

Temario

- ♦ Introducción y fundamentos
- ♦ Introducción a SQL
- ♦ Modelo Entidad / Relación
- ♦ Modelo relacional
- ♦ Diseño relacional: formas normales
- ♦ Consultas
 - Cálculo relacional
 - Álgebra relacional
- ♦ Implementación de bases de datos
 - Estructura física: campos y registros
 - Indexación
 - Índices simples
 - Árboles B
 - Hashing

Structured Query Language – SQL

- ◆ Lenguaje de “programación” para SGBDs
 - Data definition language: creación del modelo de datos (diseño de tablas)
 - Data manipulation language: inserción, modificación, eliminación de datos
 - Data query language: consultas
- ◆ Se ejecuta sobre un SGBD
- ◆ El estándar más utilizado
 - Creado en 1974 (D. D. Chamberlin & R. F. Boyce, IBM)
 - ANSI en 1986, ISO en 1987
 - Core (todos los SGBD) + packages (módulos opcionales)
- ◆ Versiones
 - SQL1 – SQL 86
 - SQL2 – SQL 92
 - SQL3 – SQL 1999, no plenamente soportado por la industria (recursión, programación, objetos...)
- ◆ Limitaciones
 - No es puramente relacional (p.e. las *vistas* son *multiconjuntos* de tuplas)
 - Importantes divergencias entre implementaciones (no es directamente portable en general, incompletitudes, extensiones) –uno termina aprendiendo variantes de SQL



Donald D.
Chamberlin

Elementos de una base de datos SQL

- ♦ Base de datos = conjunto de tablas
- ♦ Tabla (relación, entidad, esquema...) =
 - Estructura fija de campos (esquema)
 - Conjunto de registros con valores de campos
- ♦ Campo (atributo, propiedad, “columna”), tiene un tipo de dato
- ♦ Registro (tupla, “fila”)
- ♦ Clave primaria
- ♦ Claves externas

Estructura léxica del lenguaje

Operaciones SQL

- ♦ DDL – Creación, diseño, modificación y eliminación de tablas
- ♦ DML – Inserción, modificación, eliminación de registros
- ♦ DQL – Consulta

Estructura léxica de SQL

- ♦ Case-insensitive, insignificant whitespace
- ♦ Sentencias, expresiones, valores, tipos de datos
- ♦ Referencias
 - Elmasri cap. 8
 - PostgreSQL SQL ref: <http://www.postgresql.org/docs/9.4>

Ejemplo: BD para aplicación de música

The screenshot displays the Spotify Premium desktop application. The interface is dark-themed with a green accent color. At the top, the Spotify logo and 'Spotify Premium' text are visible. Below the menu bar (File, Edit, View, Playback, Help), there is a search bar and navigation icons. The left sidebar contains 'Stations', 'Local Files', and a 'PLAYLISTS' section with various playlist thumbnails. The main content area shows a playlist titled 'chosen just for you' with a description 'chosen just for you. Updated every Monday, so save your favourites!' and 'Created by: spotifydiscover • 30 songs, 1 hr 57 min'. Below the playlist title are buttons for 'PLAY', 'FOLLOWING', and 'CREATE SIMILAR PLAYLIST'. A 'Filter' search bar and an 'Available Offline' toggle are also present. The playlist table lists songs with their respective artists and update times. At the bottom, a playback control bar shows the current song 'Piano S' by Ludwig v, with a progress bar and various playback controls. A green bar at the very bottom states 'You are listening on'.

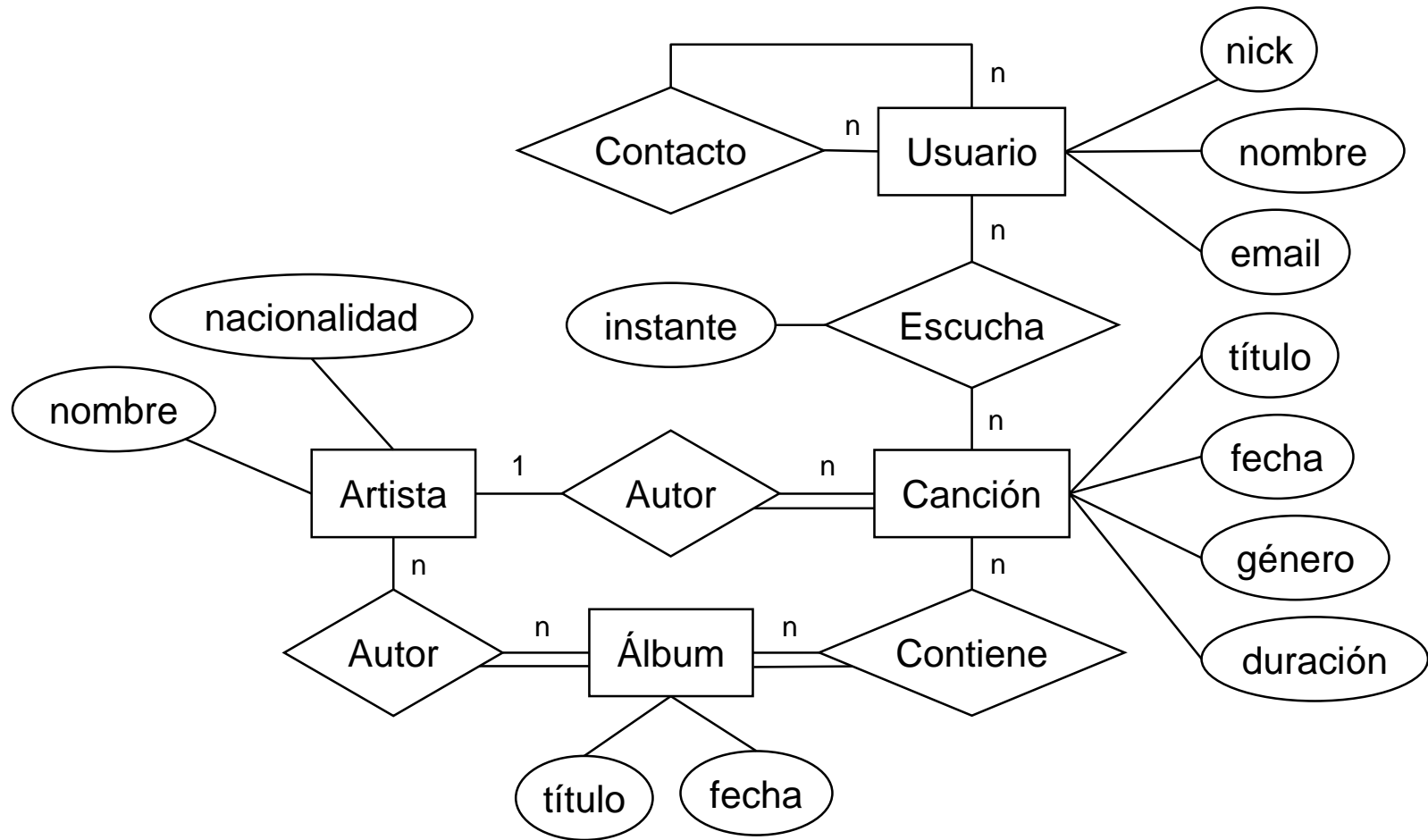
	SONG	ARTIST	
+	Same To You	Melody Gardot	2 days ago
+	Holding Up - Radio Edit	Jabberwocky, Na Kyung Lee	2 days ago
+	Palmar	Caloncho	2 days ago
+	Sunny	Bryan Adams	2 days ago
+	Jour 1 - Gostan Remix	Louane	2 days ago
+	Quicksand	Caro Emerald	2 days ago

