Diseño y Administración

de Bases de Datos

Cuadernos de Prácticas de Laboratorio

Colección manuales uex - 100

Manuel
Barrena García

100

DISEÑO Y ADMINISTRACIÓN
DE BASES DE DATOS

MANUALES UEX

100

MANUEL BARRENA GARCÍA

DISEÑO Y ADMINISTRACIÓN DE BASES DE DATOS

Cuadernos de Prácticas de Laboratorio

2015

Cualquier forma de reproducción, distribución, comunicación pública o transformación de esta obra solo puede ser realizada con la autorización de sus titulares, salvo excepción prevista por la ley. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, www.cedro.org) si necesita fotocopiar o escanear algún fragmento de esta obra.

© Manuel Barrena García, para esta edición.
© Universidad de Extremadura, para esta edición.

Edita:

Universidad de Extremadura. Servicio de Publicaciones
C/ Caldereros, 2 - Planta 2ª. 10071 Cáceres (España)
Tel. 927 257 041 ; Fax 927 257 046
E-mail: publicac@unex.es
http://www.unex.es/publicaciones

ISSN 1135-870-X
ISBN de méritos 978-84-606-9523-3

Impreso en España - Printed in Spain
Maquetación, fotomecánica e impresión: Dosgraphic, S.L. – 914 786 125

ÍNDICE GENERAL

ÍN DI CE

LABORATORIO 1. DISEÑO LÓGICO IMDB 9

1. ¿Qué es IMDB? 9
2. Análisis y colección de requisitos 10
3. Modelo conceptual 10
4. Descripción de entidades y asociaciones 11
5. Ejercicios 18

LABORATORIO 2. MATERIALIZACIÓN DE IMDB 19

1. Instalación y primeros pasos con MySql 19
2. Creación de la base de datos myimdb 20
3. Ejercicios 26

LABORATORIO 3. CONSULTAS SOBRE IMDB 27

1. Introducción 27
2. Consultando el diccionario 27
3. Consultas de agrupación y funciones agregadas 30
4. Uso de subconsultas 31
5. Consultas mediante join 34
6. Ejercicios Finales. Consultas sobre IMDB 37

LABORATORIO 4. CONSTRUCCIÓN DE ÍNDICES SOBRE IMDB 43

1. Introducción 43
2. El coste de una consulta simple 45
3. Cómo elegir un índice 46
4. La importancia del acceso a la tabla base 50
5. Ejercicios finales 53

ÍNDICE

LABORATORIO 5. OPTIMIZACIÓN DE CONSULTAS 55

1. Introducción 55
2. Ejecución de JOINs en Mysql 56
3. Optimización del JOIN mediante índices 56
4. Ejercicios finales 62

LABORATORIO 6. LA SENTENCIA EXPLAIN 63

1. Introducción a EXPLAIN 63
2. El campo ID 64
3. El campo SELECT_TYPE 65
4. El campo TABLE 68
5. El campo TYPE 69
6. KEY y ROWS 73
7. POSSIBLE_KEYS, KEY_LEN y REF 76
8. El campo EXTRA 78

LABORATORIO 7. PRACTICANDO LA OPTIMIZACIÓN DE CONSULTAS EN IMDB 81

1. Introducción 81
2. Informes de rendimiento 82
3. Ejercicios de optimización 85

LABORATORIO 1.
DISEÑO LÓGICO IMDB

Objetivos. Tras cubrir esta sesión práctica, el alumno/a ha adquirido destrezas para abordar el diseño conceptual de una base de datos y definir su modelo lógico a partir del análisis previo de la información relacionada.

1. ¿QUÉ ES IMDB?

IMDB es el acrónimo de Internet Movie Data Base, la mayor plataforma en Internet que aglutina información sobre obras cinematográficas en distintos formatos (cine, televisión, videojuegos, etc.). La página web www.imdb.com, de acceso público, proporciona información sobre la práctica totalidad de obras cinematográficas a disposición del público en general. Esta plataforma se sostiene sobre una enorme base de datos que la propia plataforma hace parcialmente pública para un uso no comercial.

