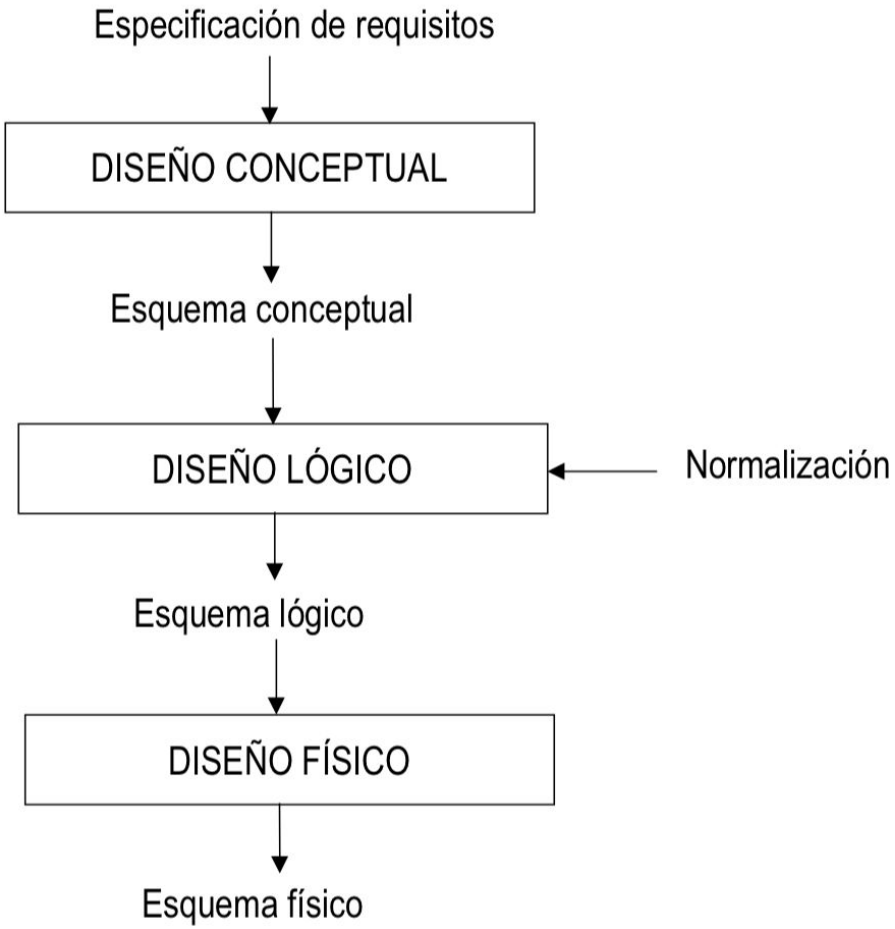
SQL

Data Definition Language

2. Metodología de diseño de bases de datos



Lenguajes de Query relacional

- Uno de los principales beneficios del modelo relacional: soporta simples y potentes queries de datos.
- Dos sublenguajes:
- DDL – Data Definition Language (Lenguaje de Definición de Datos)
  - Define y modifica el esquema
- DML – Data Manipulation Language (Lenguaje de Manipulación de Datos)
  - Los queries pueden ser escritos intuitivamente
- El DBMS es responsable de hacer una evaluación eficiente.
  - La clave: semántica precisa para queries relacionales.
  - Permitir al optimizador reordenar o cambiar operaciones, y asegurarse que el resultado no cambie.

Contenido

- DDL
  - Create (crear objetos dentro de la BD tablas, procedure, triggers, views, functions, index)
  - Alter (modificar objetos dentro de la BD)
  - Drop (eliminar objetos dentro de la BD)

  - Objetos no son datos

CREATE TABLE

- CREATE TABLE *nombre_de_tabla*
  ({ *nombre_de_columna tipo_de_dato* [DEFAULT *expr_default*] [*restriccion_de_column* [, ...]] | *restriccion_de_tabla*} [, ...])

- Tipos de Datos incluyen:
  char (n) – cadena de caracteres de tamaño fijo
  varchar2(n) – cadena de caracteres de tamaño variable (ahorra espacio, Guarda 'SMITH' no 'SMITH ‘)
  integer, numeric, float, double precision
  date, time, timestamp, …
  serial – ID unico para indexar y referencia

Create Table (con/restricciones de tabla)

- CREATE TABLE *table_name*
  ({ *column_name data_type* [DEFAULT *default_expr*] [
  column_constraint [, ...]] | *table_constraint* } [, ...])