Ejemplo: descripción informal

Aplicación de música online con red social

- ◆ Tipos de datos: usuarios, canciones, discos, autores...
- ◆ Estructuras:
 - Los usuarios tienen nick, nombre, email...
 - Las canciones tienen título, género, duración, fecha...
 - Los artistas tienen nombre, nacionalidad...
- ◆ Relaciones:
 - Las canciones tienen autores, los discos tienen canciones, los usuarios tienen amigos, discos favoritos, escuchan canciones...
- ◆ Funcionalidades:
 - Buscar una canción, escucharla, ver sus datos...
 - Ver / añadir amigos...

Ejemplo: diagrama ER



Artista

id	nombre	nacionalidad
0	The Beatles	UK
1	The Rolling Stones	UK
2	David Bowie	UK

Ejemplo: vista tablas

Cancion

id	titulo	genero	duración	fecha	autor
0	Norwegian Wood	Pop	125	1965-03-12	0
1	Here, there and everywhere	Pop	145	1966-08-05	0
2	Jumping jack flash	Pop	225	1968-04-20	1

Usuario

nick	nombre	email
lola	Dolores	lola@gmail.com
pepe	José	jose@gmail.com
chema	José María	chema@gmail.com
charo	Rosario	rosario@gmail.com

Contacto

usuario1	usuario2
pepe	lola
charo	pepe
chema	charo

Escucha

usuario	cancion	instante
charo	1	2011-09-09 16:57:54
pepe	2	2011-09-12 21:15:30

Motivación

Base de datos de músicos, canciones, usuarios, escuchas, red social

Ejemplos de consultas

Canciones de los años 60 (mostrar título y género)

Conjunto de nacionalidades de los artistas cubiertas en la BD

Canciones de artistas del Reino Unido

Título de las canciones escuchadas por un usuario

Todos los contactos de un usuario dado

Usuarios que se llaman igual

Usuarios a distancia 2 de un usuario dado en la red social

Contactos comunes a dos usuarios

Cuántas veces ha sido escuchada una canción

Artistas por orden de más a menos escuchados

Usuarios con más de dos contactos

Usuarios ordenados por nº de contactos

Usuario con más contactos

Data definition

```
CREATE TABLE nombre (  
    campo1 tipo1 [restricciones1],  
    campo2 tipo2 [restricciones2],  
    ...,  
    [restricciones]  
);
```

Comandos ALTER útiles cuando
ya hay una tabla creada y con datos



```
ALTER TABLE nombre ADD COLUMN campo tipo [restricciones];
```

```
ALTER TABLE nombre ADD restricción;
```

```
ALTER TABLE nombre DROP COLUMN campo;
```

```
DROP TABLE nombre;
```

```
DROP CONSTRAINT nombre-restricción;
```

Ejemplo

```
CREATE TABLE Artista (  
    id            int            PRIMARY KEY,  
    nombre        text          NOT NULL,  
    nacionalidad  text  
);  
  
CREATE TABLE Cancion (  
    id            int            PRIMARY KEY,  
    titulo        text          NOT NULL,  
    genero        text,  
    duracion      int,  
    fecha         date,  
    autor         int            NOT NULL REFERENCES Artista (id)  
);
```

```
CREATE TABLE Usuario (  
    nick          varchar(30) PRIMARY KEY,  
    nombre        text        NOT NULL,  
    email         text        NOT NULL UNIQUE  
);
```

```
CREATE TABLE Contacto (  
    usuario1      varchar(30) REFERENCES Usuario (nick),  
    usuario2      varchar(30) REFERENCES Usuario (nick),  
    PRIMARY KEY (usuario1,usuario2)  
);
```

```
CREATE TABLE Escucha (  
    usuario      varchar(30),  
    cancion      int          REFERENCES Cancion (id),  
    instante     timestamp,  
    PRIMARY KEY (usuario,cancion,instante)  
);
```

```
ALTER TABLE Escucha DROP COLUMN instante;  
ALTER TABLE Escucha ADD instante timestamp; /* NULL's */
```

```
ALTER TABLE Escucha ADD FOREIGN KEY (usuario)  
    REFERENCES Usuario (nick);
```

```
ALTER TABLE Usuario ADD PRIMARY KEY (nick);
```

Restricciones

En un campo

NOT NULL

UNIQUE

PRIMARY KEY

REFERENCES *tabla* (*clave*) [(ON DELETE | ON UPDATE)

(NO ACTION | RESTRICT | CASCADE
| SET NULL | SET DEFAULT)]

DEFAULT *valor*

Con nombre

CONSTRAINT *nombre* *restricción*

Si se omite, tomará
la clave primaria



En una tabla

PRIMARY KEY (*campo1*, *campo2*, ...)

