Unidad 6

Afinación del desempeño de una base de datos

Agenda

- Introducción
- Afinación de la base de datos
- Procesamiento de consultas
- Desnormalización
- Mecanismos de afinación de consultas

Introducción

Ciclo de desarrollo de la base de datos

- Definición de necesidades
- Recopilación y análisis de requisitos
- Diseño de la base de datos
  - Diseño Conceptual
  - Diseño Lógico
  - Diseño Físico
- Implementación
- Conversión y carga de datos
- **Pruebas**
- Implantación (puesta en operación)
- Mantenimiento

Incluye la afinación del rendimiento

Introducción

- Sabemos que una vez concluido el diseño de la base de datos (conceptual, lógico, y físico), se procede a materializar el servicio de la base de datos (implementación). Y se realiza la carga inicial de los datos.
- Luego, sobre la base de datos implementada se practican **PRUEBAS** de rendimiento.
- A raíz de las pruebas se podría necesitar realizar ajustes a la implementación de la base de datos. Es lo que se conoce como **AFINACIÓN**.

Introducción

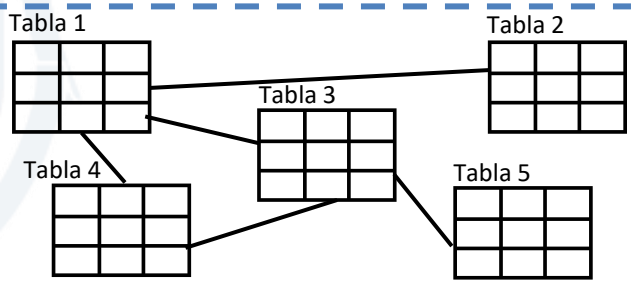
- Si el diseño de la base de datos fue bien hecho, los ajustes en fase de pruebas serán mínimos.
- Si se hizo un mal diseño, los ajustes podrían implicar incluso volver al diseño conceptual, algo no deseable.
- En su mayor parte los problemas de rendimiento de una base de datos se detectan al realizar consultas SQL.
- Ante deficiencias de rendimiento, se suele asumir en primera instancia que el problema es el servidor o la red de datos: ERROR.

Recordemos …

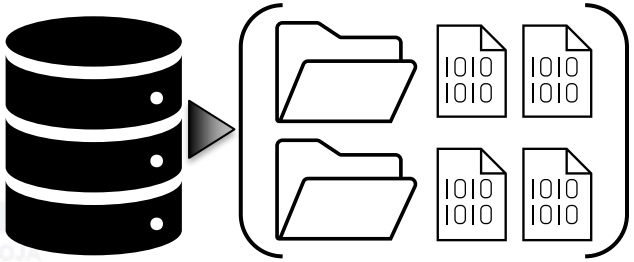
Niveles de abstracción de los datos

Nivel de Vistas → Perspectiva del usuario

Nivel Lógico → Organización lógica de los datos



Nivel Físico → Organización de archivos en almacenamiento secundario



Afinación de la base de datos

- Los ajustes a la implementación de la base de datos se deben realizar en el siguiente orden, según los tipos de causas que originan el problema:
  1. Causas originadas en la aplicación por el mal uso del lenguaje de base de datos (Afinación a nivel de vistas (SQL)).
  2. Causas originadas en las deficiencias del diseño conceptual y/o lógico de la base de datos (Afinación a nivel lógico).
  3. Causas originadas en las deficiencias del diseño físico de la base de datos (Afinación a nivel físico).

Afinación de la base de datos

Afinación a nivel de vistas (SQL)

- Implica evaluar la calidad de estritura de los comandos en el lenguaje de base de datos.
- En su mayor parte los problemas de rendimiento se evidencian en las consultas SQL.
- Se requiere:
  - Conocer el proceso de ejecución de una consulta SQL
  - Conocer y aplicar las buenas prácticas SQL

Afinación de la base de datos

Afinación a nivel lógico

- No es deseable tener que volver a modificar el diseño conceptual y lógico: conlleva corregir también las aplicaciones.
- En el caso de una base de datos relacional implica modificar la estructura de las tablas (columnas, tipos de datos, restricciones, etc.)
- Si hubo algún error en el modelado de los datos se debe corregir lo antes posible. Evitar hacerlo en producción.

Afinación de la base de datos

Afinación a nivel lógico

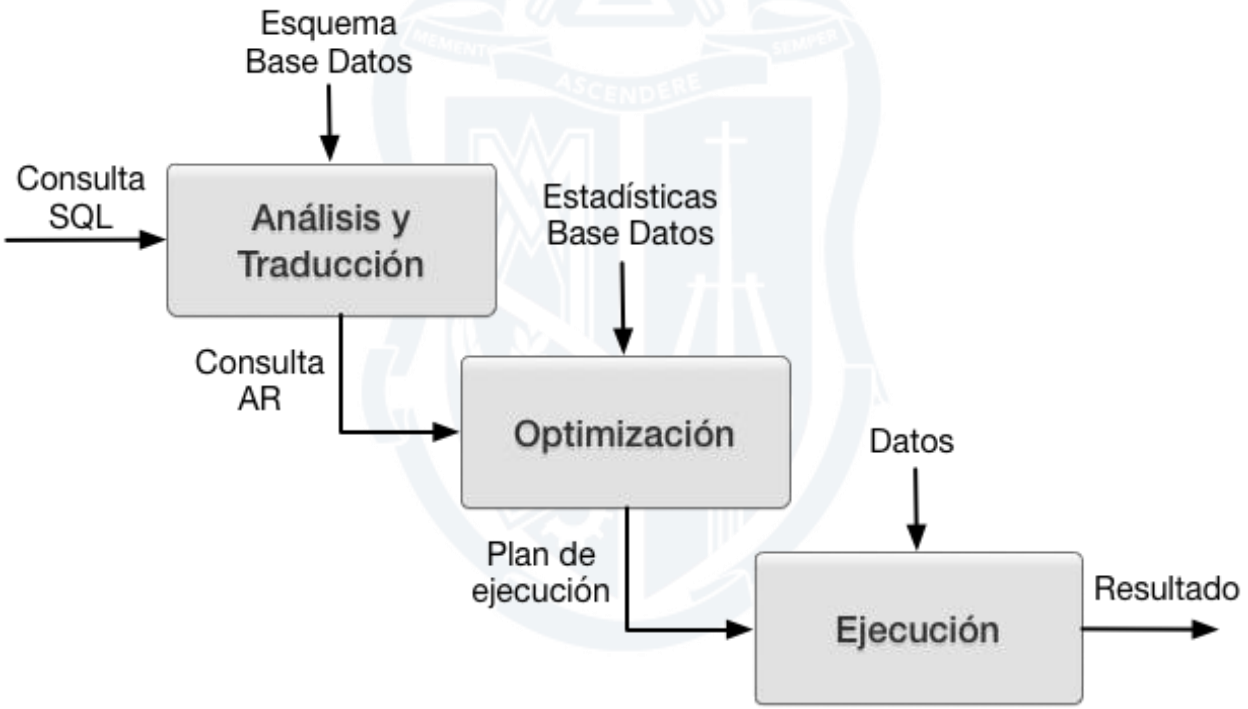
- Otras de las acciones que se suele realizar, es introducir redundancia al modelo. Lo que se conoce como **desnormalización**.
- La desnormalización busca mejorar los tiempos de respuesta, reduciendo el número de tablas y operaciones involucradas.
- En una base de datos transaccional, la desnormalización debe ser en lo posible evitada.

