

# Diseño de Bases de Datos

---

SQL. Consultas y Restricciones

# *Structured Query Language*

» lenguaje relacional diseñado para la gestión de datos

## ■ Propuesta

- uno de los primeros lenguajes comerciales
- el más utilizado en el uso de las bases de datos (aunque existen otras propuestas)

## ■ Estandarización

- SQL-86 (primera ANSI), SQL-92 (revisión importante), SQL-99 (exp.reg., recursión, triggers, with, boolean, ...), SQL-2003 (XML, ...), SQL-2008 (...)
- los sistemas comerciales tienen, al menos, SQL-92
- incluyen más elementos, algunos específicos
- los específicos van en contra de la portabilidad, aunque pueden ser muy útiles

# SQL

- Ya vista una parte de DDL
- Nos quedan partes fundamentales:
  - Consultas: operativa mayoritaria en BD
  - Restricciones
  - Disparadores
- Respecto al Álgebra Relacional
  - es relacionalmente completo
  - utiliza *multiconjuntos*
  - construcciones adicionales

# Datos ejemplo

Cliente (idc, nombre, cat, edad)

Vehículo (matr, marca, color)

Reserva (idc, matr, fecha)



**Cliente**

idc	nombre	cat	edad
22	Davila	7	45
29	Bravo	1	32
31	Lorenzo	8	24
32	Alvarez	8	31
58	Rubio	4	67
64	Huerga	2	18
71	Zurro	10	45
74	Huerga	9	35
85	Arnaud	3	25
95	Benitez	3	63

**Reserva**

idc	matr	fecha
22	101	2009-11-07
64	101	2010-12-28
22	102	2010-04-21
31	102	2012-01-14
64	102	2010-04-11
22	103	2009-11-07
31	103	2011-07-19
74	103	2009-09-13
22	104	2012-01-01
31	104	2006-12-09

**Vehículo**

matr	marca	color
101	BMW	azul
102	Lancia	rojo
103	Seat	verde
104	BMW	rojo

# Consulta básica

```
SELECT [DISTINCT] lista-atributos  
FROM lista-tablas  
WHERE condición
```

- Lista tablas
  - tablas involucradas en los datos objeto de la búsqueda
  - posiblemente con un nombre de *correlación* cada una
- Lista atributos
  - atributos de la lista de tablas que son de interés
- Condición
  - expresiones (booleanas) sobre los datos que deben formar parte del resultado de la consulta
- **[DISTINCT]** es opcional para no incluir duplicados; por defecto no son eliminados

# Estrategia de evaluación

- Obtención de resultados
  1. producto cartesiano de las tablas
  2. eliminar las filas que no cumplen la condición
  3. eliminar los atributos que no están en la lista
  4. eliminar duplicados si es necesario
- Eficiencia – optimizador
  - estrategia menos eficiente para computar la consulta (obtener los resultados)
  - el optimizador encontrará estrategias más eficientes (que devuelven los mismos resultados)

# Join de tablas —implícito

```
SELECT C.nombre  
FROM Cliente C, Reserva R  
WHERE C.idc=R.idc AND R.matr=103;
```

- Nombre correlación / variable de rango
  - sólo son realmente necesarios cuando una relación aparece dos veces en la cláusula **FROM**
  - se recomienda utilizarlos siempre

```
SELECT nombre  
FROM Cliente, Reserva  
WHERE Cliente.idc=Reserva.idc AND matr=103;
```

nombre
Davila
Lorenzo
Huerga

- Condición de *join*
  - p.e., igualdad de los campos de combinación

...

## » Clientes que han reservado al menos un coche

```
SELECT C.nombre  
FROM Cliente C, Reserva R  
WHERE C.idc=R.idc;
```

nombre
Davila
Davila
Davila
Davila
Lorenzo
Lorenzo
Lorenzo
Huerga
Huerga
Huerga

nombre	idc
Davila	22
Lorenzo	31
Huerga	64
Huerga	74

②

③

idc	nombre
22	Davila
31	Lorenzo
64	Huerga
74	Huerga

④

- habrá que poner **DISTINCT** ②
- mejor por idc's distintos ③  
(¿interviene Cliente?)
- podemos añadir el nombre ④

```
SELECT DISTINCT C.idc, C.nombre
```

...



...