FOREIGN KEY (*campo1*, *campo2*, ...) REFERENCES *tabla* (*clave1*, *clave2*, ...)

UNIQUE (*campo1*, *campo2*, ...)

CHECK (*expresión*)

Claves primarias

- ♦ Designan un identificador único de las filas de una tabla
- ♦ Sólo puede haber una clave primaria por tabla, aunque puede incluir varios campos
- ♦ Es muy aconsejable que toda tabla tenga su clave primaria
- ♦ Técnicamente equivalen a UNIQUE más NOT NULL
- ♦ Pero juegan un papel diferente en indexación (lo veremos más adelante)
- ♦ Opción de diseño: selección de clave primaria entre varias posibles
 - Clave primaria natural (email, dominio web, DNI, ISBN, etc.)
 - Clave primaria artificial: p.e. un ID entero (típicamente autoincremental), una cadena de caracteres (códigos), etc.

Claves externas

- ◆ Conceptualmente son comparables a punteros
- ◆ Referencian campos únicos de otra tabla
- ◆ Normalmente el campo referenciado es una clave primaria
 - O bien como mínimo tiene que ser declarado 'unique'
- ◆ Técnicamente no es imprescindible usarlas
- ◆ Pero ayudan a asegurar la consistencia en las referencias!
 1. Generan un error cuando se intenta insertar o cambiar una clave externa por un valor que no existe en la tabla referenciada
 2. Y permiten establecer qué se debe hacer cuando desaparece una clave referenciada
- ◆ En general es preferible (más eficiente) que sean de tipo entero

Claves: en resumen...

- ♦ La **clave primaria** de una tabla actúa como **identificador** de las filas
 - Juega un papel similar a la dirección de memoria RAM de un dato en C
 - Sólo hay una por tabla (aunque no es obligatorio, definir siempre una)
 - Puede ser un campo natural o (mejor) artificial (un entero en este caso)
 - Podría estar formada por varios campos
- ♦ Las **claves externas** juegan el papel de **punteros** entre filas de distintas (o las mismas) tablas
 - Deben apuntar a una clave primaria
 - Podrían apuntar también a un campo unique not null
 - Una clave externa puede estar formada por varios campos
- ♦ ¿Qué **ventaja** tiene declarar una **clave primaria**?
 - Obligar a que no se repitan valores en distintas filas
 - Diferencia con unique not null: cuestión de implementación (índices)
- ♦ ¿Qué **ventaja** tiene declarar **claves externas**?
 - Obligar a que su valor aparezca en alguna fila de la tabla apuntada
 - Reaccionar automáticamente a cambios en la clave primaria apuntada

Artista

id	nombre	nacionalidad
0	The Beatles	UK
1	The Rolling Stones	UK
2	David Bowie	UK

Ejemplo: vista tablas

Cancion

id	titulo	genero	duración	fecha	autor
0	Norwegian Wood	Pop	125	1965-03-12	0
1	Here, there and everywhere	Pop	145	1966-08-05	0
2	Jumping jack flash	Pop	225	1968-04-20	1

Usuario

nick	nombre	email
lola	Dolores	lola@gmail.com
pepe	José	jose@gmail.com
chema	José María	chema@gmail.com
charo	Rosario	rosario@gmail.com

Contacto

usuario1	usuario2
pepe	lola
charo	pepe
chema	charo

Escucha

usuario	cancion	instante
charo	1	2011-09-09 16:57:54
pepe	2	2011-09-12 21:15:30

Data manipulation

```
INSERT INTO tabla [(campo1, campo2, ...)] VALUES  
    (valor11, valor12, ...),  
    (valor21, valor22, ...),  
    ...  
;
```

```
UPDATE tabla SET campo1 = valor1, campo2 = valor2, ...  
/* PostgreSQL: [FROM ...] para formar condiciones con otras tablas */  
[WHERE ...];
```

```
DELETE FROM tabla  
/* PostgreSQL: [USING tabla1, tabla2, ...] para condiciones con otras tablas */  
[WHERE ...];
```

```
TRUNCATE TABLE tabla;
```

Ejemplo

```
INSERT INTO Artista VALUES (0, 'The Beatles', 'UK');
```

```
INSERT INTO Artista VALUES (1, 'The Rolling Stones', 'UK');
```

```
INSERT INTO Artista (id, nombre) VALUES (2, 'David Bowie');
```

```
INSERT INTO Cancion VALUES
```

```
    (0, 'Norwegian wood', 'Pop', 125, '1965-03-12', 0),
```

```
    (1, 'Here, there and everywhere', 'Pop', 145, '1966-08-05', 0),
```

```
    (2, 'Jumping jack flash', 'Pop', 225, '1968-04-20', 1);
```

```
INSERT INTO Usuario VALUES
```

```
    ('lola', 'Dolores', 'lola@gmail.com'),
```

```
    ('pepe', 'José', 'jose@gmail.com'),
```

```
    ('chema', 'José María', 'chema@gmail.com'),
```

```
    ('charo', 'Rosario', 'rosario@gmail.com');
```

```
INSERT INTO Contacto VALUES
```

```
    ('pepe', 'lola'),
```

```
    ('charo', 'pepe'),
```

```
    ('chema', 'charo');
```

```
INSERT INTO Escucha VALUES
```

```
    ('charo', 2, '2011-09-09 16:57:54'),
```

```
    ('pepe', 3, '2011-09-12 21:15:30');
```

```
UPDATE Artista SET nacionalidad = 'UK' WHERE nombre = 'David Bowie';
```

```
UPDATE Album SET precio = precio * 1.2;
```

```
DELETE FROM Escucha WHERE instante < '2000-01-01 00:00:00';
```

Tipos y expresiones

Tipos SQL


character(*n*) \equiv char(*n*), varchar(*n*), text

integer \equiv int, smallint

float, real, double precision

numeric (*precisión*, *escala*) \equiv decimal (*precisión*, *escala*)

date, time, timestamp


dígitos *decimales*
(por defecto 0)

Valores literales

Cadenas de caracteres entre '...'