Afinación de la base de datos

Afinación a nivel físico

- Implica revisar la implementación física de la base de datos.
- Acciones en orden de prioridad:
  - Agregar nuevos **índices** secundarios
  - Ajustar configuración y **parametrización del SGBD** (para aprovechar mejor los recursos del servidor)
  - **Potenciar el servidor** (aumentar memoria, disco, o capacidad de procesamiento)
  - **Replantear infraestructura** (comprar nuevo servidor, realizar balanceo de carga con dos o más servidores, implementar almacén de datos para generación de informes)

Procesamiento de consultas



Procesamiento de consultas

- Se parte de la consulta en lenguaje de alto nivel (SQL por ejemplo).
- Pasos:
  - **Análisis y traducción**: Su resultado es la consulta reestructurada convertida a algebra relacional.
  - **Optimización**: Selección de plan de ejecución de la consulta en base a estadísticas. Termina expresado en lenguaje de bajo nivel con instrucciones adicionales (accesos a disco requeridos, índices a usar, etc.).
  - **Ejecución**: Ejecución del plan y obtención de datos resultantes.

Procesamiento de consultas

Análisis y traducción (Pasos)

1. **Análisis léxico y sintáctico**: verifica palabras reservadas, nombres de identificadores, estructura de expresiones y de sentencia SQL, compatibilidad de tipos, etc.
2. **Análisis semántico**: verifica que las condiciones de selección no sean contradictorias, que no sean contrapongan a las definiciones de dominio e integridad. Que no existan condiciones contenidas en otras.
3. **Normalización y simplificación**: aplica conversiones lógicas, reduce y simplifica expresiones booleanas, ejemplo:

   NOT (a > b) equivale a **(a <= b)**

   (a > 10 AND a < 5) equivale a **(FALSO)**

   NOT(a) OR NOT(b) equivale a **NOT (a AND b)**
4. **Traducción a algebra relacional**.

Procesamiento de consultas

Análisis y traducción (Ejemplo)

Dada la tabla PELICULAS:

peliculas (id_pelicula, nombre_original, tipo_guion, anio, sinopsis, duracion_minutos)

Deseamos un reporte en el que se incluyan aquellos filmes **originales,** y **adaptados a partir de una obra literaria,** y además producidos en la **década de los 90**.

Para tal efecto se propone la siguiente sentencia SQL :

```
SELECT *
FROM peliculas p
WHERE p.tipo_guion = 'ORIGINAL'
  AND p.tipo_guion = 'A.LITERARIA'
  AND p.anio >= 1990
   OR p.anio < 1999;
```

Procesamiento de consultas

Análisis y traducción

1. Análisis léxico - sintáctico:

```
SELECT *
FROM peliculas p
WHERE p.tipo_guion = 'ORIGINAL'
  AND p.tipo_guion = 'A.LITERARIA'
  AND p.anio >= 1990
   OR p.anio < 1999;
```

¿La consulta es sintácticamente correcta?

Procesamiento de consultas

Análisis y traducción

1. Análisis léxico - sintáctico:

```
SELECT *
FROM peliculas p
WHERE p.tipo_guion = 'ORIGINAL'
  AND p.tipo_guion = 'A.LITERARIA'
  AND p.anio >= 1990
   OR p.anio < 1999;
```

¿La consulta es sintácticamente correcta?

SI

No contiene errores, sintácticamente está OK

Procesamiento de consultas

Análisis y traducción

2. Análisis semántico:

```
SELECT *
FROM peliculas p
WHERE p.tipo_guion = 'ORIGINAL'
  AND p.tipo_guion = 'A.LITERARIA'
  AND p.anio >= 1990
   OR p.anio < 1999;
```

¿La consulta es semánticamente correcta?

Procesamiento de consultas

Análisis y traducción

2. Análisis semántico:

```
SELECT *
FROM peliculas p
WHERE p.tipo_guion = 'ORIGINAL'
  AND p.tipo_guion = 'A.LITERARIA'
  AND p.anio >= 1990
   OR p.anio < 1999;
```

(tipo_guion = 'ORIGINAL' AND tipo_guion = 'A.LITERARIA')
son condiciones contradictorias

Procesamiento de consultas

Análisis y traducción

3. Simplificación:

```
SELECT *
FROM peliculas p
WHERE p.tipo_guion = 'ORIGINAL'
  AND p.tipo_guion = 'A.LITERARIA'
  AND p.anio >= 1990
   OR p.anio < 1999;
```

---

tipo_guion = 'ORIGINAL' AND tipo_guion = 'A.LITERARIA' AND p.anio >= 1990 OR p.anio < 1999

↓

(((tipo_guion = 'ORIGINAL' AND tipo_guion = 'A.LITERARIA') AND p.anio >= 1990) OR p.anio < 1999)

↓

((FALSE AND p.anio >= 1990) OR p.anio < 1999)

↓

(FALSE OR p.anio < 1999)

↓

p.anio < 1999

Procesamiento de consultas

Análisis y traducción

3. Simplificación:

```
SELECT *
FROM peliculas p
WHERE p.tipo_guion = 'ORIGINAL'
  AND p.tipo_guion = ' A.LITERARIA'
  AND p.anio >= 1990
   OR p.anio < 1999;
```

```
SELECT *
FROM peliculas p
WHERE p.anio < 1999;
```

Procesamiento de consultas

Análisis y traducción

4. Traducción a algebra relacional

```
SELECT *
FROM peliculas p
WHERE p.anio < 1999;
```

$\sigma_{p.anio < 1999}(PELICULAS)$

Podrían existir varias representaciones de la consulta en algebra relacional, cada una corresponde a un posible plan de ejecución que se evalúa en la fase de **OPTIMIZACIÓN**

Procesamiento de consultas

Análisis y traducción

CONCLUSION:

La consulta planteada SINTÁCTICAMENTE ES CORRECTA pero SEMÁNTICAMENTE INCORRECTA.

- Nunca puede ocurrir que el tipo de guión sea a la vez ORIGINAL y A.LITERARIA.
- Se incluirían todas las películas producidas en antes de 1999, inclusive aquellas producidas antes de 1990 y no es lo que se quiere
- No incluiría las producidas en 1999, y que si son parte del requerimiento

Lo correcto sería entonces:

Reporte en el que se incluyan aquellos filmes **originales**, **adaptados a partir de una obra literaria**, y además producidos en la década de los 90.

```
SELECT *
FROM peliculas p
WHERE (p.tipo_guion = 'ORIGINAL'
   OR p.tipo_guion = 'A.LITERARIA')
  AND p.anio >= 1990
  AND p.anio <= 1999;
```

Desnormalización

- **Normalizar**: Optimizar el diseño para eliminar redundancia de la base de datos.
- **Desnormalizar**: Modificar el diseño para introducir redundancia en la base de datos.

id	nombre	puesto	salario	emails
111	Juan Pérez	Jefe de Área	3000	juanp@ecn.es; jefe2@ecn.es
222	José Sánchez	Administrativo	1500	jsanchez@ecn.es
333	Ana Díaz	Administrativo	1500	adiaz@ecn.es; ana32@gmail.com
...	...	...	...	...

DESNORMALIZAR

NORMALIZAR

T1

id	nombre	puesto
111	Juan Pérez	Jefe de Área
222	José Sánchez	Administrativo
333	Ana Díaz	Administrativo
...	...	...

T3

puesto	salario
Jefe de Área	3000
Administrativo	1500
...	...

T2

id	email
111	juanp@ecn.es
111	jefe2@ecn.es
222	jsanchez@ecn.es
333	adiaz@ecn.es
333	ana32@gmail.com
...	...