La principal razón para el uso de la base de datos IMDB en este laboratorio es servir como herramienta experimental para conseguir muchas de las competencias específicas establecidas en la asignatura DABD. El uso de IMDB nos permitirá abordar en profundidad la mayor parte de la temática relativa al diseño y la administración de las bases de datos, comenzando por el diseño conceptual, avanzando hacia los aspectos de diseño lógico y físico de las bases de datos y finalizando aspectos de vital importancia en las tareas de administración de las bases de datos.

El objetivo de la parte de laboratorio de esta asignatura será el diseño y administración de una base de datos que sustente de modo eficiente y por supuesto efectivo el acceso por parte del usuario a una plataforma similar a www.imdb.com.

2. ANÁLISIS Y COLECCIÓN DE REQUISITOS

La fase de diseño de una base de datos se debe sustentar en un análisis previo de la realidad de la empresa u organización que desea o precisa la utilización de un SGBD. Como sabemos, este paso previo al diseño conlleva una serie de tareas definidas por un método riguroso que facilita la comprensión de la realidad que rodea al sistema. El primer paso en nuestro análisis consistiría en llevar a cabo una exploración detallada a los contenidos de la plataforma www.imdb.com. Este recorrido nos debe aportar una amplia perspectiva sobre los datos que necesitaremos almacenar y los tipos de operaciones a los que ha de enfrentarse nuestra base de datos.

La información contenida en la plataforma web IMDB es tan extensa que nos llevaría mucho más tiempo del que disponemos para finalmente poder plantear un diseño conceptual acorde al análisis realizado. Para la realización de nuestras prácticas reduciremos pues el alcance de nuestra base de datos a una porción manejable con la que podamos trabajar aún de modo realista.

3. MODELO CONCEPTUAL

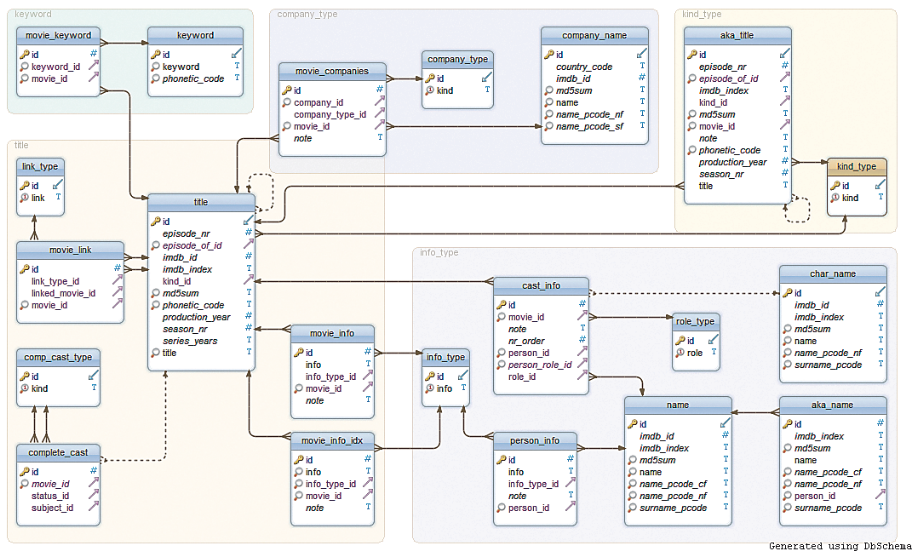

Figura 1. *Diagrama E/R para la base de datos IMDB.*

La figura 1 presenta una versión simplificada del diagrama E/R para IMDB.

4. DESCRIPCIÓN DE ENTIDADES Y ASOCIACIONES

Describimos en este apartado la colección de entidades y asociaciones que constituyen el modelo conceptual de IMDB.

Title

Almacena los títulos de todas las obras del catálogo, ya sean películas, episodios, series, programas, reportajes, etc. Además del título guarda datos que facilitan adquirir información adicional sobre la obra, tal como qué clase de obra es, su año de producción, si es un capítulo a qué serie pertenece, etc.