Valores numéricos similar p.e. a C

date 'YYYY-MM-DD', time 'HH:MM:SS'

Expresiones

Se pueden utilizar en WHERE, SELECT, SET, DEFAULT, CHECK...

Operadores

+ - * / % ^

AND OR NOT

= < > <= >= LIKE ISNULL

operaciones con strings: concatenación, like, expresiones regulares ('%' '_')

Comentarios

--

/* ... */

Motivación

Base de datos de músicos, canciones, usuarios, escuchas, red social

Ejemplos de consultas

Canciones de los años 60 (mostrar título y género)

Conjunto de nacionalidades de los artistas cubiertas en la BD

Canciones de artistas del Reino Unido

Título de las canciones escuchadas por un usuario

Todos los contactos de un usuario dado

Usuarios que se llaman igual

Usuarios a distancia 2 de un usuario dado en la red social

Contactos comunes a dos usuarios

Cuántas veces ha sido escuchada una canción

Artistas por orden de más a menos escuchados

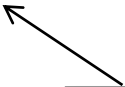
Usuarios con más de dos contactos

Usuarios ordenados por nº de contactos

Usuario con más contactos

Data query

SELECT [DISTINCT] *campos* FROM *tablas*
[WHERE *condición*];



Pueden ser expresiones
sobre campos

Ejemplos:

```
SELECT titulo, genero FROM Cancion
```

```
WHERE fecha > '1959-12-31' AND fecha < '1970-01-01';
```

```
SELECT DISTINCT nacionalidad FROM Artista;
```

```
/* Varias tablas */
```

```
SELECT * FROM Cancion, Artista
```

```
WHERE Cancion.autor = Artista.id AND Artista.nacionalidad = 'UK';
```

```
/* Expresiones */
```

```
SELECT dni, teoria * 0.6 + practicas * 0.4 FROM Notas;
```


Motivación

Base de datos de músicos, canciones, usuarios, escuchas, red social

Ejemplos de consultas

Canciones de los años 60 (mostrar título y género)

Conjunto de nacionalidades de los artistas cubiertas en la BD

Canciones de artistas del Reino Unido

Título de las canciones escuchadas por un usuario

Todos los contactos de un usuario dado el nombre de éste

Usuarios que se llaman igual

Usuarios a distancia 2 de un usuario dado en la red social

Contactos comunes a dos usuarios

Cuántas veces ha sido escuchada una canción

Artistas por orden de más a menos escuchados

Usuarios con más de dos contactos

Usuarios ordenados por nº de contactos

Usuario con más contactos

Join

SELECT *campos*

FROM *tabla1* **JOIN** *tabla2* **ON** *condición*

[WHERE *condición*];

Uso típico (pero no sólo) con claves externas: ON *externa* = *primaria*
Más eficiente (?) que el producto cartesiano (i.e. juntar tablas sin más)

Ejemplos:

Semánticamente
equivalentes

{
SELECT titulo FROM Cancion, Escucha
WHERE usuario = 'lola' AND cancion = id;

SELECT titulo FROM Cancion JOIN Escucha ON cancion = id
WHERE usuario = 'lola';

SELECT * FROM Contacto JOIN Usuario
ON (usuario1 = nick OR usuario2 = nick)
WHERE nombre = 'Rosario';

Tipos de join

- ♦ INNER Por defecto (no hace falta ponerlo)
- ♦ LEFT | RIGHT | FULL Se añaden también filas que no cumplen la condición (incompatible con INNER), útil especialmente en ciertas consultas con operaciones de agregación
- ♦ NATURAL La condición consiste en igualdad entre los campos que se llamen igual

tabla1 JOIN *tabla2* USING (*campos*) equivale a un natural join ceñido a los campos indicados en using.

Tipos de join (cont)

Ejemplo

```
CREATE TABLE Estudiante (  
    dni VARCHAR(12) PRIMARY KEY, nombre text);  
CREATE TABLE Asignatura (  
    codigo NUMERIC PRIMARY KEY, nombre text);  
CREATE TABLE Notas (  
    dni VARCHAR(12) REFERENCES Estudiante(dni),  
    codigo NUMERIC REFERENCES Asignatura(codigo),  
    teoria NUMERIC (4,2), practicas NUMERIC (4,2),  
    PRIMARY KEY (dni, codigo));  
  
SELECT nombre, teoria FROM Notas NATURAL JOIN Asignatura;  
  
SELECT nombre, teoria FROM Notas JOIN Asignatura  
ON Notas.codigo = Asignatura.codigo;
```

Motivación

Base de datos de músicos, canciones, usuarios, escuchas, red social

Ejemplos de consultas

Canciones de los años 60 (mostrar título y género)

Conjunto de nacionalidades de los artistas cubiertas en la BD

Canciones de artistas del Reino Unido

Título de las canciones escuchadas por un usuario

Todos los contactos de un usuario dado el nombre de éste

Usuarios que se llaman igual

Usuarios a distancia 2 de un usuario dado en la red social

Contactos comunes a dos usuarios

Cuántas veces ha sido escuchada una canción

Artistas por orden de más a menos escuchados

Usuarios con más de dos contactos

Usuarios ordenados por nº de contactos

Usuario con más contactos

Alias

SELECT *campos* FROM *tabla* **AS** *alias* [(*alias-campo1*, *alias-campo2*, ...)]
[WHERE *condición*];

SELECT *campo* **AS** *alias* FROM ...