Desnormalización

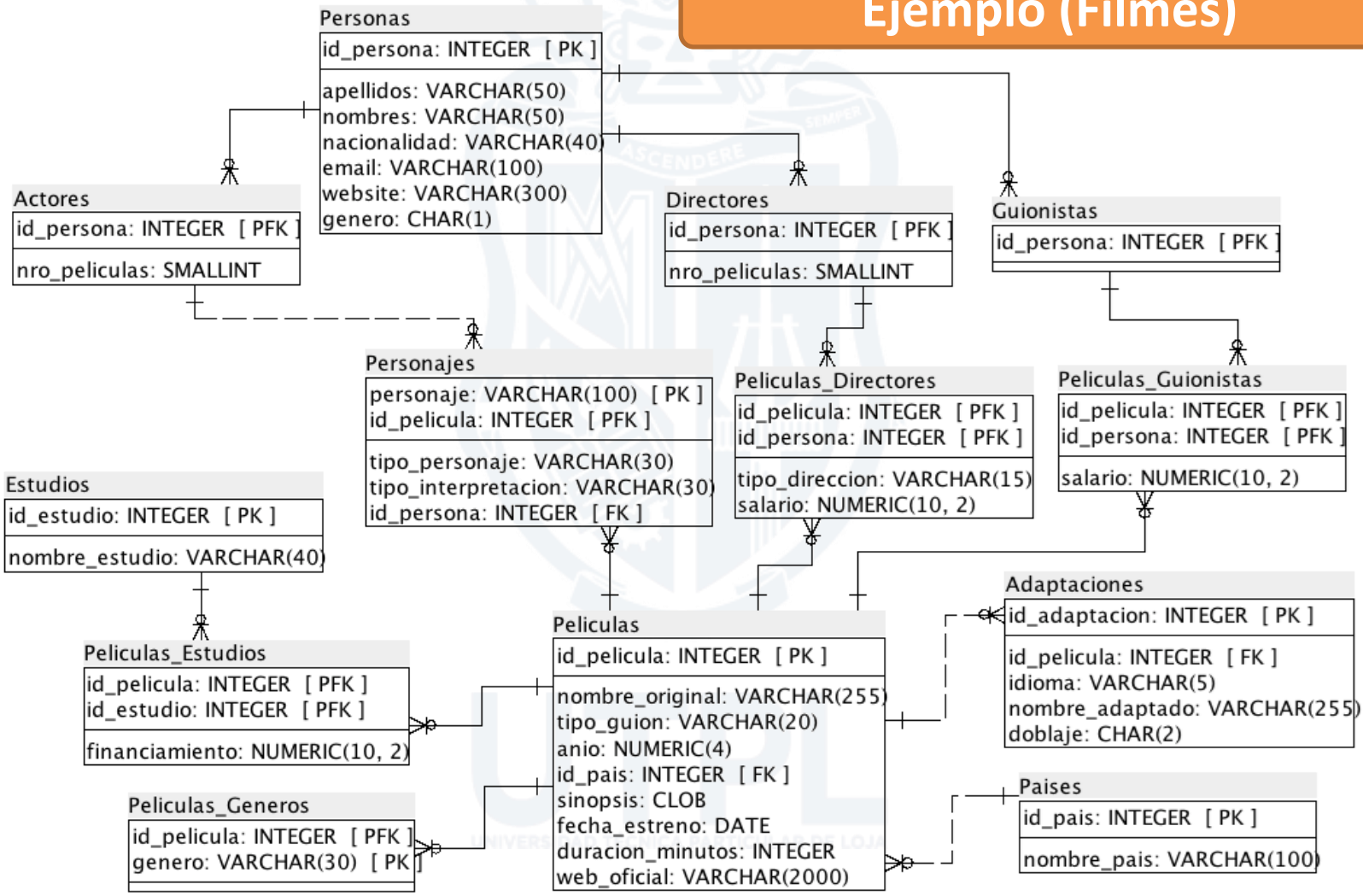
El objetivo es reducir los cálculos y/o la cantidad de tablas que requieren acceder en una consulta SQL

Desnormalización

- Técnicas comunes:
  - Combinación de tablas con asociación 1:1
  - Duplicidad de atributos que no forman parte de la clave en asociaciones 1:N
  - Duplicidad de atributos en asociaciones M:N
  - Inclusión de atributos derivados
- Técnicas adicionales:
  - Atributos multivaluados
  - Particionamiento de tablas.
  - Tablas de extracción (DataWarehouse)

Desnormalización

Ejemplo (Filmes)



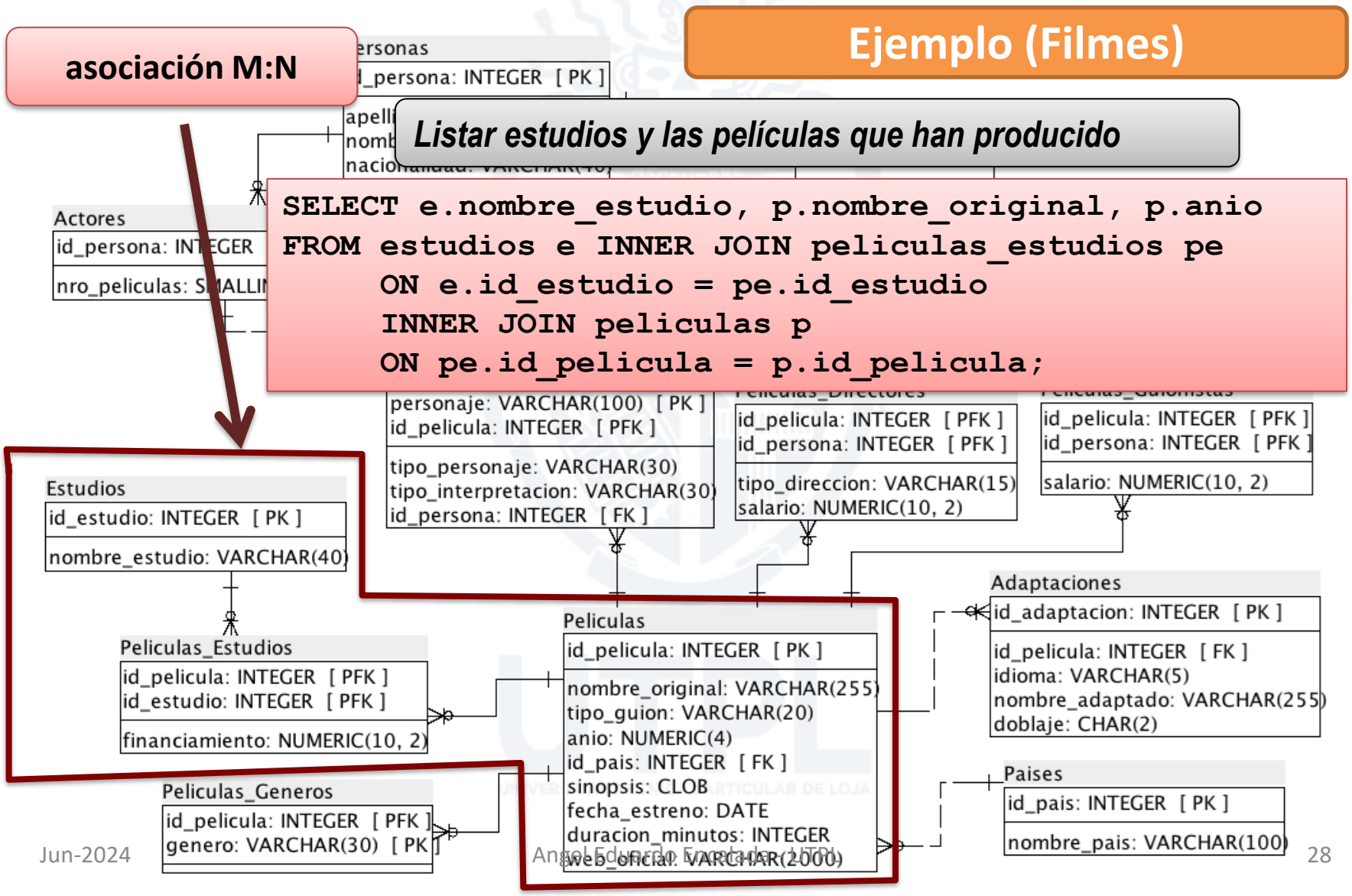
Desnormalización

asociación M:N

Ejemplo (Filmes)