- *id: int(11)*. Identificador. Autoincrementado.
- *title: text*. Título de la obra (película, serie, capítulo…).
- *imdb_index: varchar(12)*. Se usa para diferenciar dos obras producidas el mismo año que comparten el mismo título. Utiliza códigos I, II, III, IV, etc.
- *kind_id: int(11)*. Código que indica el tipo de obra en cuestión. Almacena los códigos de diferentes tipos de obra, por ejemplo (7, episode) indicando que se trata de un episodio perteneciente a una serie.
- *production_year: int(11)*. Año de producción de la obra.
- *imdb_id: int(11)*. Se utiliza para acceder directamente a un registro desde la web de IMDB. Por ejemplo, si el imdb_id para un nombre es 104, entonces se puede acceder a la dirección http://www.imdb.com/name/nm0000104/ para recuperar la página del actor Antonio Banderas. Sin embargo en nuestra base de datos no se importan los valores de imdb_id, son todos nulos.
- *phonetic_code: varchar(5)*. Código SOUNDEX del título. Este código se utiliza para representar cómo suena una palabra, de modo que todas las palabras que suenan similarmente comparten el mismo código, haciendo más sencilla la búsqueda de homónimos.
- *episode_of_id: int(11)*. Se utiliza en obras que son episodios pertenecientes a una serie. Almacena el identificador de la tabla *title* para el cual la obra en cuestión es un episodio. Sólo tiene un valor no nulo cuando el valor de kind_id = 7. Para cualquier otro valor de kind_id, episode_of_id es NULO.
- *season_nr: int(11)*. Se utiliza en obras que son episodios pertenecientes a una serie. Como las series pueden tener varias temporadas, registra la temporada en cuestión. Cuando kind_id <> 7, este valor es NULL.
- *episode_nr: int(11)*. Se utiliza en obras que son episodios pertenecientes a una serie. Guarda el número del episodio correspondiente.
- *series_years:varchar(49)*. Se utiliza en obras que son o bien episodios (kind_id=7) o bien tv-series (kind_id=2). Muestra el rango de años de la serie.
- *md5sum: varchar(32)*. Hash correspondiente al título de la obra.

AKA_TITLE

Representa una asociación entre obras, para identificar las obras por los diferentes títulos que les han puesto en otros países o ubicaciones. Registra prácticamente la misma información que *Title*, añadiendo obviamente el atributo *movie_id* que le permite conectarse con la obra original en cuestión.

- *id: int(11).* Identificador. Autoincrementado.
- *movie_id: int(11).* Identificador de la obra original (película, serie, capítulo…).
- *title: text.* Título alternativo de la obra (película, serie, capítulo…).
- *imdb_index: varchar(12).* Se usa para diferenciar dos obras producidas el mismo año que comparten el mismo título. Utiliza códigos I, II, III, IV, etc.

El resto de los atributos tiene el mismo significado que para la entidad Title.

MOVIE_LINK

Tiene como propósito registrar las relaciones que existen entre dos obras. Por ejemplo la película Rocky II es continuación de Rocky, por lo que existe una entrada en esta tabla que conecta sus movie_id's e indica el significado de esa conexión.

- *id: int(11).* Identificador de la conexión. Autoincrementado.
- *movie_id: int(11).* Identificador de la obra que se conecta con otra.
- *linked_movie_id: int(11).* Identificador de la obra conectada.
- *link_type_id: int(11)* Código que indica el tipo de conexión que existe entre la obra movie_id y la obra linked_movie_id. Estos códigos se almacenan en el diccionario link_type.

LINK_TYPE

Actúa como diccionario (o lista de valores) para traducir los códigos de los diferentes posibles conexiones que existen entre dos obras. Por ejemplo, el código 1 representa que una película sigue a otra, el código 3 que es un remake, etc.

- *id: int(11).* Código del tipo de conexión.
- *link: varchar(32).* Descripción del tipo de conexión.

COMPLETE_CAST

Sobre una obra se puede disponer o no de información completa acerca de los personajes (cast) que participan en la obra y de todo el equipo (crew) que ha participado en la obra. Esta relación muestra qué clase de información existe en imbd a este respecto. Por ejemplo sobre

la película "Blade Runner" (id = 1693255) se dispone del listado completo de personajes y además verificado, pero no del listado completo, con lo que en esta relación se guarda la tupla (1693255, 1<cast>, 4<complete+verified>) pero no existe una tupla (1693255, 2<crew>, ---).