Ejemplo:

SELECT u1.nombre FROM Usuario **AS** u1, Usuario **AS** u2
WHERE u1.nombre = u2.nombre AND u1.nick < > u2.nick;

SELECT dni, teoria * 0.6 + practicas * 0.4 **AS** media FROM Notas;

Motivación

Base de datos de músicos, canciones, usuarios, escuchas, red social

Ejemplos de consultas

Canciones de los años 60 (mostrar título y género)

Conjunto de nacionalidades de los artistas cubiertas en la BD

Canciones de artistas del Reino Unido

Título de las canciones escuchadas por un usuario

Todos los contactos de un usuario dado el nombre de éste

Usuarios que se llaman igual

Usuarios a distancia 2 de un usuario dado en la red social

Contactos comunes a dos usuarios

Cuántas veces ha sido escuchada una canción

Artistas por orden de más a menos escuchados

Usuarios con más de dos contactos

Usuarios ordenados por nº de contactos

Usuario con más contactos

Consultas anidadas

SELECT *campos* FROM (SELECT ...) AS *alias* WHERE ...;

SELECT *campos* FROM *tabla*

WHERE *campo1, campo2, ...* **IN** (SELECT *campo1, campo2, ...*);

SELECT *campos* FROM *tabla*

WHERE *campo comparación* (**SOME** | **ALL**) (SELECT ...);

SELECT *campos* FROM *tabla*

WHERE [NOT] **EXISTS** (SELECT ...);

SELECT *campos* FROM *tabla*

WHERE (SELECT ...) **CONTAINS** (SOME | ALL) (SELECT ...);

Consultas anidadas (cont)

Ejemplos:

```
SELECT c1.usuario1  
FROM Contacto AS c1 JOIN  
(SELECT * FROM Contacto WHERE Contacto.usuario2 = 'lola')  
AS c2      /* En FROM siempre con AS */  
ON c1.usuario2 = c2.usuario1
```

```
SELECT usuario1 FROM Contacto  
WHERE usuario2 IN (SELECT usuario1 FROM Contacto  
                  WHERE usuario2 = 'lola')
```

Motivación

Base de datos de músicos, canciones, usuarios, escuchas, red social

Ejemplos de consultas

Canciones de los años 60 (mostrar título y género)

Conjunto de nacionalidades de los artistas cubiertas en la BD

Canciones de artistas del Reino Unido

Título de las canciones escuchadas por un usuario

Todos los contactos de un usuario dado el nombre de éste

Usuarios que se llaman igual

Usuarios a distancia 2 de un usuario dado en la red social

Contactos comunes a dos usuarios

Cuántas veces ha sido escuchada una canción

Artistas por orden de más a menos escuchados

Usuarios con más de dos contactos

Usuarios ordenados por nº de contactos

Usuario con más contactos

Álgebra de conjuntos

consulta1 UNION *consulta2*

consulta1 INTERSECT *consulta2*

consulta1 EXCEPT *consulta2*

Tuplas homogéneas: los conjuntos de tuplas tienen que tener los mismos campos

Aplica un **DISTINCT implícito** (a menos que indiquemos ALL)

Ejemplo:

(SELECT usuario2 FROM Contacto WHERE usuario1 = 'charo'

UNION

SELECT usuario1 FROM Contacto WHERE usuario2 = 'charo')

INTERSECT

(SELECT usuario2 FROM Contacto WHERE usuario1 = 'lola'

UNION

SELECT usuario1 FROM Contacto WHERE usuario2 = 'lola')

Motivación

Base de datos de músicos, canciones, usuarios, escuchas, red social

Ejemplos de consultas

Canciones de los años 60 (mostrar título y género)

Conjunto de nacionalidades de los artistas cubiertas en la BD

Canciones de artistas del Reino Unido

Título de las canciones escuchadas por un usuario

Todos los contactos de un usuario dado el nombre de éste

Usuarios que se llaman igual

Usuarios a distancia 2 de un usuario dado en la red social

Contactos comunes a dos usuarios

Cuántas veces ha sido escuchada una canción

Cuántas veces ha sido escuchado cada artista

Usuarios con más de dos contactos

Usuarios ordenados por nº de contactos

Usuario con más contactos

Orden, agregación: ejemplos

```
SELECT COUNT (*) FROM Escucha JOIN Cancion ON cancion = id  
WHERE titulo = 'Norwegian Wood';
```

```
SELECT autor, COUNT (*) FROM Escucha JOIN Cancion ON cancion = id  
GROUP BY autor;
```

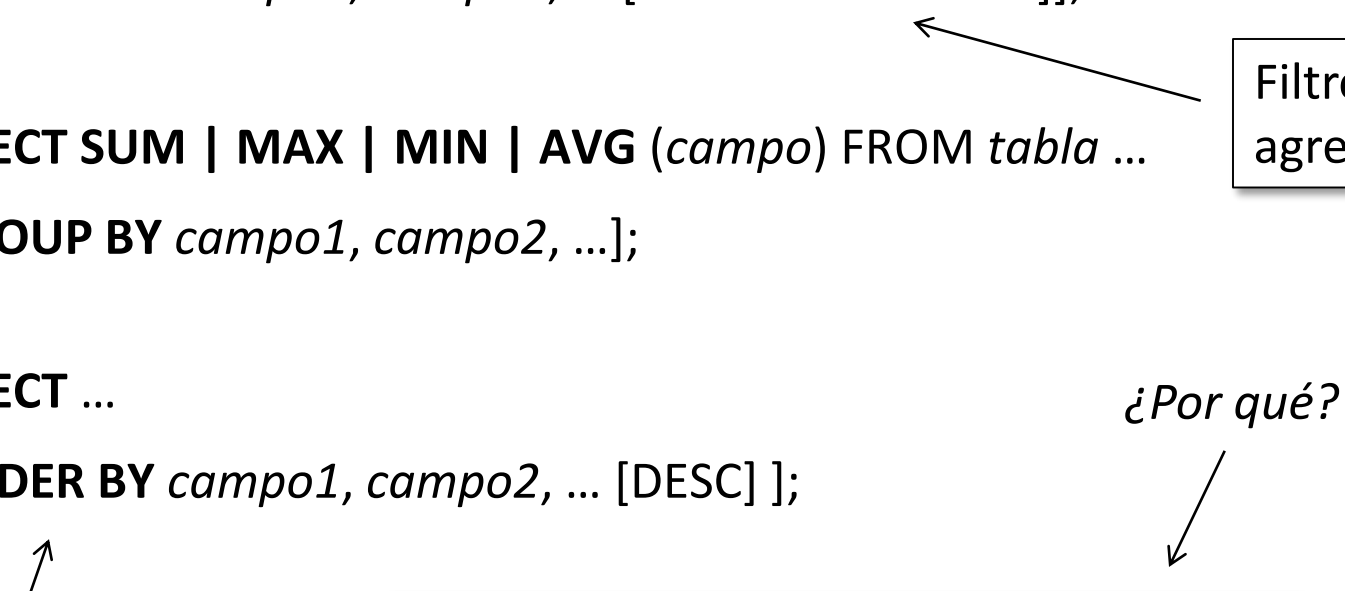
```
SELECT autor, COUNT (*) AS n FROM Escucha JOIN Cancion ON cancion = id  
GROUP BY autor  
ORDER BY n  
DESC  
LIMIT 1
```

```
SELECT autor FROM Escucha JOIN Cancion ON cancion = id  
GROUP BY autor  
HAVING COUNT (*) > 100
```