Listar estudios y las películas que han producido

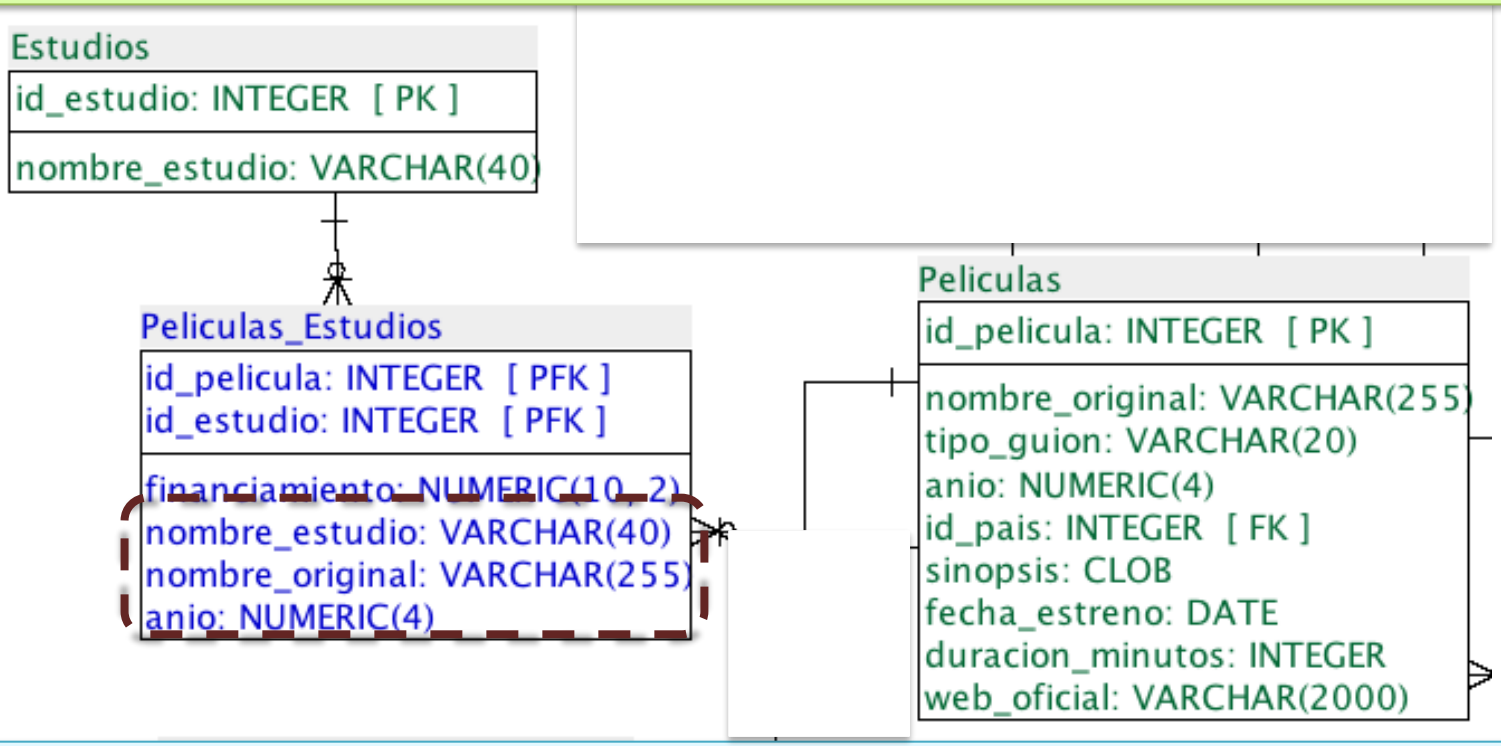
```
SELECT e.nombre_estudio, p.nombre_original, p.anio
FROM estudios e INNER JOIN peliculas_estudios pe
     ON e.id_estudio = pe.id_estudio
     INNER JOIN peliculas p
     ON pe.id_pelicula = p.id_pelicula;
```



Desnormalización

Ejemplo (Filmes)

Duplicidad de atributos en asociaciones M:N




```
SELECT pe.nombre_estudio, pe.nombre_original, pe.anio
FROM peliculas_estudios pe;
```

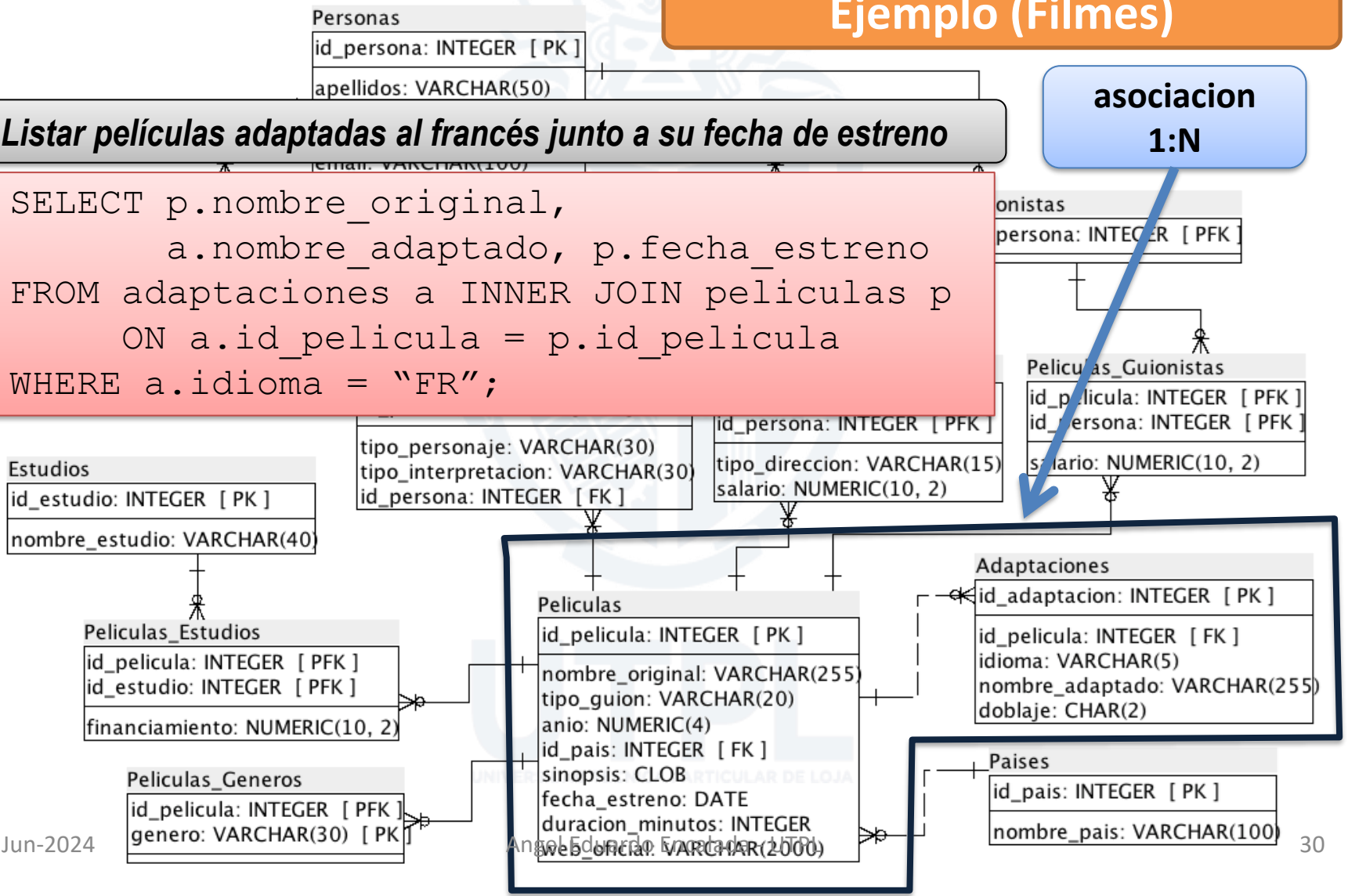
Desnormalización

Ejemplo (Filmes)