- *id: int(11).* Identificador de la información. Autoincrementado.
- *movie_id:int(11).* Identificador de la obra sobre la que se registra información.
- *subject_id: int(11).* Código que representa o bien los personajes (1-cast) o bien todo el equipo (2-crew).
- *status_id: int(11).* Código que informa sobre si lo que se tiene es un listado completo (3) o se trata de un listado completo que además ha sido verificado (4).

Comp_cast_type

Diccionario que almacena el significado de cada uno de los códigos usados en *complete_cast*.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *kind:varchar(32).* Significado del código (cast, crew, complete, complete + veryfied).

Movie_info

Almacena información de muy diversa índole sobre cada obra. Para una obra aparecen múltiples entradas, cada una mostrando una información diferente. Por ejemplo de la película "Blade Runner" se almacenan casi 800 registros en esta tabla. El asunto sobre el cual se registra información está codificado en la columna *info_type_id* y para un mismo valor de *info_type_id* y una misma obra puede haber muchas filas.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *movie_id:int(11).* Identificador de la obra sobre la que se registra información.
- *info_type_id: int(11).* Código que representa el tipo de información que se almacena en el registro.
- *info: text.* Texto con la información relativa a la obra. El valor de esta columna depende del tipo de información que se almacene en info_type_id. Así por ejemplo sobre el tipo 13 (goofs o curiosidades) se almacenan diferentes informaciones sobre una peli, tal y como errores de script, o curiosidades sobre escenas concretas.
- *note:text.* Nota adicional sobre la información que se almacena.

Movie_info_idx

Esta tabla guarda información relativa a la clasificación que los usuarios emiten sobre las obras almacenadas. La estructura de la tabla es similar a *movie_info*, pero en este caso

sólo se registra información correspondiente a los códigos 99 (votes distribution), 100 (votes), 101 (ratings), 112 (top 250 rank) y 113 (bottom 10 rank). Para las filas de código 99, la información que se muestra es una cadena de 10 caracteres representando un histograma. Cada carácter es un código que se corresponde con el porcentaje de votos que los usuarios han emitido otorgando la calificación entre 1 y 10. La posición más a la izquierda en la cadena representa el valor 1 y la posición más a la derecha el valor 10. Así por ejemplo, una cadena con valores 3 3 0 0 0 1 0 0 0 representa un histograma con el siguiente significado:

Pos	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
%	39.1	31.4	1.1	0	0.7	9.5	12.8	2.9	1.5	1.1
Cod	3	3	0	.	0	0	1	0	0	0

La tabla anterior indica que el 39,1% de los usuarios votaron a la película con valor más bajo (1) y que el 1,1% de usuarios lo votaron en la posición más alta. El código utilizado es el valor de la decena correspondiente al porcentaje de usuarios que lo votaron en esa posición.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *movie_id:int(11).* Identificador de la obra sobre la que se registra información.
- *info_type_id:int(11).* Código que representa el tipo de información que se almacena en el registro.
- *info:text.* Texto con la información relativa a la obra. El valor de esta columna depende del tipo de información que se almacene en info_type_id. Para el código 99 se almacena la cadena con el histograma. Para el código 100, el número de usuarios que votaron. Para el código 101 el *ranking* obtenido sobre 10. Para el código 112 la posición dentro del top 250. Y finalmente para el código 113 la posición dentro de las 10 peores obras.
- *note:text.* Nota adicional sobre la información que se almacena.

MOVIE_KEYWORD

Almacena las etiquetas (*keywords*) que se le han asignado a las obras. Obviamente una obra puede contener una colección amplia etiquetas. Estas etiquetas pueden utilizarse para realizar búsquedas rápidas.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *movie_id:int(11).* Identificador de la obra sobre la que se registra información.
- *keyword_id:int(11).* Código que representa la etiqueta asociada a la obra. Para decodificar la información es preciso acceder a la tabla keyword.

KEYWORD

Diccionario de etiquetas (*keywords*) utilizadas para describir las obras de imdb.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *keyword:text.* Etiqueta. Si se compone de varias palabras se utiliza el guión (-) como separador.
- *phonetic_code:varchar(5).* Código SOUNDEX de la etiqueta. Este código se utiliza para representar cómo suena una palabra, de modo que todas las palabras que suenan similarmente comparten el mismo código, haciendo más sencilla la búsqueda de homónimos.