Orden, agregación

SELECT COUNT ([DISTINCT] *campo*) FROM *tabla* ...
[**GROUP BY** *campo1*, *campo2*, ... [**HAVING** *condición*]];

Filtro post-agregación



SELECT SUM | MAX | MIN | AVG (*campo*) FROM *tabla* ...
[**GROUP BY** *campo1*, *campo2*, ...];

SELECT ...
[**ORDER BY** *campo1*, *campo2*, ... [DESC]];

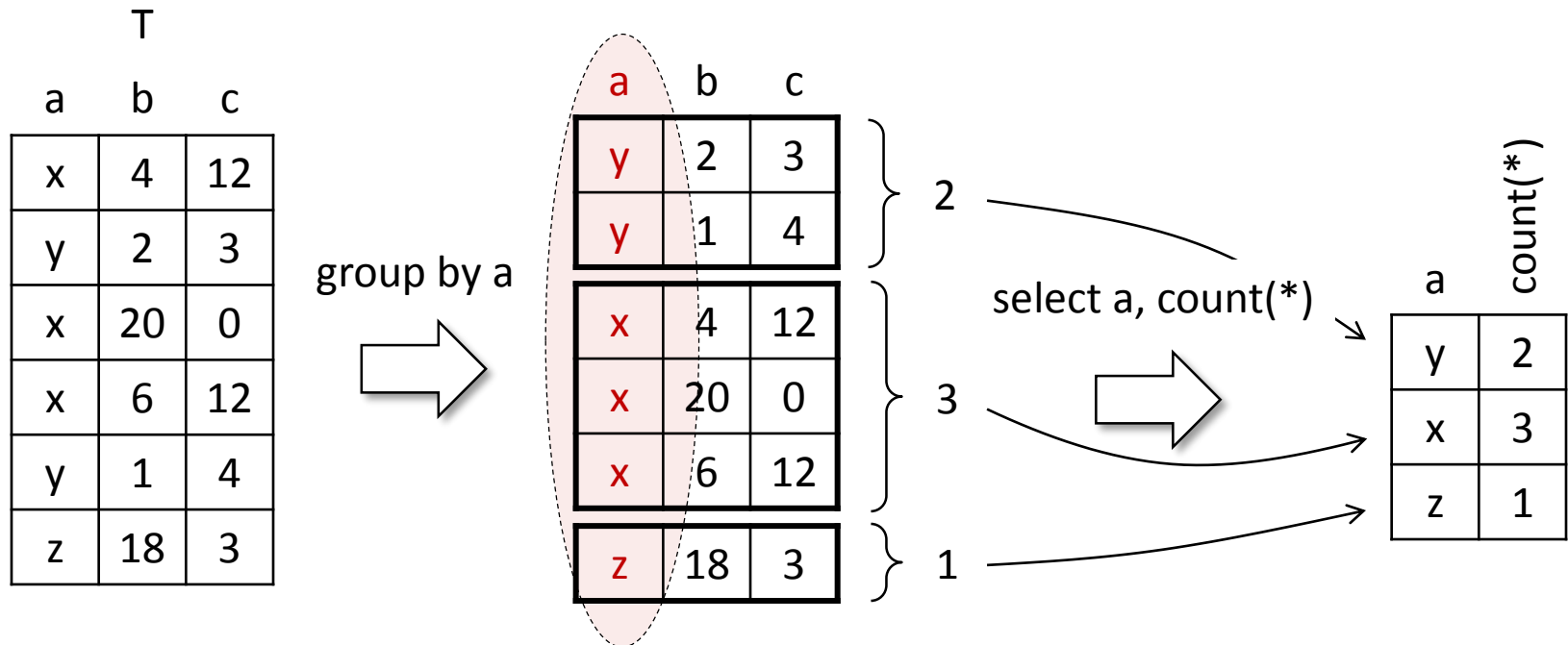
¿Por qué?

Útil combinar
con LIMIT *n*

En general si se usa GROUP BY, sólo
se pueden usar esos campos en SELECT
(aunque algunos SGBD lo toleran)

Orden, agregación: ejemplo

SELECT a, count(*) FROM T GROUP BY a

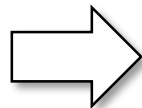


Orden, agregación: ejemplo

SELECT a, count(*), **sum(b)** FROM T GROUP BY a

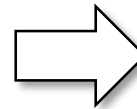
T		
a	b	c
x	4	12
y	2	3
x	20	0
x	6	12
y	1	4
z	18	3

group by a



a	b	c
y	2	3
y	1	4
x	4	12
x	20	0
x	6	12
z	18	3

select a, count(*), sum(b)