Listar películas adaptadas al francés junto a su fecha de estreno

```
SELECT p.nombre_original,
       a.nombre_adaptado, p.fecha_estreno
FROM adaptaciones a INNER JOIN peliculas p
     ON a.id_pelicula = p.id_pelicula
WHERE a.idioma = "FR";
```

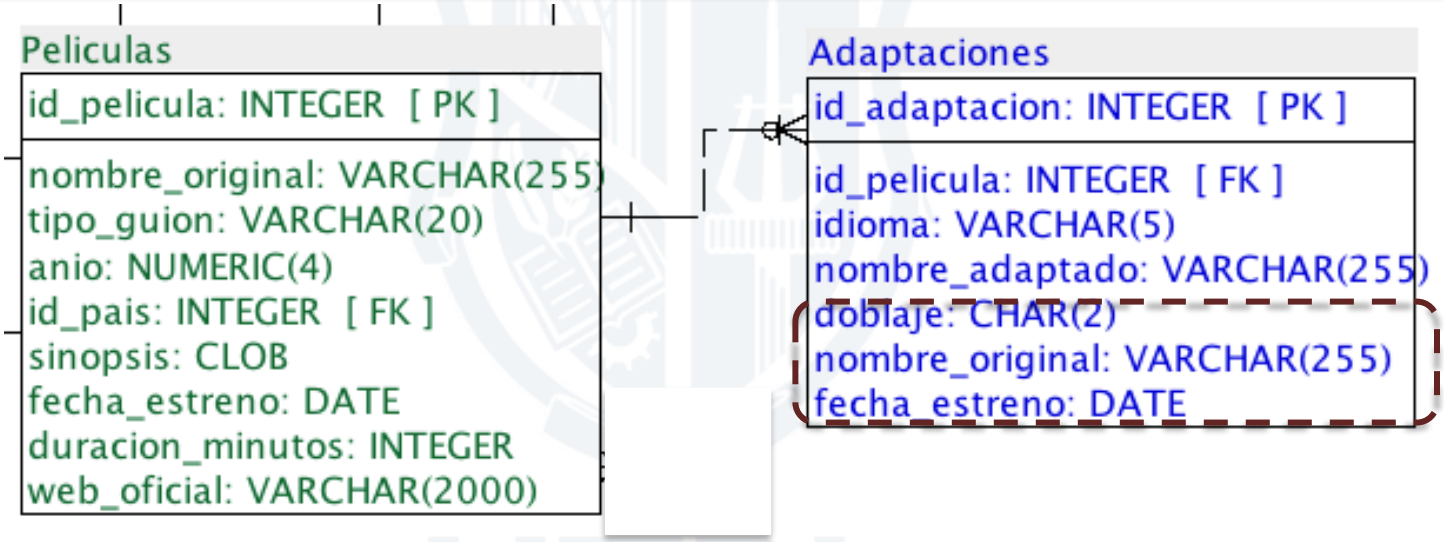
asociacion 1:N



Desnormalización

Ejemplo (Filmes)

Duplicidad de atributos que no forman parte de la clave en asociaciones 1:N

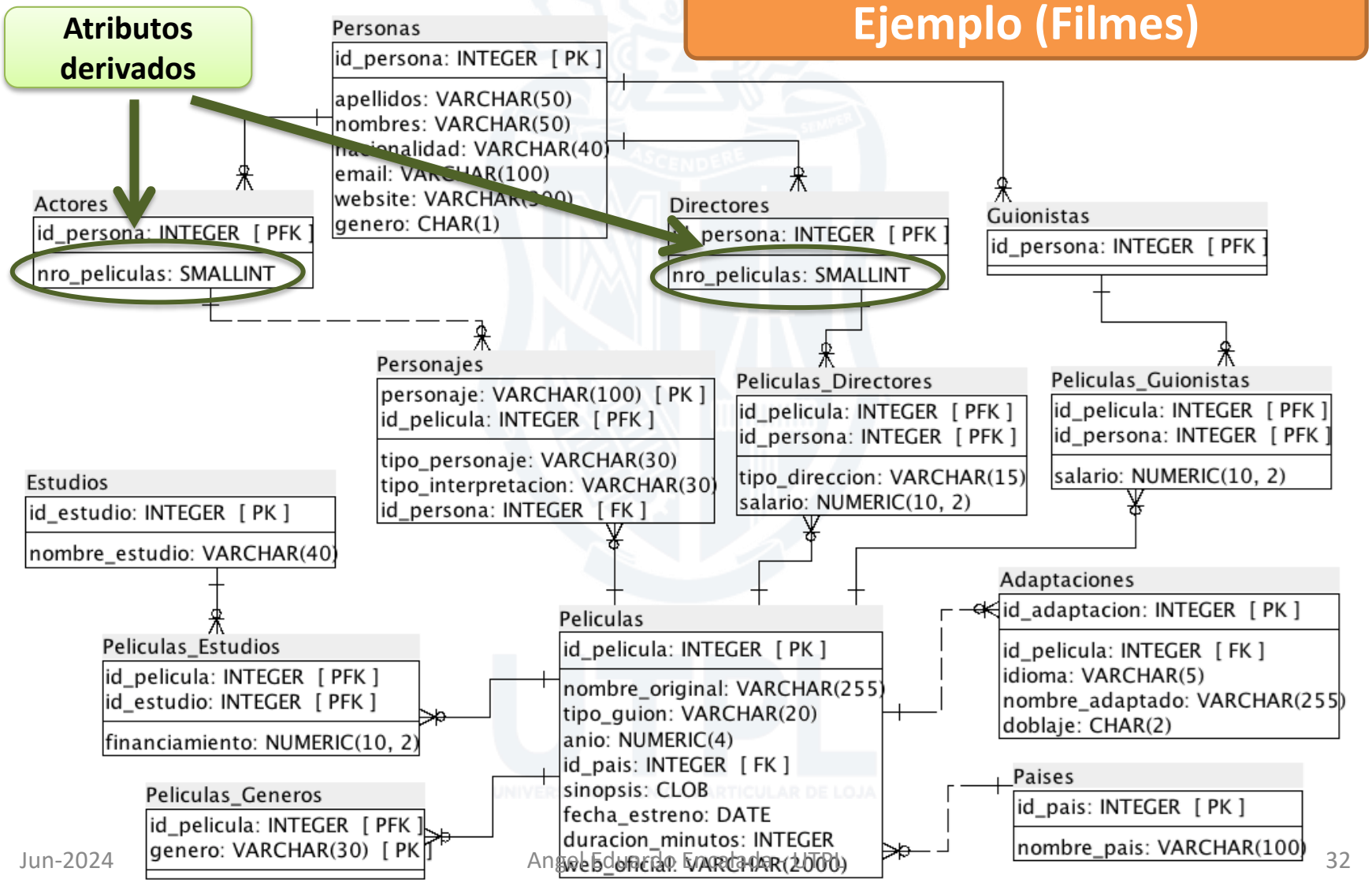



```
SELECT a.nombre_original, a.nombre_adaptado, a.fecha_estreno
FROM adaptaciones a
WHERE a.idioma = "FR";
```

Desnormalización

Atributos derivados

Ejemplo (Filmes)



Desnormalización

Ejemplo (Filmes)