Movie_companies

Guarda el registro de compañías que han intervenido de algún modo (distribuidoras, productoras, efectos especiales…) en la obra sobre la que se registra información. Para una misma obra y una misma compañía, pueden aparecer más de un registro. Por ejemplo, cuando se tratan de compañías distribuidoras, estas pueden intervenir en distintos años sobre distintos productos de la misma obra (diferentes ediciones de DVDs, remasterizaciones, etc.).

- *id: int(11).* Identificador de la información. Autoincrementado.
- *movie_id:int(11).* Identificador de la obra sobre la que se registra información.
- *company_id:int(11).* Código de la compañía que interviene en la obra. Para decodificarlo es preciso acceder al diccionario company_name.
- *company_type_id:int(11).* Código del tipo de compañía de que se trata (productora, distribuidora, etc.). Para decodificarlo es preciso acceder al diccionario company_type.
- *note:text.* Anotaciones sobre la forma de participar la compañía y otra información relacionada (país, año, etc.).

Company_type

Diccionario que almacena los diferentes tipos de empresas que participan en las obras almacenadas.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *kind: varchar(32).* Tipo de compañía (distribuidora, productora, etc.).

Company_name

Registra los nombres de las compañías que intervienen en las obras. Actúa como diccionario y además de sus nombres, almacena información sobre el país y códigos SOUNDEX.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *name:text.* Nombre de la compañía.
- *country_code:varchar(255).* Código textual del país al que pertenece la compañía.

- *imdb_id:int(11).* Se utiliza para acceder directamente a un registro desde la web de IMDB. En nuestro caso son todos nulos.
- *name_pcode_nf:varchar(5).* Código SOUNDEX del nombre de la compañía en formato normal.
- *name_pcode_sf:varchar(5).* Código SOUNDEX del nombre de la compañía más el código del país.
- *md5sum:varchar(32).* Hash correspondiente al nombre de la compañía.

Cast_info

Esta tabla almacena la información referente a la participación de actores, actrices, directores y demás miembros que intervienen en una obra (lo que se conoce como "*cast*").

- *id: int(11).* Identificador de la información. Autoincrementado.
- *person_id:int(11).* Identificador de la persona que participa en la obra. Para decodificarlo hay que acceder a name.
- *movie_id:int(11).* Identificador de la obra sobre la que se registra información.
- *person_role_id: int(11).* Identificador del personaje que representa la persona en la obra. Para decodificarlo es preciso acceder a char_name. Tiene valor NULL en el caso de directores, escritores, guionistas, etc.
- *note:text.* Anotaciones sobre la participación de la persona en la obra indicada.
- *nr_order:int(11).* Orden de aparición en los créditos.
- *role_id:int(11).* Código que describe el tipo de participación en una obra (1:actor, 2:actress, 3:producer, 4:writer…).

Role_type

Diccionario que almacena los diferentes tipos de participación de una persona en una obra (actores, directores, escritores…).

- *id: int(11).* Identificador de la información. Autoincrementado.
- *role:varchar(32).* Tipo de participación (actor, director, etc.).

Char_name

Registra los nombres de los personajes que intervienen en las obras. Actúa como diccionario y además de sus nombres, almacena otra información adicional.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *name:text.* Nombre del personaje.

- *imdb_index: varchar(12).* Se usa para diferenciar dos nombres de personajes iguales. Pero en este caso no aprecio su utilidad.
- *imdb_id:int(11).* Se utiliza para acceder directamente a un registro desde la web de IMDB. En nuestro caso son todos nulos.
- *name_pcode_nf:varchar(5).* Código SOUNDEX del nombre en formato normal.
- *surname_pcode:varchar(5).* Código SOUNDEX del sobrenombre.
- *md5sum:varchar(32).* Hash correspondiente al nombre del personaje.

Name

Registra los nombres de las personas que intervienen en las obras. Actúa como diccionario y además de sus nombres, almacena otra información adicional.

- *id: int(11).* Identificador de la información. Autoincrementado.
- *name:text.* Nombre de la persona. Tiene un formato "apellido, nombre".
- *imdb_index: varchar(12).* Se usa para diferenciar dos nombres de personas iguales.
- *imdb_id:int(11).* Se utiliza para acceder directamente a un registro desde la web de IMDB. En nuestro caso son todos nulos.
- *gender:varchar(1).* Género de la persona.
- *name_pcode_cf:varchar(5).* Código SOUNDEX del apellido en formato canónico.
- *name_pcode_nf:varchar(5).* Código SOUNDEX del nombre en formato normal.
- *md5sum:varchar(32).* Hash correspondiente al nombre.