**Empleado**

id	nombre	dpto
11	Jiménez	ventas
12	Blanco	compras
13	Gracia	ventas
14	Fonseca	compras

**Departamento**

dpto	jefe
ventas	Carcedo
compras	Arias

– join natural: **SELECT \***  
**FROM Empleado E**  
**NATURAL JOIN Departamento D;**

**Empleado ⋈ Departamento**

id	nombre	dpto	jefe
11	Jiménez	ventas	Carcedo
12	Blanco	compras	Arias
13	Gracia	ventas	Carcedo
14	Fonseca	compras	Arias

# Join de tablas —explícito

- Natural join

```
SELECT C.nombre  
FROM Cliente C NATURAL JOIN Reserva R  
WHERE ...;
```

solo sobre atributos con el mismo nombre (no tiene por qué ocurrir)

- Inner (equi-) join

```
SELECT C.nombre  
FROM Cliente C INNER JOIN Reserva R  
    ON C.idc=R.idc  
WHERE ...;
```

se puede ver como una tabla *temporal* construida como el join de las dos, pero sin nombre propio

- Outer join

```
SELECT C.nombre  
FROM Cliente C LEFT OUTER JOIN Reserva R  
    ON C.idc=R.idc  
WHERE ...;
```

igualdad con el mismo nombre,  
**USING (i dc)** —en vez de **ON ...**

**LEFT | RIGHT | FULL**

# Expresiones

- » Nueva categoría para los clientes que hayan reservado dos coches diferentes el mismo día

```
SELECT C.nombre, C.cat+1 AS scat
FROM Cliente C, Reserva R1, Reserva R2
WHERE C.idc=R1.idc AND C.idc=R2.idc
      AND R1.fecha=R2.fecha AND R1.matr<>R2.matr;
```

- expresiones en la lista de selección
- habrá que utilizar **DISTINCT**  
—¿por qué la repetición?
- repetición de Reserva en **FROM**  
para referirse a los dos roles de búsqueda

nombre	scat
Davila	8
Davila	8

# Strings

- » Clientes cuyo nombre empieza por 'A', y tienen menos de 30 años

```

+-----+
| nombre | edad |
+-----+
|  Arnaud |   25 |
+-----+

```

- GII/ADBBD

# Manipulación de conjuntos

- **UNI ON, I NTERSECT, EXCEPT**

- hay sistemas con sólo la unión, teniendo que recurrirse a otro tipo de construcciones
- hay sistemas con **MI NUS** en lugar de **EXCEPT**

— para el resto de operaciones se introducen las *subconsultas* (más inspirado en el cálculo relacional)

- **I N, op ANY, op ALL**

- pertenencia y comparaciones con conjuntos

- **EXI STS**

- comprobación de conjunto vacío

- **NOT (I N, EXI STS)**

- modificación del significado por negación

# Unión

» Clientes que han reservado un coche rojo o verde

```
SELECT DISTINCT C.idc
FROM Cliente C, Vehiculo V, Reserva R
WHERE C.idc=R.idc AND R.matr=V.matr
      AND (V.color='rojo' OR V.color='verde');
```

idc
22
31
64
74

```
SELECT C.idc
FROM Cliente C, Vehiculo V, Reserva R
WHERE C.idc=R.idc AND R.matr=V.matr AND V.color='rojo'
UNION [ALL]
SELECT ... V.color='verde';
```

- los operadores sobre conjuntos del AR están disponibles en SQL —para *multiconjuntos*
- los conjuntos tienen que ser compatibles

# Intersección

» Clientes que han reservado un coche rojo y verde

```
SELECT ... V. color='rojo'  
INTERSECT  
SELECT ... V. color='verde' ;
```

idc
22
31

- no todos los DBMS lo incluyen (aunque es SQL-92)
- la otra opción es más compleja, basada en roles

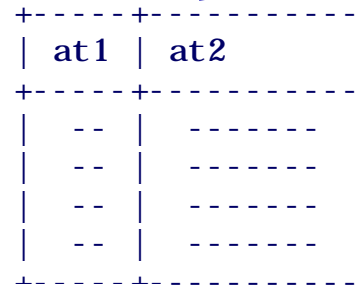
```
SELECT DISTINCT C.idc, C.nombre  
FROM Cliente C, Vehiculo V1, Reserva R1,  
               Vehiculo V2, Reserva R2  
WHERE C.idc=R1.idc AND R1.matr=V1.matr AND  
      C.idc=R2.idc AND R2.matr=V2.matr AND  
      V1.color='rojo' AND V2.color='verde' ;
```

# Consultas anidadas

- » tabla *temporal*, calculada construida como el resultado de un select, y
- » utilizada en otro select habitualmente en:
  - (1) como una tabla en el **FROM**
  - (2) en alguna condición del **WHERE**

## ■ Utilización en **FROM**

```
SELECT T.at1, T.at2  
FROM Tabla T  
WHERE ...;
```



at1	at2
--	-----
--	-----
--	-----
--	-----

sin nombre y, por tanto  
sin poder referenciarla

```
SELECT ...  
FROM (SELECT ...) Ttemp, ...  
WHERE ...;
```



nombre que permite referenciar  
Ttemp.at1, Ttemp.at2, ...