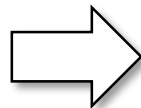
a	count(*)	sum(b)
y	2	3
x	3	20
z	1	18

Orden, agregación: ejemplo

SELECT a, count(*), sum(b) FROM T GROUP BY a, c

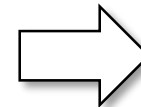
T		
a	b	c
x	4	12
y	2	3
x	20	0
x	6	12
y	1	4
z	18	3

group by a, c



a	b	c
y	2	3
y	1	4
x	4	12
x	6	12
x	20	0
z	18	3

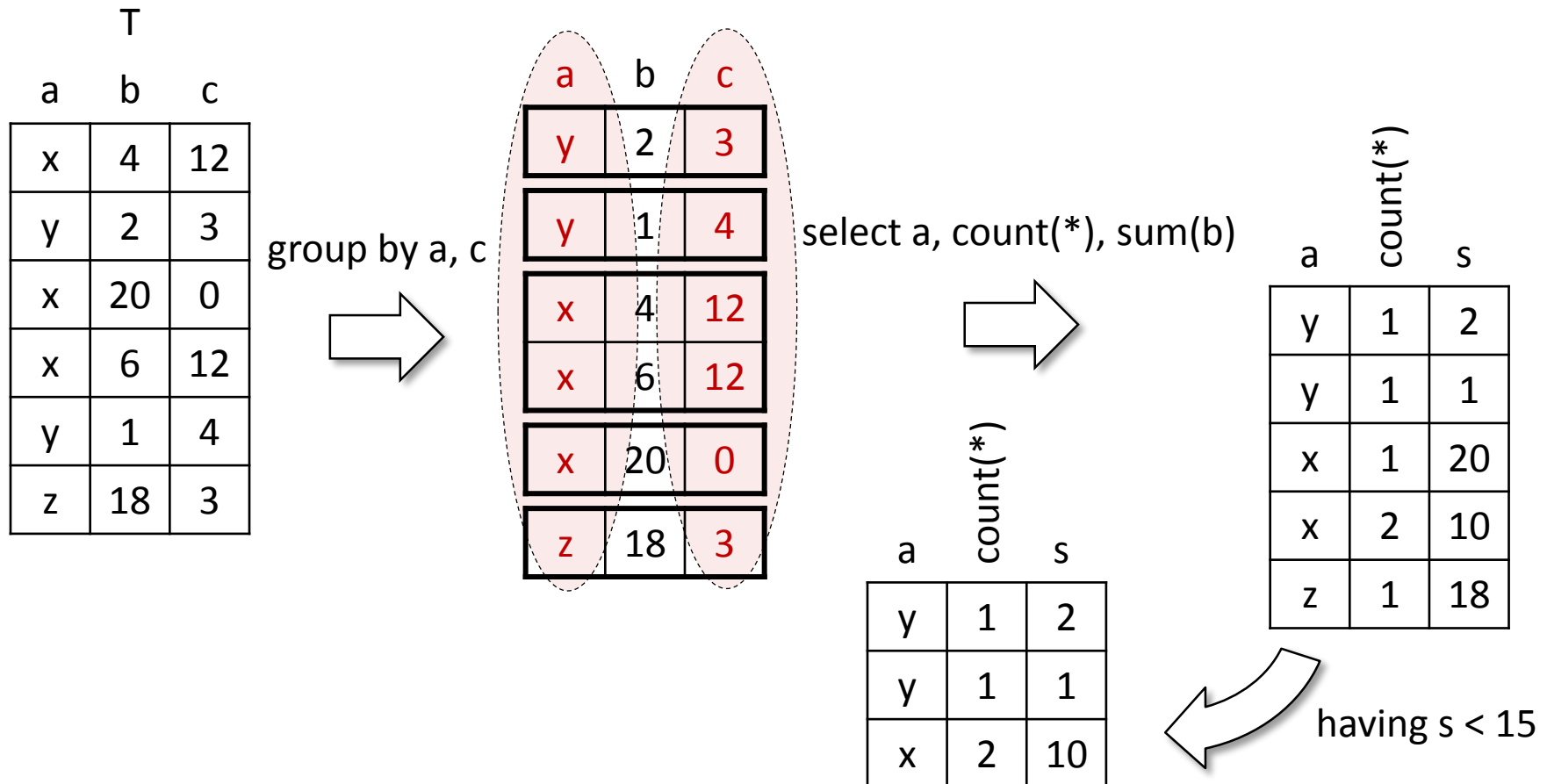
select a, count(*), sum(b)



a	count(*)	sum(b)
y	1	2
y	1	1
x	1	20
x	2	10
z	1	18

Orden, agregación: ejemplo

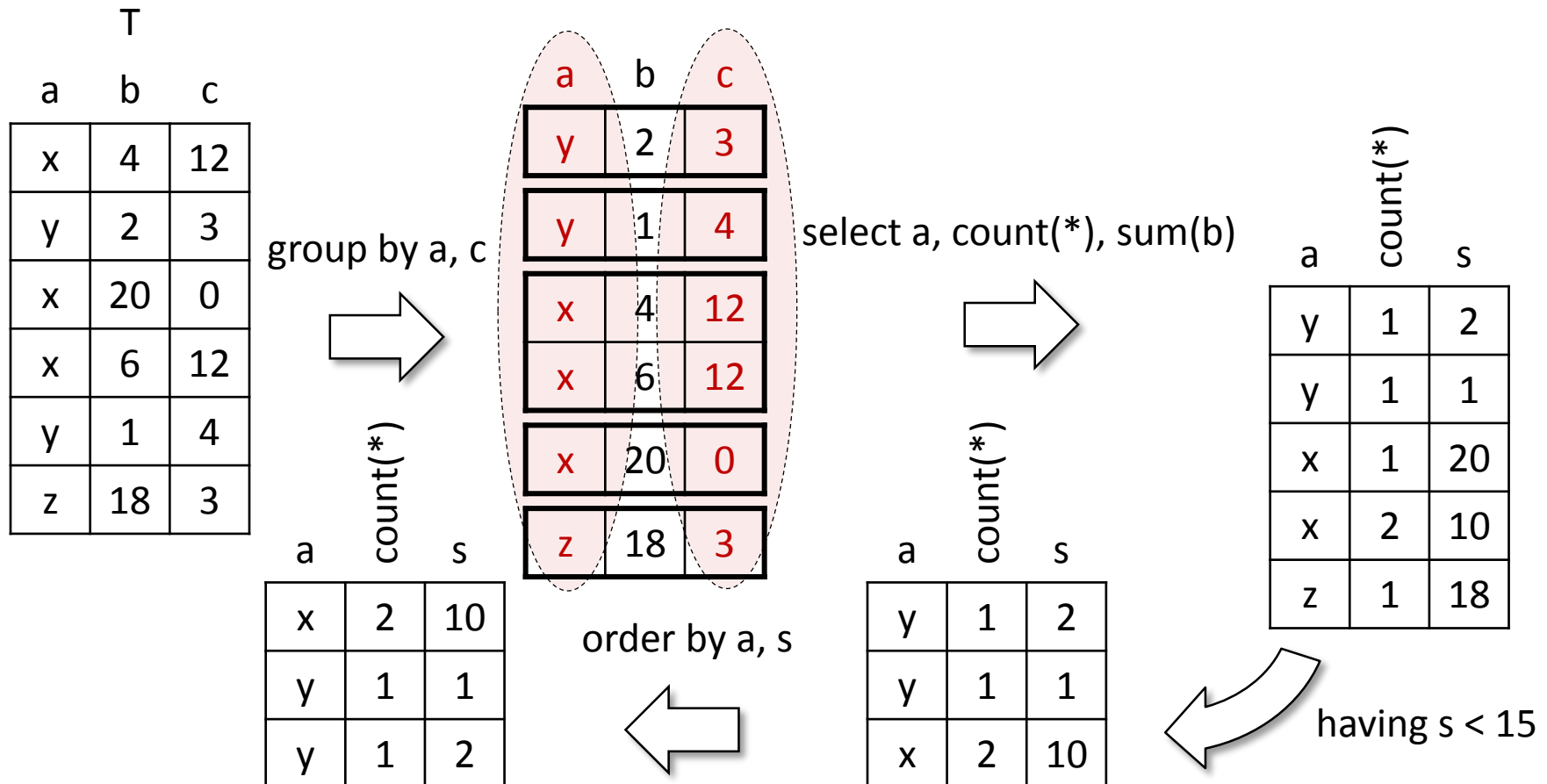
SELECT a, count(*), sum(b) as s FROM T GROUP BY a, c
HAVING s < 15



