablas combinadas. Para este tipo de **join** no se incluye una condición de enlace. Se genera el producto cartesiano en el que el número de filas del resultado es igual al número de registros de la primera tabla multiplicado por el número de registros de la segunda tabla, es decir, si hay 3 registros en una tabla y 4 en la otra, retorna 12 filas.

**Ejemplo 6:**

Realice una consulta donde se combine los datos de la tabla **ALUMNO** y **CURSOS**, provocando saber que asignaturas podrían cursar cada alumno.

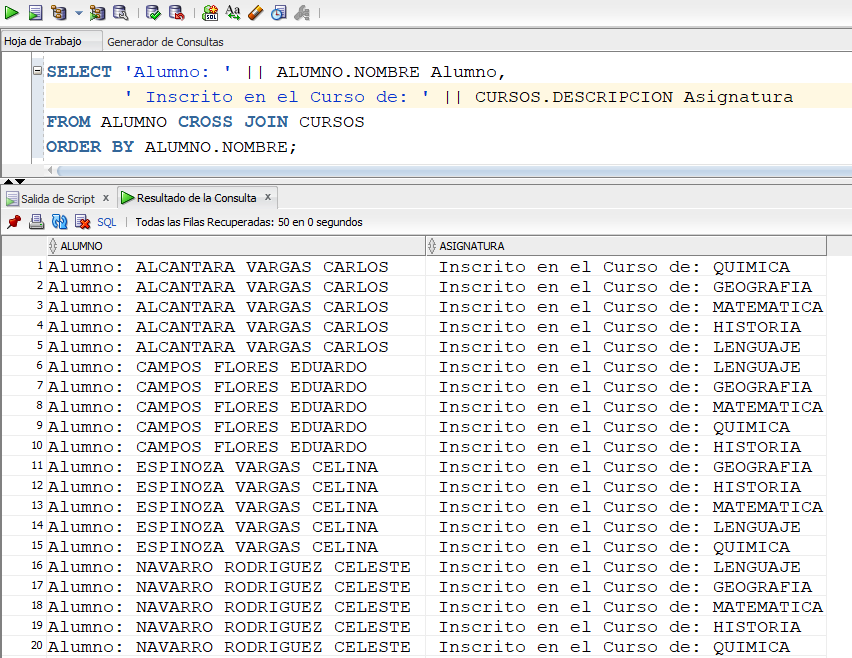
**SELECT** 'Alumno: ' || ALUMNO.NOMBRE Alumno,

' Inscrito en el Curso de: ' || CURSOS.DESCRIPCION Asignatura

**FROM** **ALUMNO** **CROSS JOIN** **CURSOS**

**ORDER BY** ALUMNO.NOMBRE

**Solución desde SQL developer:**



**Nota:**

Este ejemplo es semejante a: combinar comidas y postres, cada comida tendrá un postre.

**COMBINACIÓN EXTERNA**

**LEFT OUTER JOIN – RIGHT OUTER JOIN**

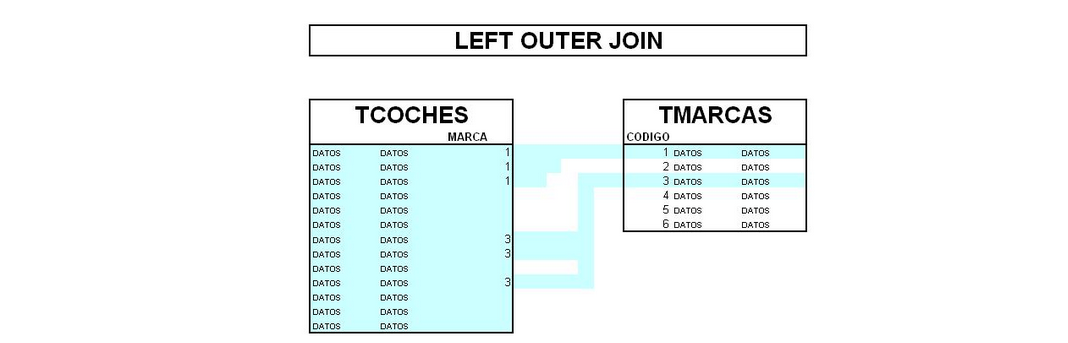
La combinación interna es excluyente. Esto quiere decir que si un registro no cumple la condición de combinación no se incluye en los resultados. De este modo en el ejemplo anterior si un coche no tiene grabada la marca no se devuelve en mi consulta.

Según la naturaleza de nuestra consulta esto puede ser una ventaja, pero en otros casos significa un serio problema. Para modificar este comportamiento SQL pone a nuestra disposición la combinación externa. La combinación externa no es excluyente. La sintaxis es muy parecida a la combinación interna,

La combinación externa puede ser diestra o siniestra, LEFT OUTER JOIN o RIGHT OUTER JOIN. Con LEFT OUTER JOIN obtenemos todos los registros de la tabla que situemos a la izquierda de la cláusula JOIN, mientras que con RIGHT OUTER JOIN obtenemos el efecto contrario. Como mejor se ve la combinación externa es con un ejemplo

Esta consulta devolverá todos los registros de la tabla TCOCHES, independientemente de que tengan marca o no.  En el caso de que el coche no tenga marca se devolverá el valor null para los campos de la tabla TMARCAS. Visualmente (la consulta devuelve los datos en celeste).

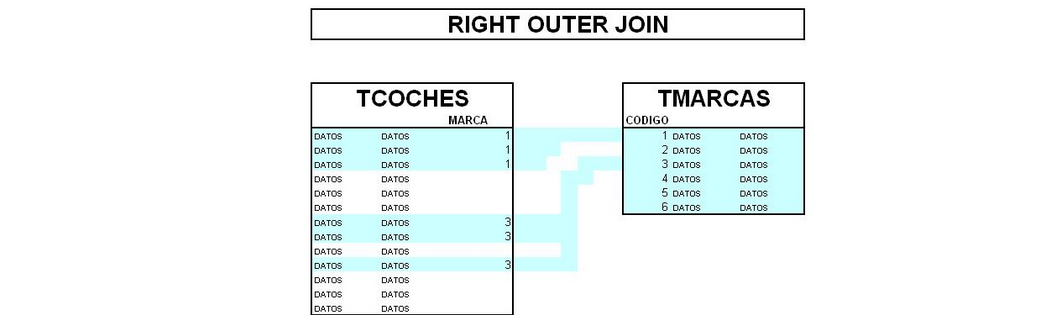
**SELECT** tCoches.matricula,   
        tMarcas.marca,   
        tCoches.modelo,   
        tCoches.color,   
        tCoches.numero\_kilometros,   
        tCoches.num\_plazas   
**FROM** tCoches **LEFT OUTER JOIN** tMarcas **ON** tCoches.marca = tMarcas.codigo**;**



Esta consulta devolverá los registros de la tabla TCOCHES que tengan marca relacionada y todos los registros de la tabla TMARCAS, tengan algún registro en TCOCHES o no. Visualmente (la consulta devuelve los datos en azul).

**El mismo ejemplo con RIGHT OUTER JOIN.**

**SELECT** tCoches.matricula,   
        tMarcas.marca,   
        tCoches.modelo,   
        tCoches.color,   
        tCoches.numero\_kilometros,   
        tCoches.num\_plazas   
**FROM** tCoches **RIGHT OUTER JOIN** tMarcas **ON** tCoches.marca = tMarcas.codigo;

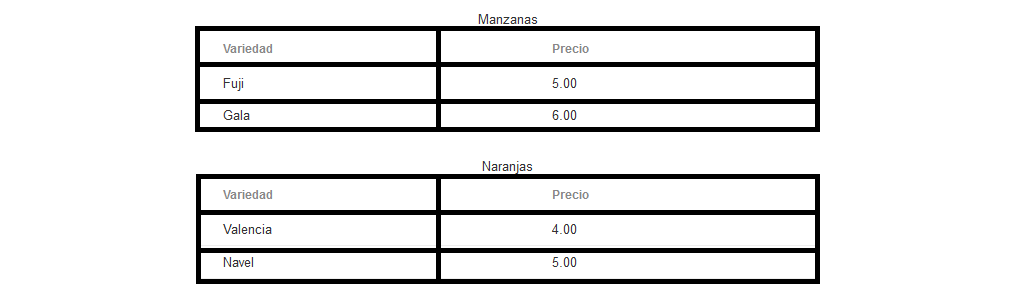


**FULL OUTER JOIN**

Para empezar, recordemos que un FULL OUTER JOIN es así como un LEFT JOIN y RIGHT JOIN a la vez. O bien, como vimos en [Inner, Outer, Natural & Cross JOINs](http://luauf.com/2008/05/31/inner-outer-natural-cross-joins/), FULL OUTER JOIN combina los resultados de dos o más tablas, tengan o no coincidencia entre sí.

**Veamos un ejemplo demostrativo:**

**Disponemos de dos tablas, MANZANAS y NARANJAS: y sus contenidos.**



**CREATE TABLE** MANZANAS (

Variedad Varchar2 (30),

Precio Numeric (10,2));

**CREATE TABLE** NARANJAS (

Variedad Varchar2 (30),

Precio Numeric (10,2));

**INSERT INTO** MANZANAS VALUES ('FUJI', 5.00);

**INSERT INTO** MANZANAS VALUES ('GALA', 6.00);

**COMMIT;**

**INSERT INTO** NARANJAS VALUES ('VALENCIA',4.00);

**INSERT INTO** NARANJAS VALUES ('NAVEL',5.00);

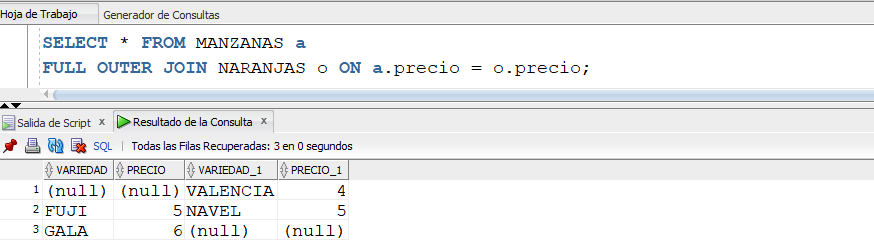
**COMMIT;**

Si realizamos un **FULL OUTER JOIN** por precio, tenemos

**SELECT** \* **FROM** MANZANAS a

**FULL OUTER JOIN** NARANJAS o **ON** a.precio = o.precio

**Solución desde SQL developer:**



**COMBINACIONES ESPECIALES, Uniones**

La palabra UNION permite añadir el resultado de un SELECT a otro SELECT. Para ello ambas instrucciones tienen que utilizar el mismo número y tipo de columnas. Ejemplo:

**SELECT** nombre **FROM** provincias

**UNION**

**SELECT** nombre **FROM** comunidades

El resultado es una tabla que contendrá nombres de provincia y de comunidades. Es decir, UNION crea una sola tabla con registros que estén presentes en cualquiera de las consultas. Si están repetidas sólo aparecen una vez, para mostrar los duplicados se utiliza UNION ALL en lugar de la palabra UNION.

**INTERSECCIONES**

De la misma forma, la palabra INTERSECT permite unir dos consultas SELECT de modo que el resultado serán las filas que estén presentes en ambas consultas.

**DIFERENCIA**

Con MINUS también se combinan dos consultas SELECT de forma que aparecerán los registros del primer SELECT que no estén presentes en el segundo.

**Ejemplo usando UNION.**

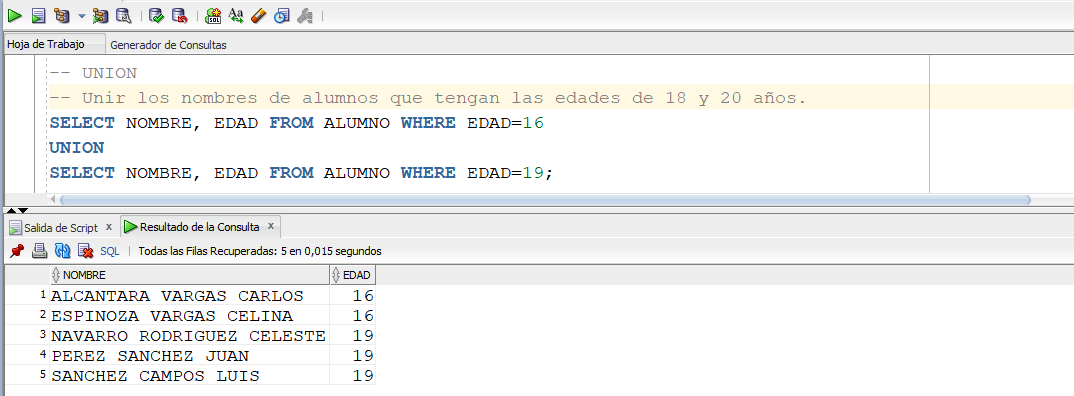
Unir los nombres de alumnos que tengan las edades de 18 y 20 años.

**SELECT** NOMBRE, EDAD **FROM** ALUMNO **WHERE** EDAD=16

**UNION**

**SELECT** NOMBRE, EDAD **FROM** ALUMNO **WHERE** EDAD=19;

**Solución desde SQL developer:**



**Ejemplo usando INSTERSECT.**

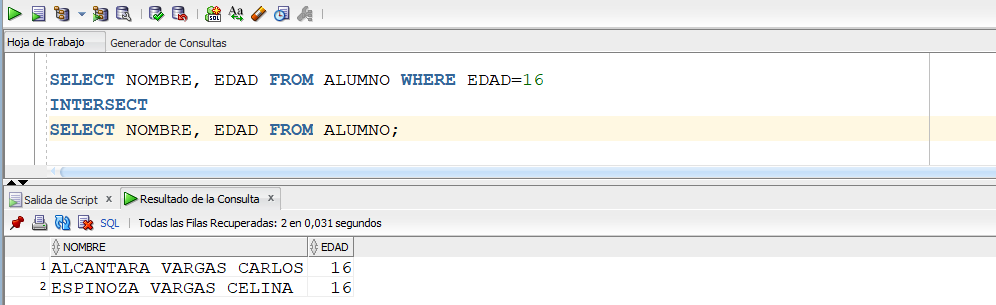
Este ejemplo muestra a los alumnos cuya edad sea de 16 años, pero debe pasar un filtro dependiente de la segunda consulta.

**SELECT** NOMBRE, EDAD **FROM** ALUMNO **WHERE** EDAD=16

**INTERSECT**

**SELECT** NOMBRE, EDAD **FROM** ALUMNO;

**Solución desde SQL developer:**



**Ejemplo usando MINUS.**

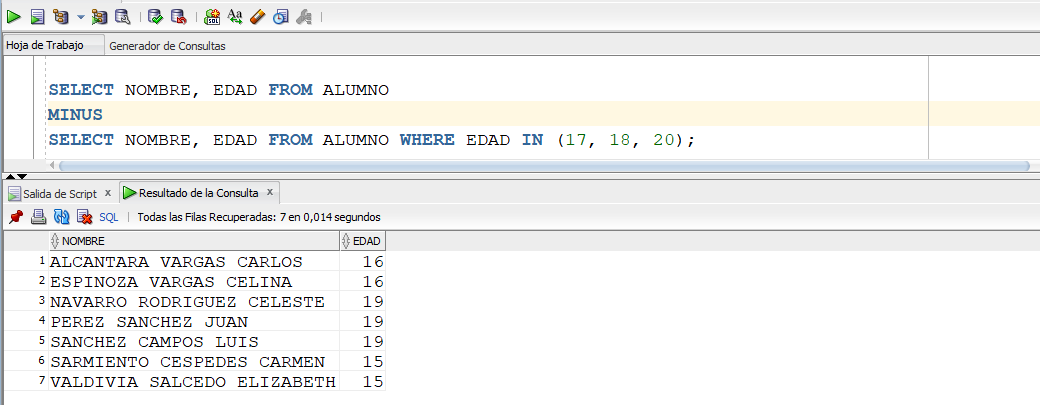
Este ejemplo muestra a toso los alumnos menos aquellos cuyas edades sea de 17, 18 y 20 años.

**SELECT** NOMBRE, EDAD **FROM** ALUMNO

**MINUS**

**SELECT** NOMBRE, EDAD **FROM** ALUMNO **WHERE** EDAD IN (17, 18, 20);

**Solución desde SQL developer:**



**Subtema 4.2:**

Extraer información mediante Sub Consultas.

Se trata de una técnica que permite utilizar el resultado de una tabla SELECT en otra consulta SELECT. Permite solucionar problemas en los que el mismo dato aparece dos veces.

La sintaxis es:

**SELECT** Lista\_expresiones

**FROM** tabla

**WHERE** expresión operador **(SELECT** Lista\_expresiones **FROM** tabla**);**

Se puede colocar el **SELECT** dentro de las cláusulas **WHERE**, **HAVING** o **FROM**. El operador puede ser: **>, <, >=, <=, <>, = o IN**.

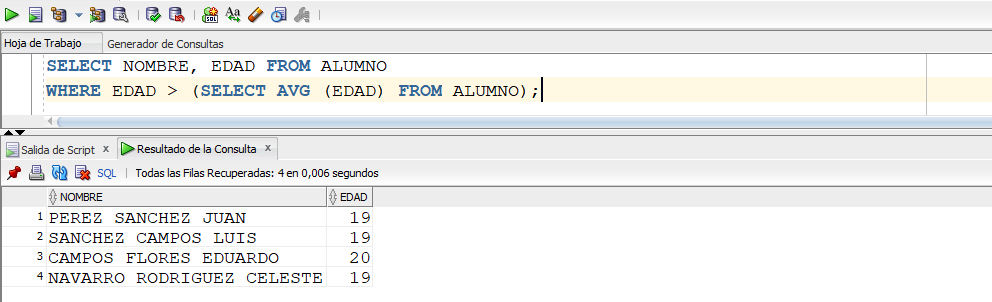
**Ejemplo 1:**

Muestre a los alumnos que superen el promedio de edades.

**SELECT** NOMBRE, EDAD FROM ALUMNO

**WHERE** EDAD > **(SELECT** AVG (EDAD) **FROM** ALUMNO**);**

**Solución desde SQL developer:**



**Ejemplo 2:**

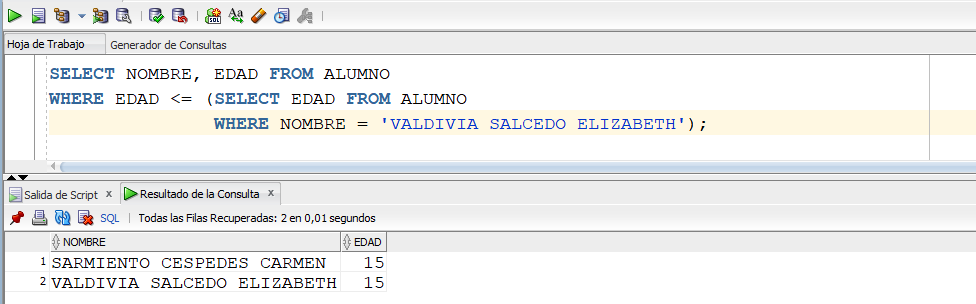
Se desea saber quiénes son los alumnos que no superan la edad de VALDIVIA SALCEDO ELIZABETH.

**SELECT** NOMBRE, EDAD **FROM** ALUMNO

**WHERE** EDAD <= **(SELECT** EDAD **FROM** ALUMNO

**WHERE** NOMBRE = 'VALDIVIA SALCEDO ELIZABETH'**);**

**Solución desde SQL developer:**



**Ejemplo 3:**

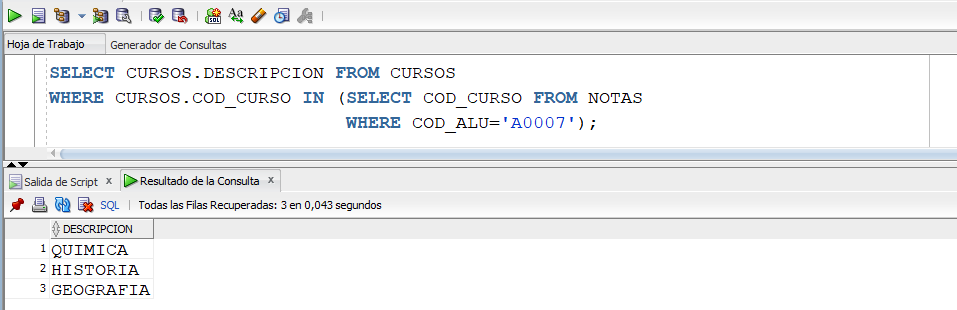
Se desea saber los cursos que tiene asignado el alumno CAMPOS FLORES EDUARDO, cuyo código de alumno es: **A0007**.

**SELECT** CURSOS.DESCRIPCION **FROM** CURSOS

**WHERE** CURSOS.COD\_CURSO **IN** (**SELECT** COD\_CURSO **FROM** NOTAS

**WHERE** COD\_ALU='A0007');

**Solución desde SQL developer:**



**Ejemplo 4:**

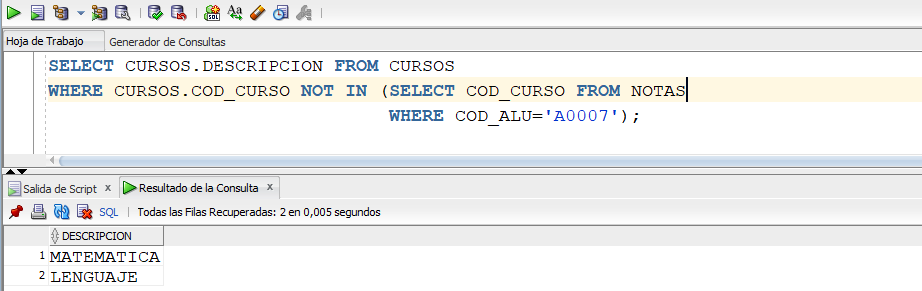
Se desea saber los cursos que no tiene asignado el alumno CAMPOS FLORES EDUARDO, cuyo código de alumno es: A0007.

**SELECT** CURSOS.DESCRIPCION **FROM** CURSOS

**WHERE** CURSOS.COD\_CURSO **NOT IN** **(SELECT** COD\_CURSO **FROM** NOTAS

**WHERE** COD\_ALU='A0007'**);**

**Solución desde SQL developer:**



**Consultas con ROWNUM**

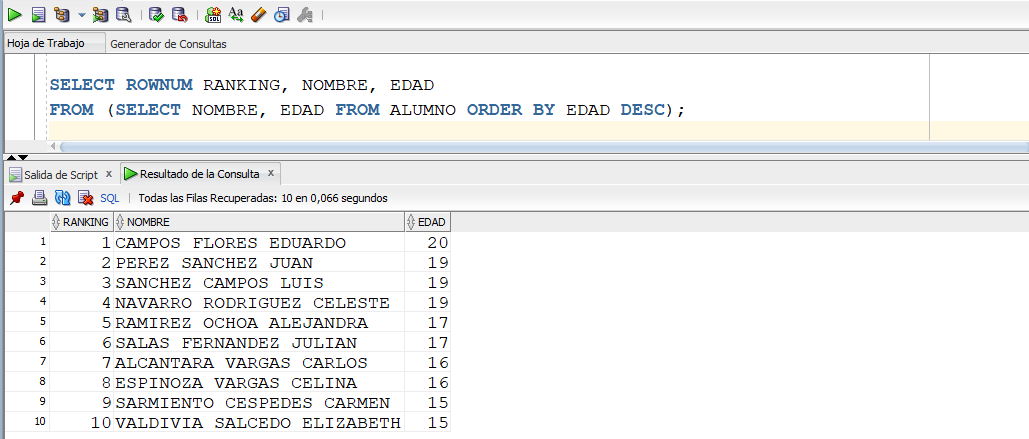
La función ROWNUM devuelve el número de la fila de una consulta. Por ejemplo:

Ejemplo: Mostrar el ranking de edades entre los alumnos.

**SELECT** **ROWNUM** RANKING, NOMBRE, EDAD

**FROM** (**SELECT** NOMBRE, EDAD **FROM** ALUMNO **ORDER BY** EDAD DESC);

**Solución desde SQL developer:**



**Subtema 4.3:**

Empleo del comando MERGE.

La sentencia MERGE declaración de Oracle [selecciona](https://www.oracletutorial.com/oracle-basics/oracle-select/) datos de una o más tablas de origen y los [actualiza](https://www.oracletutorial.com/oracle-basics/oracle-update/) o los [inserta](https://www.oracletutorial.com/oracle-basics/oracle-insert/) en una tabla de destino. La MERGE declaración le permite especificar una condición para determinar si actualizar o insertar datos en la tabla de destino.

**A continuación, se ilustra la sintaxis de la MERGE declaración de Oracle:**

**MERGE** **INTO** target\_table

**USING** source\_table

**ON** search\_condition

**WHEN MATCHED THEN**

**UPDATE** SET col1 = value1, col2 = value2, ...

**WHERE** <update\_condition>

[**DELETE** **WHERE** <delete\_condition>]

**WHEN** **NOT** **MATCHED** THEN

**INSERT** (col1, col2...)

**VALUES** (value1, value2...)

**WHERE** <insert\_condition>;

Las siguientes declaraciones crean las tablas **members** y **member\_staging**:

**CREATE TABLE** members (

member\_id NUMBER PRIMARY KEY,

first\_name VARCHAR2(50) NOT NULL,

last\_name VARCHAR2(50) NOT NULL,

rank VARCHAR2(20)

);

**CREATE TABLE** member\_staging AS

**SELECT** \* **FROM** members;

Las siguientes INSERT declaraciones insertan datos de muestra en las tablas members y member\_staging:

**--INSERCIÓN DE REGISTROS EN LA TABLA MEMBERS.**

INSERT INTO members (member\_id, first\_name, last-named, rank) VALUES(1,'Abel','Wolf','Gold');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(2,'Clarita','Franco','Platinum');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(3,'Darryl','Giles','Silver');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(4,'Dorthea','Suarez','Silver');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(5,'Katrina','Wheeler','Silver');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(6,'Lilian','Garza','Silver');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(7,'Ossie','Summers','Gold');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(8,'Paige','Mcfarland','Platinum');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(9,'Ronna','Britt','Platinum');

INSERT INTO members (member\_id, first\_name, last\_name, rank) VALUES(10,'Tressie','Short','Bronze');

**--INSERCIÓN DE REGISTROS EN LA TABLA MEMBER\_STAGING:**

INSERT INTO member\_staging (member\_id, first\_name, last\_name, rank) VALUES(1,'Abel','Wolf','Silver');

INSERT INTO member\_staging (member\_id, first\_name, last\_name, rank) VALUES(2,'Clarita','Franco','Platinum');

INSERT INTO member\_staging (member\_id, first\_name, last\_name, rank) VALUES(3,'Darryl','Giles','Bronze');

INSERT INTO member\_staging (member\_id, first\_name, last\_name, rank) VALUES(4,'Dorthea','Gate','Gold');

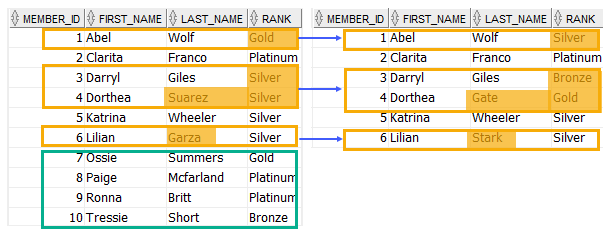
INSERT INTO member\_staging (member\_id, first\_name, last\_name, rank) VALUES(5,'Katrina','Wheeler','Silver');

INSERT INTO member\_staging (member\_id, first\_name, last\_name, rank) VALUES(6,'Lilian','Stark','Silver');

Al actualizar los datos de la members tabla a la member\_staging tabla, debemos realizar las siguientes acciones:

* Actualizamos las filas con ID de miembro 1, 3, 4 y 6 porque el rango o el apellido de estos miembros en estas tablas son diferentes.
* Insertamos las filas con ID de miembro 7 a 10 porque estas filas existen en la members tabla, pero no en la member\_stagingtabla.

En total, se deben fusionar 8 filas.



La siguiente es la MERGE declaración que realiza todas estas acciones en una sola toma.

**MERGE INTO** member\_staging x

**USING** (**SELECT** member\_id, first\_name, last\_name, rank **FROM** members) y

**ON** (x.member\_id = y.member\_id)

**WHEN MATCHED** THEN

**UPDATE** **SET** x.first\_name = y.first\_name,

x.last\_name = y.last\_name,

x.rank = y.rank

**WHERE** x.first\_name <> y.first\_name OR

x.last\_name <> y.last\_name OR

x.rank <> y.rank

**WHEN NOT MATCHED** THEN

INSERT (x.member\_id, x.first\_name, x.last\_name, x.rank)

VALUES (y.member\_id, y.first\_name, y.last\_name, y.rank);

**Realizar las pruebas desde SQL DEVELOPER.**

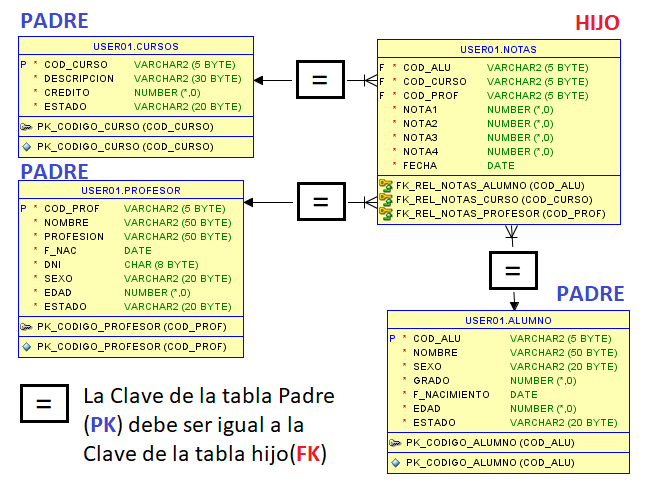
**CONCLUSIONES Y RECOMENDACIONES DE LA EXPERIENCIA**

El objetivo de este capítulo es comprender el uso de consultas avanzadas, de esta manera el estudiante se compenetra más con la base de datos de Oracle, al analizar la información, resultado de las búsquedas realizadas de manera exitosa. Mediante el uso de las herramientas SQL \*PLUS y SQL DEVELOPER.

**ACTIVIDAD VIRTUAL**

De acuerdo al material presentado del Tema 04, Responda a las siguientes Preguntas:

DE ACUERDO AL SIGUIENTE MODELO RELACIONAL.



Realizar las siguientes consultas:

* FECILITE UNA CONSULTA, USANDO LA CLAUSULA WHERE
* FACILITE UNA CONSULTA, USANDO LA CLAUSULA JOIN
* FACILITE UNA CONSULTA, USANDO LA CLAUSULA INNER JOIN.
* FACILITE UNA CONSULTA, USANDO LA CLAUSULA NATURAL JOIN.
* FACILITE UNA CONSULTA, USANDO LA CLAUSULA CROSS JOIN.
* FACILITE UNA SUB CONSULTA.