# Consultas anidadas

» en la cláusula **WHERE** aparece en construcciones más complejas

```
SELECT C.nombre  
FROM Cliente C  
WHERE C.idc IN (SELECT R.idc  
                FROM Reserva R  
                WHERE R.matr IN (SELECT V.matr  
                                FROM Vehiculo V  
                                WHERE V.color='rojo'));
```



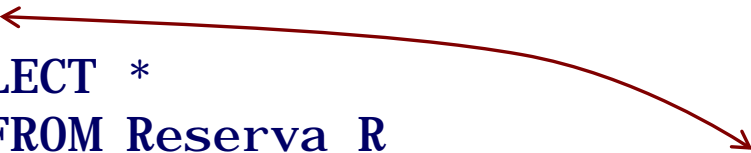
aquí no es necesario dar nombre a las tablas temporales

- **IN** comprueba la pertenencia; el complementario de la consulta se consigue con **NOT IN**
- se evitan los *joins* en base a subconsultas
- evaluación *anidada*: para cada tupla del nivel exterior, calcular la subconsulta —tiene esto sentido para subconsultas independientes del nivel exterior?

# Anidamiento correlacionado

- » subconsulta depende de la tupla que se esté examinando en la consulta exterior

```
SELECT C.nombre  
FROM Cliente C  
WHERE EXISTS (SELECT *  
               FROM Reserva R  
               WHERE R.matr=103 AND R.idc=C.idc);
```



- la aparición de **C** en la subconsulta denota la dependencia
- **EXISTS** comprueba que el conjunto no está vacío; se puede anteponer **NOT**
- el uso del **\*** se considera aquí buen estilo de programación

# Otros comparadores

» comparación (<, <=, =, <>, >=, >) que se cumplirá cuando alguno (**ANY**) o todos (**ALL**) los elementos lo cumplan

```
SELECT C.idc
FROM Cliente C
WHERE C.cat >= ALL (SELECT C2.cat
                    FROM Cliente C2);
```

```
+-----+
| idc |
+-----+
| 71  |
+-----+
```

– **IN** es equivalente a **=ANY**, y **NOT IN** a **<>ALL**

# Otras consultas anidadas

- alternativa a **INTERSECT**

```
SELECT C.nombre
FROM Cliente C, Vehiculo V, Reserva R
WHERE C.idc=R.idc AND R.matr=V.matr AND V.color='rojo' AND
      C.idc IN (SELECT C2.idc
                FROM Cliente C2, Vehiculo V2, Reserva R2
                WHERE C2.idc=R2.idc AND R2.matr=V2.matr
                  AND V2.color='verde');
```

- encontrar los idc's de clientes que hayan reservado un coche 'rojo' y, además, que estos idc's están incluidos en el conjunto de quienes hayan reservado un 'verde'

- alternativa a **EXCEPT** (ej. 'rojos', pero no 'verdes')

```
SELECT C.nombre FROM ... WHERE ... AND
      C.idc NOT IN (...);
```

# División

» ej.: clientes que han reservado todos los vehículos

» sin la utilización de **EXCEPT**

```
SELECT C.nombre
FROM Cliente C
WHERE NOT EXISTS (SELECT V.matr
                   FROM Vehiculo V
                   WHERE NOT EXISTS (SELECT R.matr
                                     FROM Reserva R
                                     WHERE R.matr=V.matr AND
                                           R.idc=C.idc));
```

nombre
Davila

- cliente tal que ...
- no exista un vehículo ...
- sin que exista una reserva de ese cliente con el vehículo

# Operadores de agregación

- » importante extensión al álgebra relacional
- » **COUNT, SUM, AVG, MAX, MIN**

```
SELECT COUNT(*)  
FROM Cliente C;
```

+	-----	+
	COUNT(*)	
+	-----	+
	10	
+	-----	+

```
SELECT AVG(DISTINCT C. edad)  
FROM Cliente C  
WHERE C. cat>5;
```

+	-----	+
	AVG(DISTINCT C. edad)	
+	-----	+
	33.7500	
+	-----	+

```
SELECT C. nombre, C. cat  
FROM Cliente C  
WHERE C. cat= (SELECT MAX(C2. cat)  
               FROM Cliente C2);
```

+	-----	+	-----	+
	nombre		cat	
+	-----	+	-----	+
	Zurro		10	
+	-----	+	-----	+

# Cláusulas **GROUP BY** y **HAVING**

- » operadores de agregación sobre grupos de tuplas
- » particionado de tuplas con los atributos de **GROUP BY** (un grupo para cada valor)
- » eliminación de grupos que no cumplan **HAVING**

```
SELECT C. cat, COUNT(*)  
FROM Cliente C  
WHERE C. edad>18  
GROUP BY C. cat;
```

cat	COUNT(*)
1	1
3	2
4	1
7	1
8	2
9	1
10	1

```
SELECT C. cat, MIN(C. edad) AS edadmi n  
FROM Cliente C  
WHERE C. edad>18  
GROUP BY C. cat  
HAVING COUNT(*)>1;
```

cat	edadmi n
3	25
8	24

...

## » Número de reservas de cada coche 'rojo'

```
SELECT V.matr, COUNT(*) AS nres
FROM Reserva R, Vehiculo V
WHERE R.matr=V.matr
GROUP BY V.matr
HAVING V.color='rojo';
```

ERROR xxxx: Unknown column  
'V.color' in 'having clause'

- solo pueden aparecer en **HAVING** los campos que aparecen en **GROUP BY**

```
SELECT V.matr, COUNT(*) AS nres
FROM Reserva R, Vehiculo V
WHERE R.matr=V.matr AND V.color='rojo'
GROUP BY V.matr;
```

matr	nres
102	3
104	2



# Cláusula ORDER BY

» operador de orden —en la salida

```
SELECT C. cat  
FROM Cliente C  
WHERE C. edad>18  
ORDER BY C. cat ASC;
```

cat
1
3
4
7
8
9
10

# Valores nulos

» cuando el valor es desconocido / inaplicable

## ■ Comparaciones

- la comparación de *desconocidos* es *desconocida*
- comparadores específicos **IS [NOT] NULL**

## ■ Conectivas lógicas

- lógica de tres valores: *verdadero, falso, desconocido*

## ■ Estructuras de SQL

- en cláusulas **WHERE** el desconocido no es seleccionado
- aritmética con nulos devuelve nulos; las operaciones de agregación los descartan —salvo **COUNT(\*)**

## ■ Outer joins

- se incluyen tuplas sin correspondencia, añadiendo nulos

# Restricciones de integridad

- » condiciones que debe cumplir toda instancia *legal* de la BD
- » se utilizan para asegurar la semántica de la aplicación

## ■ Tipos

- Implícitas: propias del modelo (relacional)
  - dominio, clave primaria, clave foránea
- Generales: condiciones de la aplicación
  - cualquier condición impuesta por los requisitos —ej.:  
“no se puede contratar TV sin ADSL de alta velocidad”
  - pueden afectar a una o a varias tablas

# Restricciones sobre una tabla

- » se comprueban cuando hay una actualización de esa tabla
- » pueden hacer referencia a otras tablas

```
CREATE TABLE Reserva (  
    idc ..., matr ..., fecha ...,  
    PRIMARY KEY ..., FOREIGN KEY ...,  
    CHECK (fecha > ...),  
    CONSTRAINT nocliente  
        CHECK ( 'Meloquedo' <> ( SELECT C.nombre  
                                   FROM Cliente C  
                                   WHERE C.idc=idc )));
```

- se pueden utilizar consultas para expresar restricciones

# Restricciones sobre varias tablas

## ■ Aserciones

- implica a dos o más tablas, sin asociación concreta a una de ellas
- se comprueba cuando se modifica alguna de ellas

```
CREATE ASSERTION limite  
CHECK ( (SELECT COUNT(*) FROM Cliente C) +  
        (SELECT COUNT(*) FROM Vehiculo V) < 100 );
```

- se puede implementar con un **CHECK** pero sería engorroso y con el problema de tener la tabla de clientes vacía, p.e.

# Disparadores / *triggers*

» procedimiento que se dispara automáticamente si se producen cambios específicos (SQL-99)

## ■ Partes

1. Evento: activa el disparador
2. Condición: instrucción (falso|verd.) o consulta (vacía|no)
3. Acción: procedimiento que se ejecuta cuando se activa el disparador y la condición es verdadera

```
CREATE TRIGGER actualizarTitularActual  
  AFTER INSERT ON HistorialLicencia  
  REFERENCING NEW AS nuevoTraspaso  
  FOR EACH ROW  
  UPDATE Licencia L  
    SET L.titularActual = nuevoTraspaso.dniTitular  
  WHERE L.nro = nuevoTraspaso.nroLicencia;
```