Orden, agregación: ejemplo

SELECT a, count(*), sum(b) as s FROM T GROUP BY a, c
HAVING s < 15

ORDER BY a, s



Vistas

CREATE VIEW *nombre* **AS** SELECT...;

Dan un nombre a una consulta

Equivalente a consulta anidada, pero...

- Útil para reutilizar consultas y simplificar la sintaxis
- Se mantienen siempre actualizadas
- Pueden configurarse para que se almacenen en disco

Ejemplos:

```
CREATE VIEW ContactosUsuario AS
```

```
SELECT u1.nick, u2.nombre FROM Usuario AS u1, Usuario AS u2
```

```
WHERE (u1.nick, u2.nick) IN
```

```
((SELECT usuario1, usuario2 FROM Contacto)
```

```
UNION (SELECT usuario2, usuario1 FROM Contacto));
```

```
SELECT nombre FROM ContactosUsuario WHERE nick = 'pepe';
```

Otros elementos de SQL...

- ◆ Esquemas
 - Para definir espacios de nombres de tablas, similar p.e. a packages Java
- ◆ Dominios
 - Tipos de datos definidos por propiedades y condiciones sobre un tipo primitivo (p.e. una cadena de texto con un cierto formato en expresión regular)
- ◆ Triggers
 - Ejecutar un procedimiento cuando se producen acciones de actualización (insert, update, delete) de una tabla
- ◆ Asserts
 - Checks que pueden hacerse sobre varias filas y varias tablas
- ◆ Transacciones
 - Expresan secuencias de acciones y consultas que deben completarse o cancelarse en bloque
 - Permiten también sincronizar (bloquear) operaciones concurrentes
- ◆ Y muchas más funcionalidades básicas soportadas por cada SGBD extendiendo el estándar SQL...

Comentarios prácticos informales

- ♦ Cuando se incluyen **varias tablas en un from**, normalmente se definirán dos tipos de condiciones en el where
 - Las que pide la consulta
 - Las que conectan unas tablas con otras (no las pide la consulta)
 - Típicamente clave externa = clave primaria
 - Cada tabla se conecta a alguna de las otras, de modo que todas queden conectadas
 - Esto no es obligatorio, pero es habitual para que una consulta tenga sentido
- ♦ Los **alias** se utilizan principalmente...
 - Los de tablas: cuando interviene varias veces la misma tabla en una consulta
 - Los de campos: para utilizar campos definidos por operaciones (típicamente en consultas anidadas, o con agregación)
- ♦ En una versión inicial de una consulta puede no valer la pena utilizar **join**
 - Dejar que el motor de SQL optimice el orden de las operaciones
 - Si se tiene (o cuando se tenga) muy claro el orden óptimo, entonces sí
 - Pueden ser útiles también los join externos en ciertas consultas con agregación

Comentarios prácticos informales (cont)

- ♦ Las **consultas anidadas** son útiles típicamente...
 - Para facilitar operaciones complejas usando p.e. not exists, not in
 - Como vistas, para aligerar consultas complejas y/o reutilizar subconsultas
 - En otros casos a menudo es posible escribir una consulta sin anidamientos
- ♦ Es habitual que una consulta se pueda escribir de diferentes maneras equivalentes
 - Unas formas suelen ser más eficientes que otras
 - En particular, generalmente las consultas con IN se pueden reescribir con EXISTS y viceversa, con EXCEPT cuando están negadas, etc.
- ♦ El orden en que se ejecutan las partes de una consulta es:
 - FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY

Comentarios prácticos informales (cont)

- ♦ Las **consultas anidadas** son útiles típicamente...
 - Para facilitar operaciones complejas usando p.e. not exists, not in
 - Como vistas, para aligerar consultas complejas y/o reutilizar subconsultas
 - En otros casos a menudo es posible escribir una consulta sin anidamientos
- ♦ Es habitual que una consulta se pueda escribir de diferentes maneras equivalentes
 - Unas formas suelen ser más eficientes que otras
 - En particular, generalmente las consultas con IN se pueden reescribir con EXISTS y viceversa, con EXCEPT cuando están negadas, etc.
 - Y cualquier consulta con OR, AND y NOT en el where se puede reescribir como UNION, INTERSECT, EXCEPT
- ♦ El orden en que se ejecutan las partes de una consulta es:
 - FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY