Universidad Nacional Autónoma de Honduras
Ingeniería en Sistemas
Bases de Datos I

Resumen Unidad II

Ing. Erick Vladimir Reyes Marín

Contenido

Recomendaciones al momento de diseñar bases de datos:

- Evitar utilizar acentos, símbolos, ñ
- Nombres descriptivos
- Cambiar nombres de los campos de las llaves foráneas
- Tipos de datos para una llave primaria y llaves foráneas.
- Orden de los campos.
- Tamaño de los Varchar
- Faltan campos
- Tablas intermedias

Sentencias SQL

Sentencias DDL (Data Definition Language)

Las sentencias DDL son las que se utilizan para definir, modificar o eliminar estructuras o elementos físicos dentro de la base de datos.
A continuación se definen algunas sentencias DML básicas:

Crear una tabla

```
CREATE TABLE NOMBRE_TABLA
(
  CAMPO1 TIPO_DATO [MODIFICADORES],
  CAMPO2 TIPO_DATO [MODIFICADORES],

  CAMPON TIPO_DATO [MODIFICADORES],
  CONSTRAINT NOMBRE_PK PRIMARY KEY
  (
    CAMPOS_LLAVE_PRIMARIA
  )
  ENABLE
);
```

Sentencia DDL para agregar una nueva columna:

```
ALTER TABLE NOMBRE_TABLA
ADD (CAMPO1 TIPO_DATO [MODIFICADORES], ... CAMPON TIPO_DATO
[MODIFICADORES]);
```

Eliminar una columna:

```
ALTER TABLE NOMBRE_TABLA
DROP COLUMN CAMPO;
```

Modificar una columna:

```
ALTER TABLE NOMBRE_TABLA
MODIFY (CAMPO TIPO_DATO_NUEVO);
```

Eliminar una tabla:

```
DROP TABLE NOMBRE_TABLA.
```

Sentencias DML

```
INSERT INTO TBL_CLIENTES (
     CODIGO_CLIENTE,
     NOMBRE_CLIENTE,
     APELLIDO_CLIENTE,
     GENERO
) VALUES (2, 'Maria', 'Ruiz', 'F');

INSERT INTO NOMBRE_TABLA(CAMPOS) VALUES(VALORES)
```

Actualizar un registro:

```
UPDATE NOMBRE_TABLA
SET CAMPO1=NUEVO_VALOR1,
     CAMPO2=NUEVO_VALOR2,
     ....
     CAMPON=NUEVO_VALOR_N
WHERE CONDICION;

UPDATE TBL_CLIENTES
SET APELLIDO_CLIENTE = 'Juarez'
WHERE CODIGO_CLIENTE = 2
```

Eliminar un registro:

```
DELETE FROM NOMBRE_TABLA
```

```
WHERE CONDICIONES;
```

Gestion de transacciones (Activa)
Unicamente aplica a instrucciones DML

*Hacer efectiva una transaccion
commit;

*Deshacer los cambios hasta el último commit
rollback;

Consulta de informacion (QL)

Instrucciones cuya principal funcionalidad es consultar informacion para ser mostrada o ser gestionada por el usuario desde un aplicativo.

Consulta basica

```
SELECT *
FROM NOMBRE_TABLA;
```

Es recomendable escribir siempre los nombres de los campos que se necesitan.

Consulta por campos

```
SELECT CAMPOS
FROM NOMBRE_TABLA;
```

Consulta por campos

```
SELECT CAMPOS
FROM NOMBRE_TABLA
WHERE CONDICIONES;
```

Operadores logicos

*AND
*OR
*NOT

*IN (Retorna lor registros que cumplan con una lista especifica de opciones)

Operadores de comparacion

=
>=
<=

!= o <>
BETWEEN

Consultas con el mismo resultado:

```
SELECT *
FROM tbl_clientes
WHERE nombre_cliente != 'Maria';

SELECT *
FROM tbl_clientes
WHERE nombre_cliente <> 'Maria';

SELECT *
FROM tbl_clientes
WHERE NOT nombre_cliente = 'Maria';
```

Gestión de transacciones

//Pendiente

Gestión de fechas

Para convertir de caracteres a fecha se utiliza la función TO_DATE(CADENA, FORMATO): retorna un date.