Inclusión de atributos derivados

Sin atributos derivados:

```
SELECT p.apellidos, p.nombres,
       COUNT(distinct f.id_pelicula) nro_peliculas
FROM personas p INNER JOIN directores d
    ON p.id_persona = d.id_persona
    INNER JOIN peliculas_directores f
    ON d.id_persona = f.id_persona
GROUP BY p.apellidos, p.nombres;
```

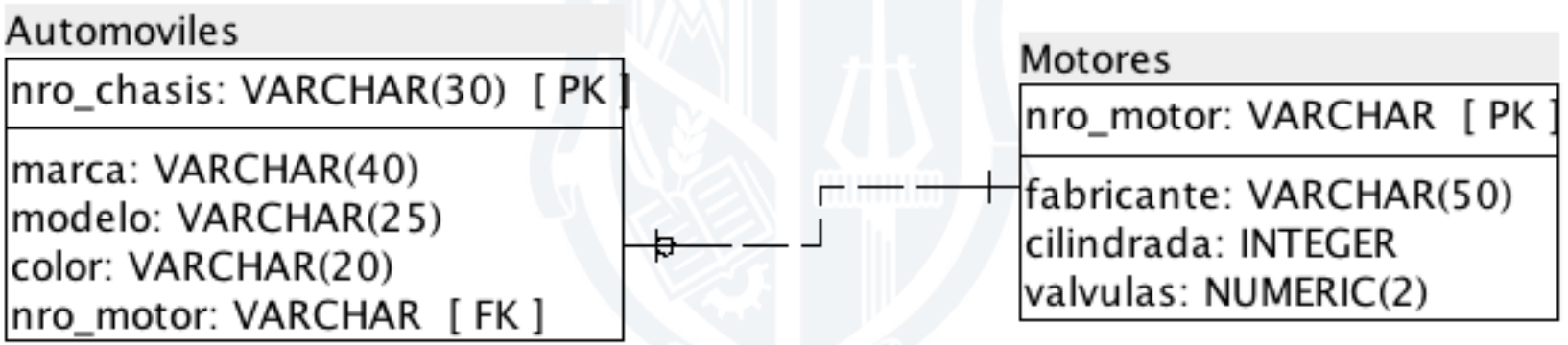
CON atributos derivados:

```
SELECT p.apellidos, p.nombres, d.nro_peliculas
FROM personas p INNER JOIN directores d
    ON p.id_persona = d.id_persona;
```

Desnormalización

Ejemplo (Filmes)

asociacion 1:1

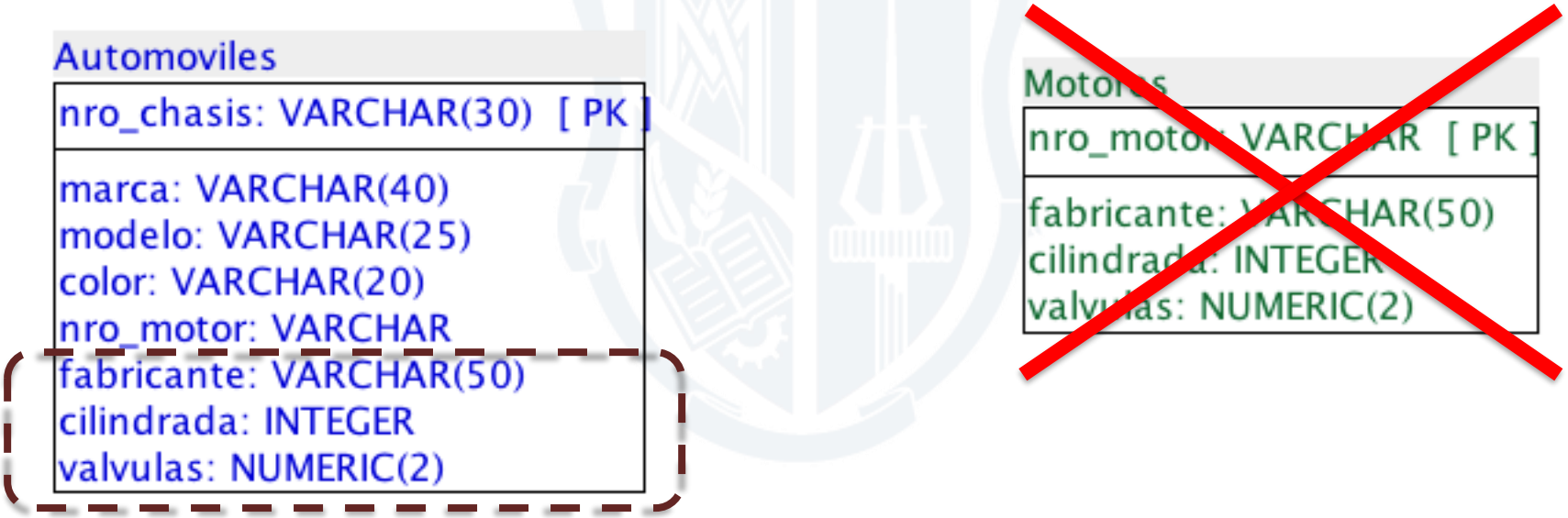



```
SELECT a.nro_chasis, a.marca, a.modelo, m.cilindrada
FROM automoviles a INNER JOIN motores m
    ON a.nro_motor = m.nro_motor;
```

Desnormalización

Ejemplo (Filmes)

Combinación de tablas con asociación 1:1



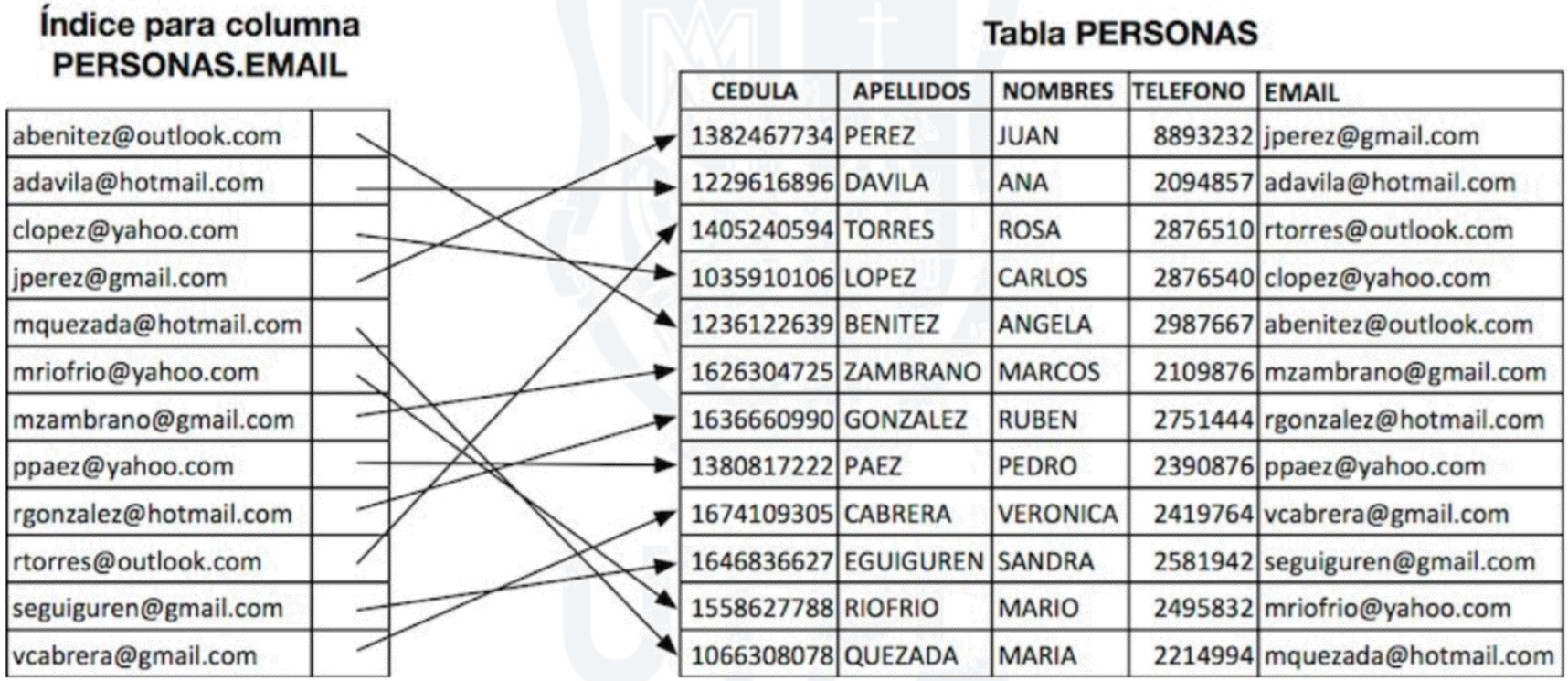

```
SELECT a.nro_chasis, a.marca, a.modelo, a.cilindrada
FROM automoviles a;
```

Desnormalización

- Consideraciones importantes
  - Busca mejorar tiempos en operaciones de consulta.
  - Implica descender a 2FN o 1FN (tablas con alta redundancia de datos).
  - Tener cuidado que la redundancia no implique demoras en actualizaciones.
  - Puede utilizarse vistas para reducir complejidad de consultas (cuando el problema no es rendimiento).
  - **En bases de datos transaccionales la desnormalización debe usarse como último recurso.**

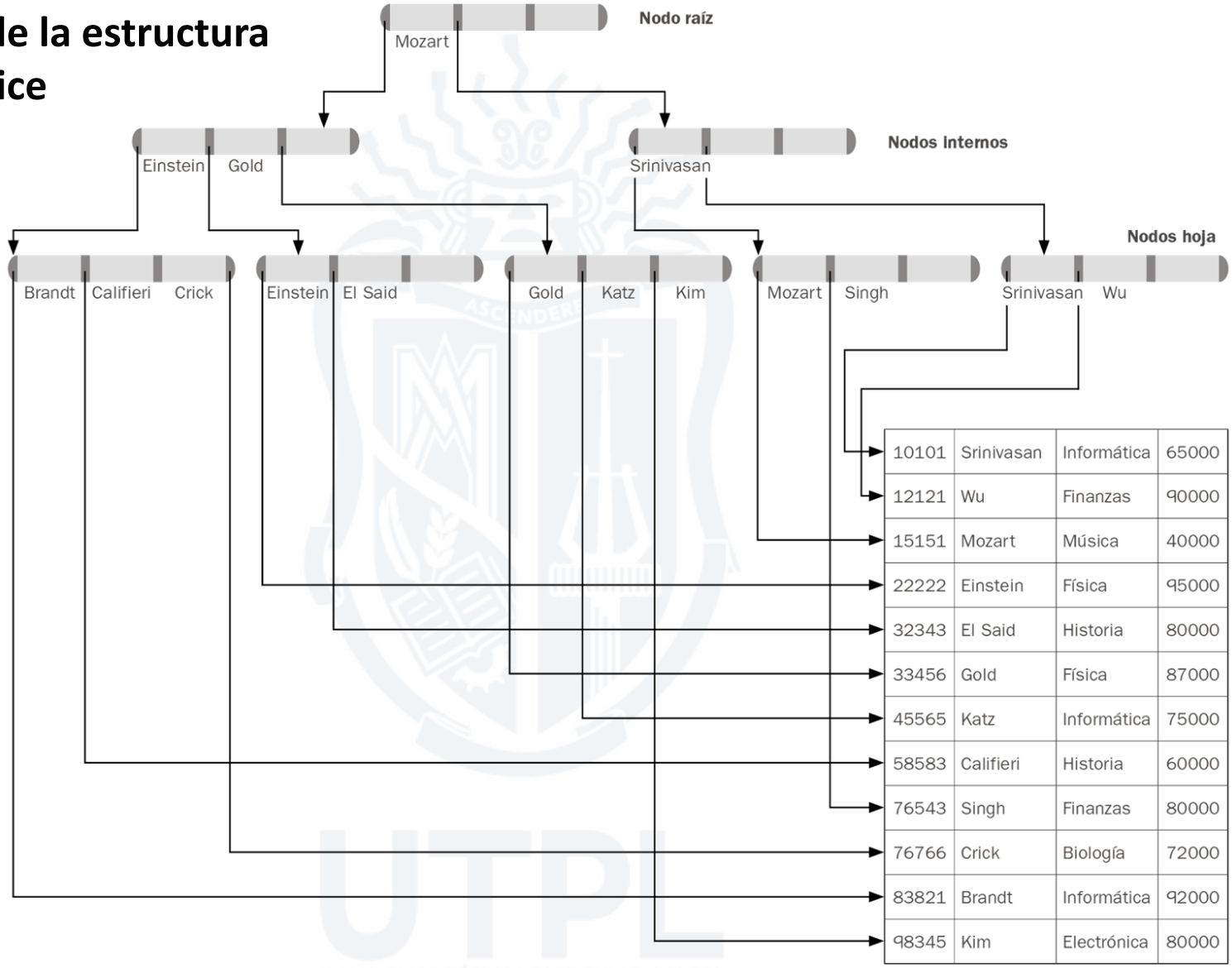
Índices

Representación de un índice



CEDULA	APELLIDOS	NOMBRES	TELEFONO	EMAIL
1382467734	PEREZ	JUAN	8893232	jperez@gmail.com
1229616896	DAVILA	ANA	2094857	adavila@hotmail.com
1405240594	TORRES	ROSA	2876510	rtorres@outlook.com
1035910106	LOPEZ	CARLOS	2876540	clopez@yahoo.com
1236122639	BENITEZ	ANGELA	2987667	abenitez@outlook.com
1626304725	ZAMBRANO	MARCOS	2109876	mzambrano@gmail.com
1636660990	GONZALEZ	RUBEN	2751444	rgonzalez@hotmail.com
1380817222	PAEZ	PEDRO	2390876	ppaez@yahoo.com
1674109305	CABRERA	VERONICA	2419764	vcabrera@gmail.com
1646836627	EGUIGUREN	SANDRA	2581942	seguiguren@gmail.com
1558627788	RIOFRIO	MARIO	2495832	mriofrio@yahoo.com
1066308078	QUEZADA	MARIA	2214994	mquezada@hotmail.com

Ejemplo de la estructura de un índice



10101	Srinivasan	Informática	65000
12121	Wu	Finanzas	90000
15151	Mozart	Música	40000
22222	Einstein	Física	95000
32343	El Said	Historia	80000
33456	Gold	Física	87000
45565	Katz	Informática	75000
58583	Califieri	Historia	60000
76543	Singh	Finanzas	80000
76766	Crick	Biología	72000
83821	Brandt	Informática	92000
98345	Kim	Electrónica	80000

Fuente: Silberschatz et al. 2014

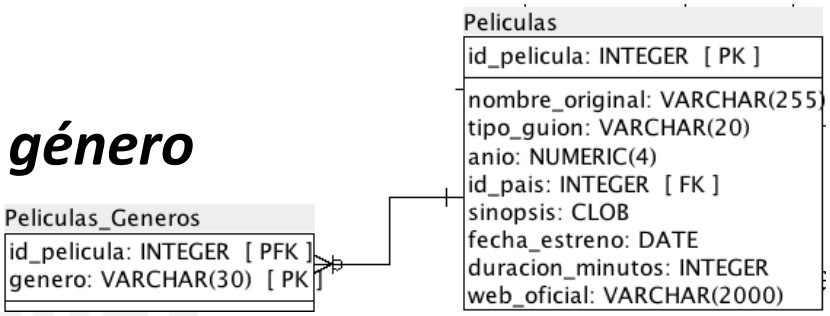
Índices

- A los índices se los identifica con base en:
  - Atributos que son más usados en operaciones de consulta para **filtrar, combinar, ordenar o agrupar** información.
  - Llaves primarias (PK) siempre tienen un índice asociado (El SGBD lo crea automáticamente).
  - Para llaves foráneas se recomienda siempre crear un índice (algunos SGBD también lo hacen de forma automática)
  - No todas las operaciones de consulta requieren filtrar, combinar, ordenar o agrupar información.

Índices

Ejemplo transacción:

Listar las películas según el género



Consulta SQL:

```
SELECT p.nombre_original, p.anio, p.fecha_estreno
FROM peliculas p INNER JOIN peliculas_generos g
    ON p.id_pelicula = g.id_pelicula
WHERE g.genero = "DRAMA"
ORDER BY p.anio, p.nombre_original;
```

Columnas usadas para **combinar, filtrar, ordenar o agrupar**:

- peliculas.**id_pelicula**
- peliculas_generos.**idpelicula**
- peliculas_generos.**genero**
- peliculas.**anio**
- peliculas.**nombre_original**

Posibles **índices secundarios** a crear:

- peliculas_generos.**genero**
- peliculas.**anio**
- peliculas.**nombre_original**

Índices

```
CREATE INDEX pel_gen_genero_idx
          ON peliculas_generos (genero);

CREATE INDEX pel_anio_idx
          ON peliculas (anio);

CREATE INDEX pel_nombre_original_idx
          ON peliculas (nombre_original);
```

Índices

Recomendaciones

- En principio se crearían índices para todas las columnas candidatas.
- Posteriormente durante la etapa de monitoreo se evaluará si hay algún índice que está de más, o si falta alguno.
- Cuando las tablas son muy pequeñas en cuanto a volumen de datos, se recomienda no crear índices (Por ejemplo Países en nuestro caso).
- En BBDD transaccionales se debe reducir el uso de índices al mínimo necesario. Sobre todo en columnas de tablas que se actualizan con mucha frecuencia.
- Los índices aceleran las consultas, pero ralentizan las actualizaciones

Buenas prácticas SQL

- Las buenas prácticas SQL son normas o directrices que se deben considerar al escribir SQL.
- Se busca obtener un código ordenado, legible, entendible, y sobre todo EFICIENTE.
- Pueden ayudar en mucho a disminuir la carga al motor, y con ello tener un servicio más eficiente.
- Sobre todo se aplican a consultas (comando SELECT)

Buenas prácticas SQL

Regla N° 1

Traer únicamente los datos que necesitamos.

Evitar el SELECT *

Buenas prácticas SQL

- Evitar productos cartesianos.

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e CROSS JOIN oficina o;
```

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e INNER JOIN oficina o
     ON e.codigo_oficina = o.codigo_oficina;
```

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e, oficina o;
```

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e, oficina o
WHERE e.codigo_oficina = o.codigo_oficina;
```

Buenas prácticas SQL

- Combinar en la cláusula FROM (JOIN)

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e, oficina o
WHERE e.codigo_oficina = o.codigo_oficina
  AND e.apellido1 like "S%"
ORDER BY e.email;
```

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e INNER JOIN oficina o
     ON e.codigo_oficina = o.codigo_oficina
WHERE e.apellido1 like "S%"
ORDER BY e.email;
```

Buenas prácticas SQL

- Cualificar los nombres de columna en consultas multitabla

```
SELECT d.department_id, d.department_name,
       COUNT(*) as "Total empleados",
       COUNT(e.commission_pct) as "Empleados con comisión",
       COUNT(*) - COUNT(e.commission_pct) as "Empleados sin comisión"
FROM employees e
     INNER JOIN departments d ON e.department_id = d.department_id
WHERE e.salary >= 4000
GROUP BY d.department_id, d.department_name
HAVING count(*) > 5;
```

Buenas prácticas SQL

- En subconsultas, si es posible, usar IN en lugar de NOT IN

```
SELECT e.employee_id, e.last_name
FROM employees e
WHERE e.department_id NOT IN (SELECT DISTINCT x.department_id
                              FROM employees x
                              WHERE e.salary <= 5000);
```

```
SELECT e.employee_id, e.last_name
FROM employees e
WHERE e.department_id IN (SELECT DISTINCT x.department_id
                          FROM employees x
                          WHERE e.salary > 5000);
```

Buenas prácticas SQL

- Usar EXISTS en lugar de IN

```
SELECT *
FROM clientes
WHERE ciudad_res IN (SELECT ciudad
                     FROM sucursales);
```

```
SELECT *
FROM clientes c
WHERE EXISTS (SELECT 1
              FROM sucursales s
              WHERE s.ciudad = c.ciudad_res);
```

Buenas prácticas SQL

- Usar LIKE al final de las condiciones, ya cuando se hayan filtrado una parte de los registros

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e INNER JOIN oficina o
     ON e.codigo_oficina = o.codigo_oficina
WHERE e.apellido1 LIKE "%a%"
  AND o.region = "Madrid";
```

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e INNER JOIN oficina o
     ON e.codigo_oficina = o.codigo_oficina
WHERE o.region = "Madrid"
  AND e.apellido1 LIKE "%a%";
```

Buenas prácticas SQL

- Evitar DISTINCT si no es necesario.

```
SELECT DISTINCT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e INNER JOIN oficina o
     ON e.codigo_oficina = o.codigo_oficina
WHERE e.apellido1 LIKE "S%"
ORDER by e.email;
```

```
SELECT e.apellido1, e.nombre, e.email, o.ciudad
FROM empleado e INNER JOIN oficina o
     ON e.codigo_oficina = o.codigo_oficina
WHERE e.apellido1 LIKE "S%"
ORDER by e.email;
```

Buenas prácticas SQL

En lugar de	Es mejor
`PRECIO > COSTO * 1.12`	`PRECIO > COSTO_CON_IVA`
`CODIGO = '1285'`	`CODIGO = 1285`
`GENERO <> 'F'`	`GENERO = 'M'`
`TIPO <> 'REPUESTO' AND DESCUENTO = 0.2 AND PRECIO > 200`	`DESCUENTO = 0.2 AND PRECIO > 200 AND TIPO <> 'REPUESTO'`

Buenas prácticas SQL

En lugar de	Es mejor
`TIPO = 'UNIFAMILIAR' AND BARRIO = 'SAUCES'`	`BARRIO = 'SAUCES' AND TIPO = 'UNIFAMILIAR'`
`BARRIO = 'SAUCES' OR TIPO = 'UNIFAMILIAR'`	`TIPO = 'UNIFAMILIAR' OR BARRIO = 'SAUCES'`
`NOT (PRECIO > 500 AND MARCA = 'LG')`	`PRECIO <= 500 OR MARCA <> 'LG'`

Buenas prácticas SQL

- Paginar:

MySQL

```
SELECT employee_id, last_name, first_name
FROM employees
ORDER BY 3,4
LIMIT 20 OFFSET 40;
```

Oracle

```
SELECT *
FROM (SELECT rownum rn, x.*
      FROM (SELECT employee_id, last_name, first_name
            FROM employees
            ORDER BY 2,3) x)
WHERE rn BETWEEN 41 AND 60;
```

Buenas prácticas SQL

- Ordenar solo si es necesario
- Si hay un índice compuesto, en la condición WHERE ubicar los campos en el mismo orden que en el índice.

Buenas prácticas SQL

Para lograr un rendimiento adecuado del servicio de la base de datos, es fundamental el uso correcto del lenguaje SQL.