Restricciones de Tabla:

[CONSTRAINT *nombe_restriccion*]

{ UNIQUE (*nombre_de_columna* [, ...]) |

PRIMARY KEY (*nombre_de_columna* [, ...]) |

CHECK (*expresion*) |

FOREIGN KEY (*nombre_de_columna* [, ...]) REFERENCES *tablaref*[(
refcolumn [, ...])] [ON DELETE *accion*] [ON UPDATE *accion*
] }

Aqui, *las expresiones, llaves, etc pueden incluir múltiples columnas*

Constraint(restricciones)

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column when no value is specified
- INDEX - Used to create and retrieve data from the database very quickly

Create Table (Ejemplos)

```
CREATE TABLE films (
    codigo        CHAR(5) PRIMARY KEY,
    titulo        VARCHAR2(40) not null,
    did           DECIMAL(3),
    fecha_prod    DATE ,
    genero        VARCHAR(10),
CONSTRAINT production UNIQUE(fecha_prod),
FOREIGN KEY (did) REFERENCES distribuidores(did)
);

CREATE TABLE distribuidores(
    did     DECIMAL(3) PRIMARY KEY,
    nombre VARCHAR2(40),
    CONSTRAINT cons1 CHECK (did > 100 AND nombre <> ' ')
);
```

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

Para estas tablas:

Reservacion

sid	bid	dia
22	101	10/10/96
95	103	11/12/96

Marineros

sid	snombre	rating	edad
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

Botes

bid	bnombre	color
101	Interlake	azul
102	Interlake	rojo
103	Clipper	verde
104	Marine	rojo

Esquemas de Ejemplo

```
CREATE TABLE Marineros
( sid INTEGER PRIMARY KEY,
    snombre CHAR(20),
  rating INTEGER,
  edad REAL);

CREATE TABLE Botes
( bid INTEGER PRIMARY KEY,
    bnombre CHAR (20),
  color CHAR(10));

 CREATE TABLE Reservacion
( sid INTEGER,
    bid INTEGER,
  dia DATE,
    PRIMARY KEY (sid, bid, dia),
  FOREIGN KEY (sid) REFERENCES Marineros,
    FOREIGN KEY (bid) REFERENCES Botes);
```

AUTO_INCREMENT

```
CREATE TABLE Persons (
    ID int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

Drop

- Instrucción para el borrado de un elemento:

  DROP <elemento> <nombre>
- No tiene marcha atrás
- Ejemplos:

  DROP TABLE films;

Alter(se lo realiza cuando ya se tiene datos)

- Instrucción para la modificación de un elemento.

ALTER <elemento_p> <nombre>

{ADD | MODIFY| DROP} <elemento_s [<definicion>]

Ejemplos: ALTER TABLE Botes

ADD column Edad REAL;

ALTER TABLE Marineros

MODIFY rating REAL;

ALTER TABLE Distribuidores

DROP CONSTRAINT cons1;

ALTER TABLE Films

DROP COLUMN genero;

ALTER TABLE Botes

ADD CONSTRAINT con CHECK (edad<100);

Mostrar columnas y describirlas

- SHOW COLUMNS FROM <tbl_name>;
- DESCRIBE <tbl_name>;

DML

Manipulación de datos

- insert (crear/grabar datos en una tabla)
- update (actualizar datos existentes en una tabla)
- delete (borrar datos existentes)
- **select (leer datos de una tabla ….. Proyeccion, selección, join, producto cruz, diferencia, etc)**

Operaciones CRUD. Create, Retrieve, Update, Delete

INSERT

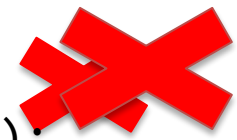
```
INSERT INTO    nombreTabla[(lista_columnas)]
VALUES         (lista_valores)
```

- El número de ítems debe ser el mismo.
- Correspondencia directa entre nombre_columnas y lista_valores.
- Compatible en tipos de datos.
- Lista_columnas opcional:
  - si se omite se asume el orden usado para crear la tabla.
  - Si se especifica y se omiten columnas, las columnas debieron de ser declaradas como NULL o un valor DEFAULT.

INSERT

```
CREATE TABLE `Tarjeta` (
  `idTarjeta` int(11) NOT NULL,
  `nombre_tarjeta` varchar(45) NOT NULL,
  `numero_tarjeta` varchar(45) NOT NULL,
  `banco_emisor` varchar(45) NOT NULL
DEFAULT 'ABC',
  `fecha_expiracion` date NOT NULL,
  `debito` tinyint(1) DEFAULT NULL,
  `credito` tinyint(1) DEFAULT NULL,
  `tipo` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`idTarjeta`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO Tarjeta
(idTarjeta,nombre_tarjeta,numero_tarjeta,fecha_expiracion)
VALUES (2,'Vanessa E','00112233445567','2019-07-01');
(2,'Vanessa E','00112233445567','2019-07-01',NULL,NULL,NULL);
```



INSERT

```
CREATE TABLE `Tarjeta` (
  `idTarjeta` int(11) NOT NULL,
  `nombre_tarjeta` varchar(45) NOT NULL,
  `numero_tarjeta` varchar(45) NOT NULL,
  `banco_emisor` varchar(45) NOT NULL
DEFAULT 'ABC',
  `fecha_expiracion` date NOT NULL,
  `debito` tinyint(1) DEFAULT NULL,
  `credito` tinyint(1) DEFAULT NULL,
  `tipo` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`idTarjeta`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO Tarjeta
VALUES
(3,'Vanessa E','PACIFICO','00112233445567','2019-07-01',NULL,NULL,NULL);
```

INSERT

```
CREATE TABLE `Tarjeta` (
  `idTarjeta` int(11) NOT NULL,
  `nombre_tarjeta` varchar(45) NOT NULL,
  `numero_tarjeta` varchar(45) NOT NULL,
  `banco_emisor` varchar(45) NOT NULL
DEFAULT 'ABC',
  `fecha_expiracion` date NOT NULL,
  `debito` tinyint(1) DEFAULT NULL,
  `credito` tinyint(1) DEFAULT NULL,
  `tipo` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`idTarjeta`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO
Tarjeta (idTarjeta,nombre_tarjeta,numero_tarjeta,fecha_expiracion)
VALUES (2,'Vanessa E','00112233445567','2019-07-01');
```

INSERT

```
INSERT INTO    nombreTabla[(lista_columnas)]
SELECT
```

- lista_columna es definida de igual manera.
- SELECT corresponde a cualquier select válido
- El resultado del SELECT debe de ser compatible con las columnas de la tabla.

UPDATE (modificar un dato ya existente)

```
UPDATE      nombreTabla
SET         columna1 = valor1 [,columna2 = valor2]
[WHERE          condicion]
```

- Nuevos valores deben ser compatibles (TD).
- WHERE opcional:
  - Si se omite: Se modifican TODAS las filas.
  - Si se especifica: Se modifican las filas que cumplen con la condición.

UPDATE

- Modificar todas las filas:

```
UPDATE CLIENTES
SET LIMITECREDITO= LIMITECREDITO*1.12;
```

UPDATE

- Modificar filas específicas:

```
UPDATE CLIENTES
SET LIMITECREDITO= LIMITECREDITO*1.12
WHERE REPCLIE=102;
```

UPDATE

- Modificar múltiples columnas:

```
UPDATE CLIENTES
SET LIMITECREDITO= LIMITECREDITO*1.12,
REPCLIE=101
WHERE NUMCLIE=2001;
```

DELETE

```
DELETE FROM nombreTabla
WHERE       condicion
```

- Borra las filas de una tabla específica.
- WHERE opcional:
  - Si se omite: se borran todas las filas.
  - Si se especifica: borra las filas que cumplen con la condición.

DELETE

```
DELETE FROM CLIENTES
```

```
DELETE FROM CLIENTES
WHERE NUMCLIE=2002
```

SELECT

- Recupera y muestra datos de una o más tablas.
- Ejecuta el equivalente de las operaciones de *proyección*, *selección* y *join* del algebra relacional.

SELECT

- Forma general:

```
SELECT      [DISTINCT | ALL] {* | [columnExpression [AS newName]] [, . . . ]}
FROM        TableName [alias] [, . . . ]
[WHERE      condition]
[GROUP BY   columnList] [HAVING condition]
[ORDER BY   columnList]
```

FROM	especifica la tabla o tablas que van a ser usadas
WHERE	filtra las filas dependiendo de alguna condición
GROUP BY	forma grupos de filas con el mismo valor en la columna
HAVING	Filtra los grupos dependiendo de alguna condición
SELECT	especifica las columnas que aparecerán en el resultado
ORDER BY	Especifica el orden del resultado

SELECT

I. Recuperar todas las filas y todas las columnas.

```
SELECT   [*] [lista columnas]
FROM     nombreTabla
```

SELECT

Cliente

NUMCLIE	EMPRESA	REPCLIE	LIMITECREDITO
2000	ACDC E.P.	105	65000
2001	ABCD E.P.	102	70000
2002	ZXWY E.P.	103	50000
2003	PEPITO E.P.	105	65000
2101	JONES MFG.	106	65000
2102	FISRT CORP.	101	65000
2103	ACME MFG.	105	50000

SELECT

II. Recuperar todas las filas y columnas específicas (π).

```
SELECT   nombre_columna1, nombre_columna2,...
FROM     nombreTabla
```

SELECT

```
SELECT NUMCLIE, EMPRESA FROM Cliente
```

Cliente

NUMCLIE	EMPRESA	REPCLIE	LIMITECREDITO
2000	ACDC E.P.	105	65000
2001	ABCD E.P.	102	70000
2002	ZXWY E.P.	103	50000
2003	PEPITO E.P.	105	65000
2101	JONES MFG.	106	65000
2102	FISRT CORP.	101	65000
2103	ACME MFG.	105	50000

SELECT

```
SELECT NUMCLIE, EMPRESA FROM Cliente
```

Resultado:

NUMCLIE	EMPRESA
2000	ACDC E.P.
2001	ABCD E.P.
2002	ZXWY E.P.
2003	PEPITO E.P.
2101	JONES MFG.
2102	FISRT CORP.
2103	ACME MFG.

SELECT

III. Uso de DISTINCT

```
SELECT DISTINCT   nombre_columna1,
nombre_columna2,...
FROM              nombreTabla
```

SELECT

```
SELECT LIMITECREDITO
FROM Cliente
```

Resultado:

LIMITECREDITO
65000
70000
50000
65000
65000
65000
50000

```
SELECT DISTINCT LIMITECREDITO
FROM Cliente
```

Resultado:

LIMITECREDITO
65000
70000
50000

PRODUCTO CRUZ - FROM

- El producto cruz de dos relaciones en SQL se obtiene simplemente colocando más de una tabla en la cláusula FROM.

```
SELECT * FROM PROFESORES, DEPARTAMENTOS
```

- El producto cartesiano raramente se utiliza en la práctica, pero es interesante conocerlo para diferenciarlo de la operación de combinación (join).

SELECT

IV. Campos derivados de cálculos o expresiones aritméticas utilizando una o más columnas.

– Ej: Mostrar por cada cliente, el 10% del límite de crédito

```
SELECT NUMCLIE, LIMITECREDITO, LIMITECREDITO*0.10 AS porcentaje
FROM Cliente
```

Renombra la columna

Resultado:

NUMCLIE	LIMITECREDITO	porcentaje
2000	65000	6500
2001	70000	7000
2002	50000	5000
2003	65000	6500
2101	65000	6500
2102	65000	6500
2103	50000	5000

SELECT - WHERE

Selección de filas (σ)

```
SELECT    nombre_columna1, nombre_columna2,...
FROM      nombreTabla
WHERE     condicion
```

Comparación	Compara el valor de una expresión con el valor de otra expresión.
Rangos	Verifica si el valor de una expresión se encuentra dentro de un rango específico de valores.
Set membership	Verifica si el valor de una expresión equivale a uno dentro del conjunto de valores.
Pattern match	Verifica si un string match con un patrón específico.
Null	Verifica si una columna tiene valores *null.*

SELECT - WHERE

I. Condición de comparación (σ)

```
SELECT    nombre_columna1, nombre_columna2,...
FROM      nombreTabla
WHERE     condicion
```

Se pueden utilizar operadores simples de comparación: < , = , > , < , > , >= , <=

SELECT - WHERE

I. Condición de comparación
   - Ej. Seleccionar los clientes con límite de crédito menor a 65000.

```
SELECT NUMCLIE, LIMITECREDITO
FROM Cliente
WHERE LIMITECREDITO < 65000
```

Resultado:

NUMCLIE	LIMITECREDITO
2002	50000
2103	50000

SELECT - WHERE

II. Condición de comparación compuesta (σ)

```
SELECT    nombre_columna1, nombre_columna2,...
FROM      nombreTabla
WHERE     condicion1 AND | OR condicion2
```

Se pueden utilizar operadores simples de comparación: < , = , > , < , > , >= , <=

SELECT - WHERE

II. Condición de comparación compuesta
   - Ej. Seleccionar los clientes con límite de crédito menor a 65000 y con código de representantes igual a 105.

```
SELECT NUMCLIE, REPCLIE, LIMITECREDITO
FROM Cliente
WHERE LIMITECREDITO > 50000 AND REPCLIE = 105 ;
```

Resultado:

NUMCLIE	REPCLIE	LIMITECREDITO
2000	105	65000
2003	105	65000

SELECT - WHERE

III. Condición por rango

-[illegible] se e[illegible]

```
SELECT      nombre_columna1, nombre_columna2,...
FROM        nombreTabla
WHERE       campo (NOT) BETWEEN valor1 AND valor2 ;
```

```
SELECT *
FROM Cliente
WHERE LIMITECREDITO BETWEEN 60000 AND 70000 ;
```

NUMCLIE	EMPRESA	REPCLIE	LIMITECREDITO
2000	ACDC E.P.	105	65000
2001	ABCD E.P.	102	70000
2003	PEPITO E.P.	105	65000
2101	JONES MFG.	106	65000
2102	FISRT CORP.	101	65000

SELECT - WHERE

IV. Set membership

```
SELECT    nombre_columna1, nombre_columna2,...
FROM      nombreTabla
WHERE     campo (NOT) IN lista_de_valores
```

SELECT - WHERE

IV. Set membership

- Ej: Muestre los clientes que tengan a los representantes 102 y 105.

```
SELECT *
FROM Cliente
WHERE REPCLIE IN (102,105);
```

Resultado:

NUMCLIE	EMPRESA	REPCLIE	LIMITECREDITO
2000	ACDC E.P.	105	65000
2001	ABCD E.P.	102	70000
2003	PEPITO E.P.	105	65000
2103	ACME MFG.	105	50000

SELECT - WHERE

V. Búsqueda de patrones en cadenas de caracteres.

```
SELECT    nombre_columna1, nombre_columna2,...
FROM      nombreTabla
WHERE     campo (NOT) LIKE patron
```

- % (wildcard) alguna secuencia de cero o muchos caracteres.
- _ (underscore) representa un simple caracter (cualquier).

LIKE ‘A%’

LIKE ‘A_ _ _’

LIKE ‘%A’

LIKE ‘%ABC%’

NOT LIKE

SELECT - WHERE

V. Búsqueda de patrones en cadenas de caracteres.

– Ej: Muestre los clientes cuyo nombre de empresa incluya "E.P."

```
SELECT *
FROM Cliente
WHERE EMPRESA LIKE '%E.P.%';
```

Resultado:

NUMCLIE	EMPRESA	REPCLIE	LIMITECREDITO
2000	ACDC E.P.	105	65000
2001	ABCD E.P.	102	70000
2002	ZXWY E.P.	103	50000
2003	PEPITO E.P.	105	65000

EJERCICIOS – LIKE

- Muestre la descripción de los productos que empiezan con la letra R.
- Muestre el idproducto que tenga 4 caracteres.
- Muestre los nombres de las empresas en la tabla Clientes que tengan de segundo caracter la letra A.
- Muestre los nombres de las empresas que tengan las letras ES.

SELECT - WHERE

VI. Condición NULL

```
SELECT    nombre_columna1, nombre_columna2,...
FROM      nombreTabla
WHERE     campo IS (NOT) NULL
```

Los valores en una tupla a veces son *desconocidos.*
No es posible hacer una comparación con una cadena de caracteres, por este motivo es necesario comparar con la expresión NULL.
Ej: No se ingresó un número de teléfono porque era opcional, por defecto SQL agrega NULL a ese registro.

Paso 1 – Producto Cruz

E.eid	E.nombre	E.login	E.edad	E.prom	R.eid	R.cid	R.nota
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

```
SELECT E.nombre, R.cid
 FROM Estudiantes E, Registros R
 WHERE E.eid=R.eid AND R.nota='B'
```

Paso 2. descartar tuplas que fallan

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

SELECT E.nombre, R.cid
FROM Estudiantes E, Registros R
WHERE E.eid=R.eid AND R.nota='B'

Paso 3. Descartar Columnas no requeridas

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

```
SELECT S.name, E.cid
 FROM Students S, Enrolled E
 WHERE S.sid=E.sid AND E.grade='B'
```

SELECT - EJERCICIOS

- Muestre los pedidos que se hicieron en la oficina “Chicago”.
- Muestre los productos que compro el cliente “HOLM LANDIS”.
- Muestre las oficinas donde el cliente “HOLM LANDIS” hizo pedidos.

SELECT – ORDER BY

- Ordenar por una sola columna

```
SELECT      nombre_columna1, nombre_columna2,...
FROM        nombreTabla
ORDER BY    nombre_columna1 [DESC |ASC]
```

SELECT – ORDER BY

- Ordenar por varias columnas

```
SELECT      nombre_columna1, nombre_columna2,...
FROM        nombreTabla
ORDER BY    nombre_columna1,nombre_columna2 [DESC
|ASC]
```

SELECT - Funciones SQL

Funciones	Descripción
AVG	Calcula el valor medio de "n" ignorando los valores nulos.
COUNT	Cuenta el numero de veces que la expresión evalúa algún dato con valor no nulo. La opción "*" cuenta todas las filas seleccionadas.
MAX	Calcula el máximo.
MIN	Calcula el mínimo.
SUM	Obtiene la suma de los valores de la expresión.

- El resultado es un solo número(relación con una sola fila y una sola columna)
- Operadores de agregación *no se pueden utilizar en la clausula* WHERE

Ejemplos – Funciones SQL

- Muestre cual es el cliente que tiene mayor límite de crédito.
- Muestre cuantos clientes hay en total.
- Muestre el valor total entre los pedidos realizados por el cliente FRED LEWIS.
- Muestre el promedio de los pedidos por cliente.

SELECT – GROUP BY

- Agrupa los datos y muestra un resumen de los datos por grupo.

```
SELECT      nombre_columna1, nombre_columna2,...
FROM        nombreTabla
GROUP BY    nombre_columna1
ORDER BY    nombre_columna1,nombre_columna2 (DESC
|ASC)
```

SELECT – GROUP BY

- Muestre por cada representante el numero de clientes y el total de límite de crédito que maneja.

```
SELECT REPCLIE,NUMCLIE,LIMITECREDITO
FROM Cliente
GROUP BY REPCLIE
ORDER BY REPCLIE;
```

Resultado:

REPCLIE	NUMCLIE	LIMITECREDITO
101	2102	65000
102	2001	70000
103	2002	50000
105	2000	65000

SELECT – GROUP BY

- Muestre por cada representante el numero de clientes y el total de límite de crédito que maneja.

```
SELECT REPCLIE,COUNT(NUMCLIE) AS NUMERO_CLIENTES,
SUM(LIMITECREDITO) AS SUMA_CREDITO
FROM Cliente
GROUP BY REPCLIE
ORDER BY REPCLIE;
```

Resultado:

REPCLIE	NUMERO_CLIENTES	SUMA_CREDITO
101	1	65000
102	1	70000
103	1	50000
105	3	180000

SELECT - HAVING

- Especifica una condición de búsqueda para un grupo o agregado. HAVING solo se puede utilizar con la instrucción SELECT.
- Normalmente, HAVING se utiliza con una cláusula GROUP BY.

```
SELECT      nombre_columna1, nombre_columna2,...
FROM        nombreTabla
GROUP BY    nombre_columna1
HAVING      condición(nombre_columna1)
ORDER BY    nombre_columna1,nombre_columna2 (DESC
|ASC)
```

SELECT - HAVING

- Muestre los representantes que tienen más de un cliente y el total de límite de crédito que maneja.

```
SELECT REPCLIE, COUNT(NUMCLIE) AS NUMERO_CLIENTES,
SUM(LIMITECREDITO) AS SUMA_CREDITO
FROM Cliente
GROUP BY REPCLIE
HAVING NUMERO_CLIENTES > 1
ORDER BY REPCLIE;
```

REPCLIE	NUMERO_CLIENTES	SUMA_CREDITO
105	3	180000

Ejercicios

- Muestre los productos que cuesten mas de $700.
- Muestre los empleados que trabajan en la ciudad de Chicago o de de Atlanta.
- Muestre los empleados que NO trabajan en la Ciudad de Chicago.

Ejercicios GROUP BY

- Muestre el promedio del importe de los pedidos por cliente.
- Mostrar la cantidad de artículos vendidos por cada fábrica.
- Mostrar cuántos trabajadores hay en cada oficina.

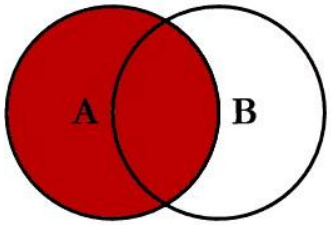
Ejercicios Having

- Para cada fábrica muestre el número de productos que produce, pero solo muestre aquellas que produzcan más de 5 productos.

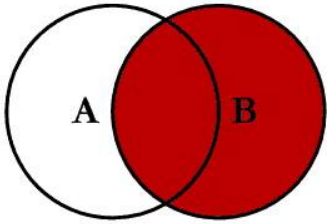
- Muestre el total (importe) de pedidos por cliente y, que el total sea menor a 10000.

- Muestre por el total de ventas por oficina y que sean mayores que 60000

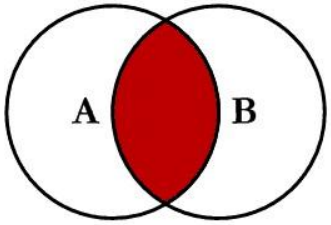
SQL JOINS



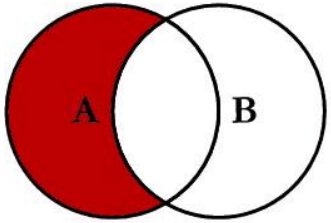

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



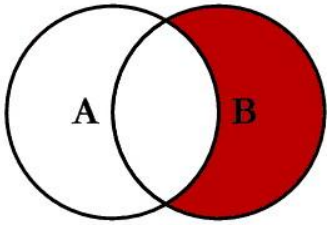

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



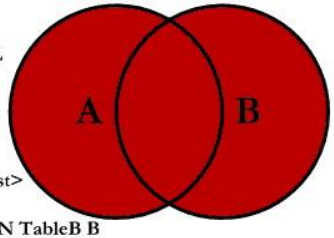

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



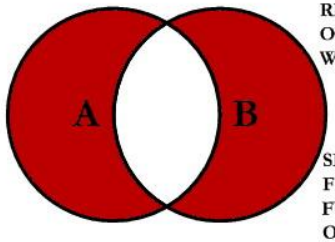

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

JOIN

- La sentencia **JOIN** combina los valores de la primera tabla con los valores de la segunda tabla dependiendo del campo por el cual se unen.

```
SELECT    nombre_columna1, nombre_columna2,...
FROM      tabla1
JOIN      tabla2
ON        tabla1.campo = tabla2.campo
```

JOIN

- Por cada pedido muestre el número de cliente y su razón social.

```
select NUMPEDIDO,NUMCLIE,EMPRESA
from Pedidos
join Clientes
on Clientes.NUMCLIE = Pedidos.CLIE;
```

NUMPEDIDO	NUMCLIE	EMPRESA
112968	2102	FISRT CORP.
112963	2103	ACME MFG.
112983	2103	ACME MFG.
112987	2103	ACME MFG.
113027	2103	ACME MFG.
112993	2106	FRED LEWIS

RIGHT JOIN

- La sentencia **RIGHT JOIN** combina los valores de la primera tabla con los valores de la segunda tabla. Siempre devolverá las filas de la segunda tabla, incluso aunque no cumplan la condición.

```
SELECT      nombre_columna1, nombre_columna2,...
FROM        tabla1
RIGHT JOIN  tabla2
ON          tabla1.campo = tabla2.campo
```

Right Join

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	4

Tabla "departamentos", con la clave primaria "dep"

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

```
SELECT nombre, apellido1, departamento
FROM personas
RIGHT JOIN departamentos
WHERE personas.dep = departamentos.dep
```

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
		COMERCIAL

LEFT JOIN

- La sentencia **LEFT JOIN** combina los valores de la primera tabla con los valores de la segunda tabla. Siempre devolverá las filas de la primera tabla, incluso aunque no cumplan la condición.

```
SELECT      nombre_columna1, nombre_columna2,...
FROM        tabla1
LEFT JOIN   tabla2
ON          tabla1.campo1 = tabla2.campo1
```

Left Join

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	4

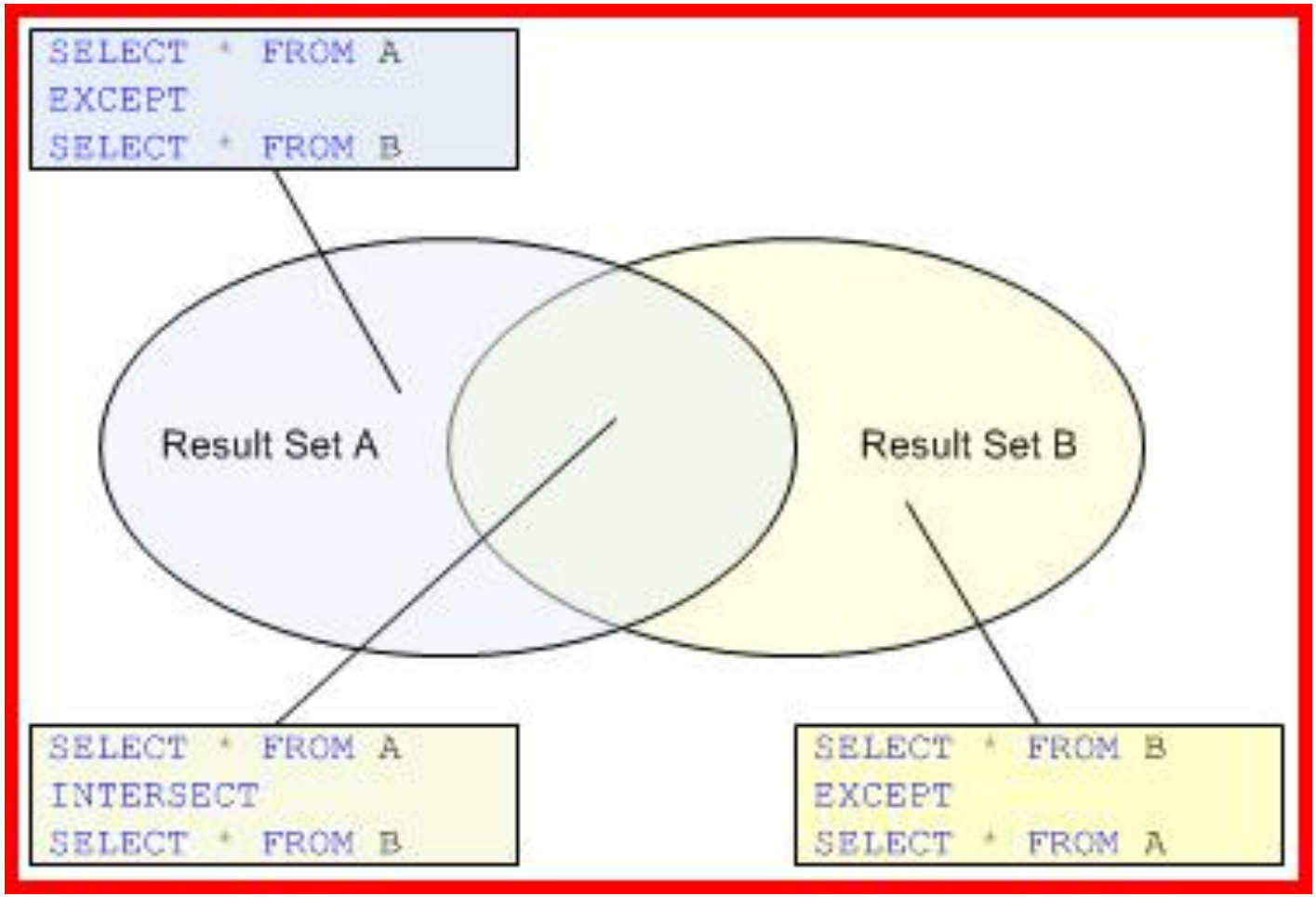
Tabla "departamentos", con la clave primaria "dep"

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

```
SELECT nombre, apellido1, departamento
FROM personas
LEFT JOIN departamentos
WHERE personas.dep = departamentos.dep
```

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
PEDRO	RUIZ	

Operadores de conjunto



Operadores de conjunto

- INTERSECT
  - Devuelve la intersección entre dos o más conjuntos de resultados en uno. El conjunto obtenido como resultado de **INTERSECT** tiene la misma estructura que los conjuntos originales.
  - **SELECT** columna1 **FROM** tabla1 **INTERSECT**

    SELECT columna1 **FROM** tabla2

Operadores de conjunto

- EXCEPT
  - Devuelve la diferencia (resta) de dos o más conjuntos de resultados. El conjunto obtenido como resultado de **EXCEPT** tiene la misma estructura que los conjuntos originales.
  - **SELECT** columna1 **FROM** tabla1 **EXCEPT**

    SELECT columna1 **FROM** tabla2

SUBQUERY

- Se lo puede usar en la cláusula WHERE o HAVING
- También dentro de la cláusula SELECT y FROM
- Se lo usa con el INSERT, UPDATE y DELETE

SUBQUERY

- Tres tipos:

1. Retorna un valor escalar
2. Retorna una columna.
3. Retorna una tabla.

SUBQUERIES

- Muestre todos los pedidos que incluyan el producto "MANIVELA"

```
SELECT p.NUMPEDIDO, p.FECHAPEDIDO, p.CLIE, p.PRODUCTO, p.CANT,p.IMPORTE
FROM Pedidos p
WHERE p.PRODUCTO = 41003
```

```
(SELECT IDPRODUCTO
  FROM PRODUCTO
  WHERE DESCRIPCION='MANIVELA');
```

41003

SUBQUERY

- *Muestre todos los clientes cuyo límite de crédito es mayor que el promedio y muestre por cada cliente la diferencia entre el límite de crédito y el promedio.*

- Necesitamos saber en primer lugar cuál es el promedio de los límites de crédito.

SUBQUERY

```
SELECT NUMCLIE, EMPRESA, LIMITECREDITO
FROM CLIENTES
WHERE LIMITECREDITO > AVG(LIMITECREDITO)
```

Las funciones de agregación no se pueden utilizar en la cláusula WHERE!

SUBQUERY

```
SELECT AVG(LIMITECREDITO) FROM CLIENTES;
```

Retorna el promedio del límite de crédito de todos los clientes (un solo valor).

```
SELECT NUMCLIE, EMPRESA, REPCLIE, LIMITECREDITO
FROM clientes
WHERE LIMITECREDITO > (SELECT AVG(LIMITECREDITO)
                       FROM CLIENTES);
```

Primero se evalúa el query anidado.

42791.6667

Luego se verifica la condición y se seleccionan los campos.

SUBQUERY

```
SELECT NUMCLIE, EMPRESA, REPCLIE,
LIMITECREDITO, LIMITECREDITO-(SELECT
AVG(LIMITECREDITO) FROM CLIENTES) AS
DIFERENCIA
FROM CLIENTES
WHERE LIMITECREDITO > (SELECT
AVG(LIMITECREDITO) FROM CLIENTES);
```

SUBQUERY

```
SELECT NUMCLIE, EMPRESA, REPCLIE,
LIMITECREDITO, LIMITECREDITO-(SELECT
AVG(LIMITECREDITO) FROM CLIENTES) AS
DIFERENCIA
FROM CLIENTES
WHERE LIMITECREDITO > (SELECT
AVG(LIMITECREDITO) FROM CLIENTES);
```

El subquery retorna un valor por cada fila y luego es restado del límite de crédito para obtener la diferencia.

SUBQUERY

```
SELECT NUMCLIE, EMPRESA, REPCLIE,
LIMITECREDITO, LIMITECREDITO-(SELECT
AVG(LIMITECREDITO) FROM CLIENTES) AS
DIFERENCIA
FROM CLIENTES
WHERE LIMITECREDITO > (SELECT
AVG(LIMITECREDITO) FROM CLIENTES);
```