Para convertir de fecha a carácter se utiliza la función TO_CHAR(FECHA, FORMATO): retorna una cadena.

A continuación se muestra una tabla que define los diferentes caracteres que pueden ser utilizados para definir el formato:

http://mundodb.es/fechas-funciones-y-tipos-en-oracle

Siglos y años	
CC	Siglo
SCC	Siglo. Si es AC (Antes de Cristo), lleva un signo -
YYYY	Año, formato de 4 dígitos
SYYY	Año, formato de 4 dígitos. Si es AC lleva un signo -
YY	Año, formato de 2 dígitos
YEAR	Año, escrito en letras y en inglés (por ejemplo, 'TWO THOUSAND TWO')

SYEAR	Ídem, pero si es AC lleva el signo -
BC	Antes o Después de Cristo (AC o DC) para usar con los anteriores, por ejemplo YYYY BC
Meses	
Q	Trimestre: Ene-Mar=1, Abr-Jun=2, Jul-Sep=3, Oct-Dic=4
MM	Número de mes (1-12)
RM	Número de mes en números romanos (I-XII)
MONTH	Nombre del mes completo rellenado con espacios hasta 10 espacios (SEPTIEMBRE)
FMMONTH	Nombre del mes completo, sin espacios adicionales
MON	Tres primeras letras del mes: ENE, FEB,...
Semanas	
WW	Semana del año (1-52)
W	Semana del mes (1-5)
Días	
DDD	Día del año (1-366)
DD	Día del mes (1-31)
D	Día de la semana (1-7)
DAY	Nombre del día de la semana rellenado a 9 espacios (MIÉRCOLES)
FMDAY	Nombre del día de la semana, sin espacios
DY	Tres primeras letras del nombre del día de la semana
DDTH	Día (ordinal): 7TH
DDSPTH	Día ordinal en palabra, en inglés: SEVENTH
Horas	
HH	Hora del día (1-12)
HH12	Hora del día (1-12)
HH24	Hora del día (1-24)
SPHH	Hora del día, en palabra, inglés: SEVEN
AM	am o pm, para usar con HH, como 'HH:MI am'
PM	am o pm

A.M.	a.m. o p.m.
P.M.	a.m. o p.m.
Minutos y segundos	
MI	Minutos (0-59)
SS	Segundos (0-59)
SSSS	Segundos después de medianoche (0-86399)

Campos auto numéricos, Secuencias

Mediante las secuencias, Oracle puede proporcionar una lista consecutiva de números unívocos que sirve para simplificar las tareas de programación. La primera vez que una consulta llama a una secuencia, se devuelve un valor predeterminado.

Para crear una secuencia en Oracle mediante SQL utilizaremos el comando create sequence con la siguiente sintaxis:

```
CREATE SEQUENCE nombre_secuencia
INCREMENT BY numero_incremento
START WITH numero_por_el_que_empezara
MAXVALUE valor_maximo | NOMAXVALUE
MINVALUE valor_minimo | NOMINVALUE
CYCLE | NOCYCLE
ORDER | NOORDER
```

Por ejemplo, si queremos crear una secuencia que empiece en 100 y se incremente de uno en uno utilizaremos la siguiente consulta SQL:

```
CREATE SEQUENCE secuencia_id_cliente
INCREMENT BY 1
START WITH 100
```

Para utilizar la secuencia, en primer lugar, crearemos una tabla de prueba (para insertar un registro y comprobar que la secuencia anterior funciona correctamente):

```
CREATE TABLE  clientes (
     codigo number not null primary key,
     nombre varchar2(100) unique not null
)
```

Para utilizar la secuencia creada en una inserción de fila:

```
INSERT INTO clientes(codigo,nombre)
VALUES (secuencia_id_cliente.NEXTVAL, 'Juan');
```

Realizamos otra inserción para comprobar que el incremento es de 1:

```
INSERT INTO clientes(codigo,nombre)
VALUES (secuencia_id_cliente.NEXTVAL, Pedro);
```

Algebra Relacional

Se define como un conjunto de operaciones que se ejecutan sobre las relaciones (tablas) para obtener un resultado, el cual es otra relación.

Selección (Select)

Este operador se aplica a una relación R produciendo una nueva relación con un subconjunto de tuplas de R. Las tuplas de la relación resultante son las que satisfacen una condición C sobre algún atributo de R. Es decir selecciona **filas (tuplas)** de una tabla según un cierto criterio C. El criterio C es una expresión condicional, similar a las declaraciones del tipo "if", es "booleana" esto quiere decir que para cada tupla de R toma el valor Verdad(true) o Falso(false).

(relación = tabla)
(tupla = registro)

Para representar `Select` en álgebra relacional se utiliza la letra griega sigma σ. Por lo tanto, si utilizamos la notación $\sigma_c R$ queremos decir que se aplica la condición C a cada tupla de R. Si la condición es Verdad `true`, dicha tupla pertenecerá al resultado y si es Falsa `false`, dicha tupla no será seleccionada. El esquema de la relación resultante es el mismo esquema R, se muestran los atributos en el mismo orden que se usan en la tabla R.

Ejemplo:

Algebra relacional:

$\sigma_{edad>30}$ Ingenieros

SQL:

SELECT * FROM INGENIEROS WHERE EDAD > 30;

Proyección (Project)

El operador `Proyección` se utiliza para producir una nueva relación desde R. Esta nueva relación contiene sólo algunos de los atributos de R, es decir, realiza la selección de algunas de las **columnas** de una tabla R.

Notación en Álgebra Relacional

`Project` en Álgebra Relacional se representa por la letra griega **pi**: $\pi_{(A_1,...,A_n)}R$

Ejemplo:

Algebra relacional:

$\pi_{(id,nombre)}$Ingenieros

SQL:

SELECT ID, NOMBRE FROM INGENIEROS;

Producto cartesiano (Cross-Product)

En teoría de conjuntos, el `producto cartesiano` de dos conjuntos es una operación que resulta en otro conjunto cuyos elementos son todos los pares ordenados que pueden formarse tomando el primer elemento del par del primer conjunto, y el segundo elemento del segundo conjunto. En el Álgebra Relacional se mantiene esta idea con la diferencia que R y S son relaciones, entonces los miembros de R y S son tuplas, que generalmente consisten de más de un componente, cuyo resultado de la vinculación de una tupla de R con una tupla de S es una tupla más larga, con un componente para cada uno de los componentes de las tuplas constituyentes. Es decir`Cross-product` define una relación que es la concatenación de cada una de las filas de la relación R con cada una de las filas de la relación S.

Notación en Álgebra Relacional

Para representar `Cross-product` en Álgebra Relacional se utiliza la siguiente terminología:

$$R \times S$$

Ejemplo:

Algebra Relacional:

RxS

SQL

SELECT * FROM R,S;

--Siendo R y S dos tablas.

NaturalJoin

Este operador se utiliza cuando se tiene la necesidad de unir relaciones vinculando sólo las tuplas que coinciden de alguna manera. `NaturalJoin` une sólo los pares de tuplas de R y S que sean comunes. Más precisamente una tupla r de R y una tupla s de S se emparejan correctamente si y sólo si r y s coinciden en cada uno de los valores de los atributos comunes, el resultado de la vinculación es una tupla, llamada **joined tuple**. Entonces, al realizar `NaturalJoin` se obtiene una relación con los atributos de ambas relaciones y se obtiene combinando las tuplas de ambas relaciones que tengan el mismo valor en los atributos comunes.

Notación en Álgebra Relacional

Para denotar `NaturalJoin` se utiliza la siguiente simbología: $R \rhd\lhd S$.

Ejemplo:

Algebra relacional:

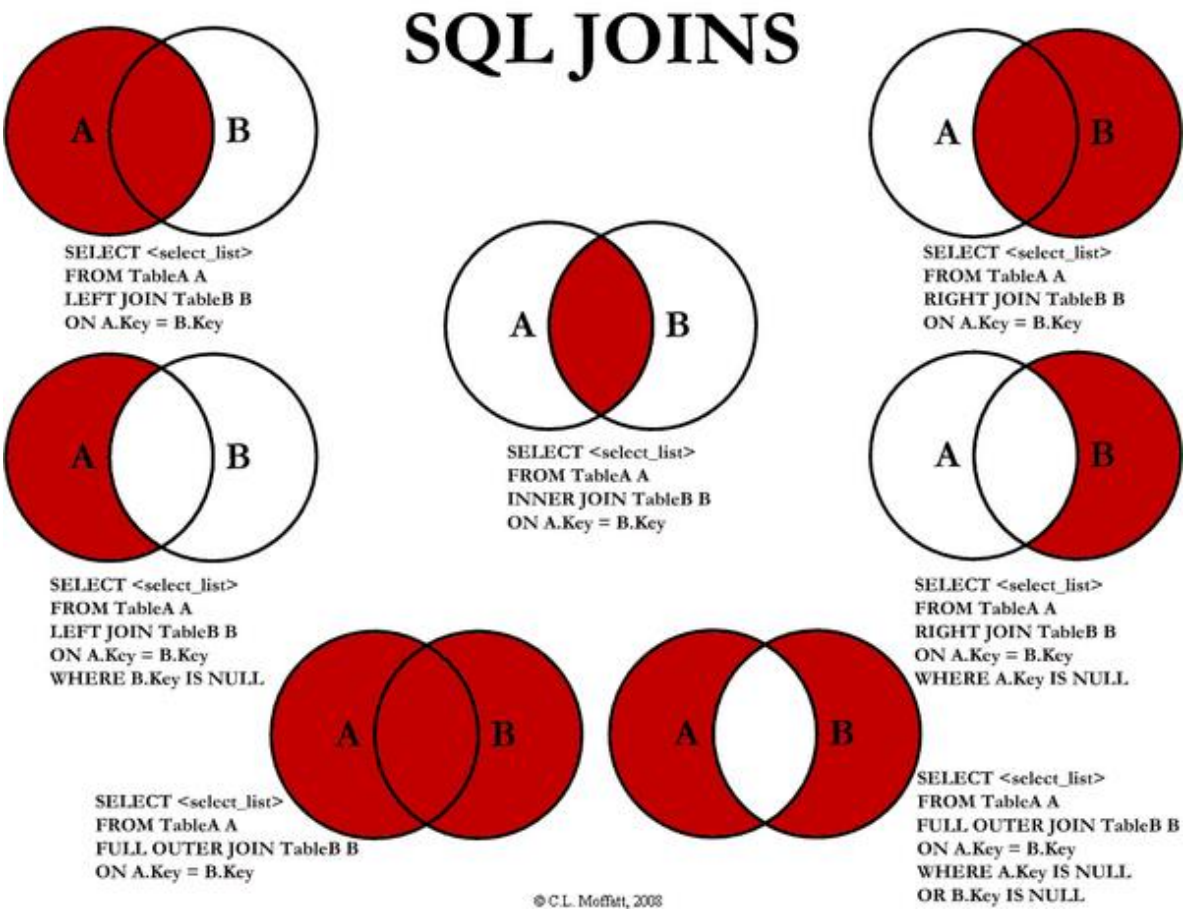
USUARIO ⊳⊲ TIPO_USUARIO

SQL:

```
SELECT *
FROM TBL_USUARIOS
INNER JOIN TBL_TIPO_USUARIO
ON (TBL_USUARIOS.CODIGO_TIPO_USUARIO =
TBL_TIPO_USUARIO.CODIGO_TIPO_USUARIO)
```

Más información sobre algebra relacional:
http://csrg.inf.utfsm.cl/~jfuentes/_build/html/lectures/week1/lecture3.html
http://www.unirioja.es/cu/arjaime/Temas/04.Anexo.pdf

Resumen grafico sobre Joins



Cruces (Joins) Convencionales (INNER JOIN, NATURAL JOIN)

A continuación se presenta tres scripts cuyo resultado es el mismo. El resultado y el rendimiento a nivel de Oracle es el indiferente. Debe utilizar un INNER JOIN cuando este seguro que la llave primaria y la llave foránea no permiten campos nulos, de no ser así puede existir perdida de información.

Utilizando INNER JOIN

```
SELECT  A.NOMBRE_USUARIO,
        A.CORREO,
        B.TIPO_USUARIO,
        B.DESCRIPCION
FROM TBL_USUARIOS A
INNER JOIN TBL_TIPO_USUARIO B
ON (A.CODIGO_TIPO_USUARIO = B.CODIGO_TIPO_USUARIO);
```

Utilizando NATURAL JOIN: En el caso de utilizar este tipo de JOIN los campos deben llamarse igual en ambas tablas.

```
SELECT  A.NOMBRE_USUARIO,
        A.CORREO,
        B.TIPO_USUARIO,
        B.DESCRIPCION
FROM TBL_USUARIOS A
NATURAL JOIN TBL_TIPO_USUARIO B;
```

Utilizando un producto cartesiano con un criterio en la cláusula WHERE

```
SELECT A.NOMBRE_USUARIO,
        A.CORREO,
        B.TIPO_USUARIO,
        B.DESCRIPCION
FROM  tbl_usuarios A,
      tbl_tipo_usuario B
WHERE A.codigo_tipo_usuario = B.codigo_tipo_usuario;
```

Usuario de pruebas HR (Human Resources)

Oracle viene por defecto con un esquema de pruebas llamado HR, el cual contiene un conjunto de tablas relacionadas a un sistema de Recursos Humanos, dicho esquema por defecto esta bloqueado por lo cual se define a continuacion el script para desbloquearlo y asignar un password:

Sustituya la palabra PASSWORD por la constraseña de su preferencia:

```
ALTER USER HR IDENTIFIED BY "PASSWORD" ACCOUNT UNLOCK;
```

Ejemplo de cruces de la tabla de empleados y trabajos:

INNER JOIN:

```
SELECT first_name ||' '|| last_name  as name,
       email,
       job_title
FROM employees A
INNER JOIN jobs B
ON (A.job_id = B.job_id);
```

NATURAL JOIN:

```
SELECT first_name ||' '|| last_name  as name,
       email,
       job_title
FROM employees A
NATURAL JOIN jobs B;
```

Producto cartesiano aplicando la cláusula WHERE

```
SELECT first_name ||' '|| last_name  as name,
       email,
       job_title
FROM employees A,
     jobs B
WHERE A.job_id = B.job_id;
```

LEFT JOIN e RIGHT JOIN

Con la sentencia INNER JOIN existe perdida de información cuando uno de los campos que especifica en la sentencia ON o el campo que hace la relación (llave foránea) es nulo o tiene un valor que no existe (sin integridad referencial). Los registros que no coinciden no son tomados en cuenta al momento de utilizar un INNER JOIN, esto en muchas ocasiones puede ser catastrófico ya que puede haber pérdida de información valiosa.

Para evitar este problema se utiliza la sentencia LEFT JOIN o RIGHT JOIN de tal forma que se incluye los registros de la tabla de la derecha o de la izquierda (dependiendo del join) de tal forma que no exista perdida de registros para los que no coincidan con la llave primaria y la llave foránea.

Ejemplo:

Intente probar este script con un INNER JOIN y luego con un LEFT JOIN y podrá ver que existe perdida de información utilizando el INNER JOIN mientras que con el otro se conserva la misma cantidad de registros.

```
SELECT A.first_name||' '||A.last_name as empleado,
      b.first_name||' '||b.last_name as jefe
FROM EMPLOYEES A --EMPLEADOS
LEFT JOIN EMPLOYEES B --JEFES
ON (A.MANAGER_ID = B.EMPLOYEE_ID);
```

Cruce de tres tablas utilizando LEFT JOIN

```
SELECT A.first_name||' '||A.last_name as empleado,
      B.first_name||' '||B.last_name as jefe,
      C.first_name||' '||C.last_name as super_jefe
FROM EMPLOYEES A --EMPLEADOS
LEFT JOIN EMPLOYEES B --JEFES
ON (A.MANAGER_ID = B.EMPLOYEE_ID)
LEFT JOIN EMPLOYEES C --SUPER JEFES
ON (B.MANAGER_ID = C.EMPLOYEE_ID);
```

Recomendaciones al realizar consultas cruzadas:

1. Consultar cada tabla por aparte para ver la información.
2. Al hacer el cruce, verificar si hay pérdida o duplicidad de información
3. La cantidad de registros resultantes debe coincidir con la cantidad de registros de la tabla principal.

Funciones de agregación

Las funciones de agregación en SQL nos permiten efectuar operaciones sobre un conjunto de resultados, pero devolviendo un único valor agregado para todos ellos. Es decir, nos permiten obtener medias, máximos, etc... sobre un conjunto de valores.

Las funciones de agregación básicas que soportan todos los gestores de datos son las siguientes:

- COUNT: devuelve el número total de filas seleccionadas por la consulta.
- MIN: devuelve el valor mínimo del campo que especifiquemos.
- MAX: devuelve el valor máximo del campo que especifiquemos.
- SUM: suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- AVG: devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

Las funciones anteriores son las básicas en SQL, pero cada sistema gestor de bases de datos relacionales ofrece su propio conjunto, más amplio, con otras funciones de agregación particulares. Puedes consultar las que ofrecen SQL Server, Oracle y MySQL.

Todas estas funciones se aplican a una sola columna, que especificaremos entre paréntesis, excepto la función COUNT, que se puede aplicar a una columna o indicar un "*". La diferencia entre poner el nombre de una columna o un "*", es que en el primer caso no cuenta los valores nulos para dicha columna, y en el segundo sí.

Ejemplos de agrupación:

http://www.campusmvp.es/recursos/post/Fundamentos-de-SQL-Agrupaciones-y-funciones-de-agregacion.aspx

http://www.mundoracle.com/funciones-de-grupo.html?Pg=sql_plsql_4.htm

Agregacion-->Resumir

Jefes

Gerentes

Directores

Analistas

Segregacion-->Detalle

Clientes

Usuarios de soporte

Programadores

Funciones especiales para cadenas

Las funciones de manejo de caracteres alfanuméricos aceptan argumentos de tipo caracter y retornan caracteres o valores numéricos.

Las siguientes son algunas de las funciones que ofrece Oracle para trabajar con cadenas de caracteres:

chr(x): retorna un caracter equivalente al código enviado como argumento "x". Ejemplo:

```
select chr(65) from dual;-- retorna 'A'.
select chr(100) from dual;-- retorna 'd'.
```

- concat(cadena1,cadena2): concatena dos cadenas de caracteres; es equivalente al operador ||. Ejemplo:

```
select concat('Buenas',' tardes') from dual;--retorna
'Buenas tardes'.
```

- initcap(cadena): retorna la cadena enviada como argumento con la primera letra (letra capital) de cada palabra en mayúscula. Ejemplo:

```
select initcap('buenas tardes alumno') from dual;--retorna
'Buenas Tardes Alumno'.
```

- lower(cadena): retorna la cadena enviada como argumento en minúsculas. "lower" significa reducir en inglés. Ejemplo:

```
select lower('Buenas tardes ALUMNO') from dual;--retorna
"buenas tardes alumno".
```

- upper(cadena): retorna la cadena con todos los caracteres en mayúsculas. Ejemplo:

```
select upper('www.oracle.com') from dual;-- 'WWW.ORACLE.COM'
```

- lpad(cadena,longitud,cadenarelleno): retorna la cantidad de caracteres especificados por el argumento "longitud", de la cadena enviada como primer argumento (comenzando desde el primer caracter); si "longitud" es mayor que el tamaño de la cadena enviada, rellena los espacios restantes con la cadena enviada como tercer argumento (en caso de omitir el tercer argumento rellena con espacios); el relleno comienza desde la izquierda. Ejemplos:

```
 select lpad('alumno',10,'xyz') from dual;-- retorna
'xyzxalumno'

 select lpad('alumno',4,'xyz') from dual;-- retorna 'alum'
```

- rpad(cadena,longitud,cadenarelleno): retorna la cantidad de caracteres especificados por el argumento "longitud", de la cadena enviada como primer argumento (comenzando desde el primer caracter); si "longitud" es mayor que el tamaño de la cadena enviada, rellena los espacios restantes con la cadena enviada como tercer argumento (en caso de omitir el tercer argumento rellena con espacios); el relleno comienza desde la derecha (último caracter). Ejemplos:

```
 select rpad('alumno',10,'xyz') from dual;-- retorna
'alumnoxyzx'

 select rpad('alumno',4,'xyz') from dual;-- retorna 'alum'
```

- ltrim(cadena1,cadena2): borra todas las ocurrencias de "cadena2" en "cadena1", si se encuentran al comienzo; si se omite el segundo argumento, se eliminan los espacios. Ejemplo:

```
select ltrim('la casa de la cuadra','la') from dual;-- ' casa
de la cuadra'

 select ltrim(' es la casa de la cuadra','la') from dual;--
no elimina ningún caracter

 select ltrim('  la casa') from dual;-- 'la casa'
```

- rtrim(cadena1,cadena2): borra todas las ocurrencias de "cadena2" en "cadena1", si se encuentran por la derecha (al final de la cadena); si se omite el segundo argumento, se borran los espacios. Ejemplo:

```
select rtrim('la casa lila','la') from dual;-- 'la casa li'

 select rtrim('la casa lila ','la') from dual;-- no borra
ningún caracter

 select rtrim('la casa lila    ') from dual; --'la casa lila'
```

- trim(cadena): retorna la cadena con los espacios de la izquierda y derecha eliminados. "Trim" significa recortar. Ejemplo:

```
select trim('   oracle     ') from dual;--'oracle'
```

- replace(cadena,subcade1,subcade2): retorna la cadena con todas las ocurrencias de la subcadena de reemplazo (subcade2) por la subcadena a reemplazar (subcae1). Ejemplo:

```
select replace('xxx.oracle.com','x','w') from dual;
retorna "www.oracle.com'.
```

- substr(cadena,inicio,longitud): devuelve una parte de la cadena especificada como primer argumento, empezando desde la posición especificada por el segundo argumento y de tantos caracteres de longitud como indica el tercer argumento. Ejemplo:

```
select substr('www.oracle.com',1,10) from dual;--
'www.oracle'
select substr('www.oracle.com',5,6) from dual;-- 'oracle'
```

- length(cadena): retorna la longitud de la cadena enviada como argumento. "lenght" significa longitud en inglés. Ejemplo:

```
select length('www.oracle.com') from dual;-- devuelve 14.
```

- instr (cadena,subcadena): devuelve la posición de comienzo (de la primera ocurrencia) de la subcadena especificada en la cadena enviada como primer argumento. Si no la encuentra retorna 0. Ejemplos:

```
 select instr('Jorge Luis Borges','or') from dual;-- 2
 select instr('Jorge Luis Borges','ar') from dual;-- 0, no se
encuentra
```

Fuente:

http://www.oracleya.com.ar/temarios/descripcion.php?cod=179&punto=21

https://sirchokolate.wordpress.com/2010/08/28/funciones-de-manipulacion-de-cadenas-oracle/

Subconsultas

//Anotaciones de clase

SELECT X
FROM Y
WHERE Z;

X: (1 solo campo, 1 solo registro).
Y: (Hay que tener cuidado al momento de realizar cruces)
Z:

- in o not in
  Un solo campo y varios registros
  Por ejemplo:
  where edad in (SELECT edad FROM tabla_x)
- =,<,>,>=,<=
  Un solo campo, un solo registro
  Por ejemplo:
  edad = (SELECT edad FROM tablaX WHERE condición que retorna un único registro)

Usos de create select e insert select:

- ETL
  - Tablas intermedias
- Backups
- Analisis aislados en nuevas tablas

http://mioracle.blogspot.com/2008/02/subconsultas.html

http://www.mundoracle.com/subconsultas.html?Pg=sql_plsql_6.htm

Operadores UNION, UNION ALL, MINUS, INTESECT

Podemos combinar multitud de consultas con los operadores UNION, UNION ALL, INTERSECT, y MINUS. Todo este conjunto de operadores tienen la misma precedencia, por lo que si una consulta contiene varios de estos operadores Oracle los evaluará de izquierda a derecha siempre que no haya paréntesis que expecifiquen otro orden.

Para entender estos operadores, os mostraré un ejemplo de cada uno de ellos con un sencillo ejemplo, en el que tendremos 2 tablas, cada una de ellas con los datos de los clientes que compraron un articulo en un año determinado.

Ejemplo operador UNION

La siguiente consulta combina el resultado de 2 consultas gracias al operador UNION, el cual eliminna las filas duplicadas después de realizar la unión. La siguiente consulta nos devolverá los clientes únicos que han comprado tanto en el año 2010 como en el año 2011:

```
SELECT nombre, apellidos, dni FROM tabla_clientes_2010
  UNION
SELECT nombre, apellidos, dni FROM tabla_clientes_2011
```

Ejemplo operador UNION ALL

El operador UNION devuelve solo las filas distintas que aparecen en ambas tablas, mientras que el operador UNION ALL devuelve todas las filas, ya que no elimina las filas duplicadas. La siguiente consulta nos devolverá los clientes que han comprado tanto en el año 2010 como en el año 2011 (si un cliente compró en ambos años, aparecerá 2 veces):

```
SELECT nombre, apellidos, dni FROM tabla_clientes_2010
  UNION ALL
SELECT nombre, apellidos, dni FROM tabla_clientes_2011
```

Ejemplo operador INTERSECT

El operador INTERSECT devuelve solo las filas que aparecen en ambas consultas, es decir los clientes que fueron clientes en el 2011 y en el 2010:

```
SELECT nombre, apellidos, dni FROM tabla_clientes_2010
INTERSECT
SELECT nombre, apellidos, dni FROM tabla_clientes_2011
```

Ejemplo operador MINUS

La siguiente consulta combina el resultado de las 2 queries, y devuelve solo los elementos de la primera consulta que no se encuentren en la segunda, es decir todos aquellos clientes que lo fueron en el 2010 pero no lo fueron en el 2011:

```
SELECT nombre, apellidos, dni FROM tabla_clientes_2010
MINUS
SELECT nombre, apellidos, dni FROM tabla_clientes_2011
```

Fuente: http://deckerix.com/blog/los-operadores-union-all-intersect-y-minus-de-oracle/