Aka_name

Registra los nombres alternativos de las personas que intervienen en las obras. Son nombres por los que también son conocidos estas personas (aka = "also known as").

- *id: int(11).* Identificador de la información. Autoincrementado.
- *person_id:int(11).* Identificador del nombre de la persona que es conocida también por este nombre.
- *name:text.* Nombre alternativo de la persona. Tiene un formato "apellido, nombre".
- *imdb_index: varchar(12).* Se usa para diferenciar dos nombres de personas iguales.
- nuestro caso son todos nulos.
- *name_pcode_cf:varchar(5).* Código SOUNDEX del apellido en formato canónico.
- *name_pcode_nf:varchar(5).* Código SOUNDEX del nombre en formato normal.
- *surname_pcode:varchar(5).* Código SOUNDEX del sobrenombre.
- *md5sum:varchar(32).* Hash correspondiente al nombre.

Person_info

Registra información diversa sobre cada persona. Cada información se almacena en un registro diferente y el tipo de información a que se refiere viene codificado en una de sus columnas. Sobre un mismo actor puede haber múltiples entradas mostrando distintos tipos de información.

- *id: int(11)*. Identificador de la información. Autoincrementado.
- *person_id:int(11)*. Identificador del nombre de la persona sobre la que se guarda información.
- *info_type_id:int(11)*. Código que indica el tipo de información que se almacena. Para decodificarla hay que acceder a la tabla info_type.
- *info: text*. Información sobre la persona.
- *note:text*. Información adicional que complementa la anterior.

Info_type

Diccionario para codificar los distintos tipos de información que se registran sobre una obra o una persona.

- *id: int(11)*. Código identificador de la información. Autoincrementado.
- *info:text*. Explicación del código.

5. EJERCICIOS

Con la información recogida del análisis realizado sobre la plataforma imdb y toda la que se incluye en este documento deberás realizar los siguientes ejercicios.

5.1. Diagrama UML

Construye un diagrama UML que aporte una información completa sobre la base de datos IMDB.

5.2. Diseño lógico IMDB

Redacta un documento que recoja el diseño lógico de la base de datos imdb.

LABORATORIO 2.
MATERIALIZACIÓN DE IMDB

Objetivos. Tras cubrir esta sesión práctica, el alumno/a es capaz de utilizar el SGBD MySql y crear una base de datos a partir del modelo lógico de datos construido con anterioridad. En concreto se habrán adquirido habilidades para:

- ✓ Conectar con el servidor MySQL a través del terminal de línea de comandos.
- ✓ Hacer consultas sencillas al diccionario de datos.
- ✓ Utilizar sentencias SQL para la creación de una base de datos MySql.
- ✓ Escribir y ejecutar scripts para la ejecución batch de sentencias SQL.
- ✓ Utilizar sentencias SQL para insertar filas en tablas MySql.

1. INSTALACIÓN Y PRIMEROS PASOS CON MYSQL

Para la realización de esta práctica es preciso haber instalado y configurado correctamente el SGDB MySql. Para simplificar este proceso y homogeneizar el trabajo en el laboratorio, utilizaremos una máquina virtual "*Debian Mysql*" creada con la herramienta gratuita *VirtualBox*. Para ejecutarla, instala VirtualBox en tu equipo y arranca la máquina *Debian Mysql*. La máquina virtual tiene un usuario denominado "dabd" cuya contraseña es "**dabd_2013**", la cual es válida también para el usuario *root* de Debian y para el usuario *root* de Mysql.

La máquina virtual tiene ya preinstalado el SGDB MySql así como la herramienta web que facilita in interfaz de acceso a MySql denominada *phpmyadmin*. En las sesiones de laboratorio, así como en los documentos de uso como este, accederemos a MySql a través del terminal, pero si el alumno/a conoce con antelación la interfaz *phpmyadmin* o aprende a utilizarla, esta

herramienta puede ser útil para la realización de las prácticas de laboratorio. Para acceder a la interfaz mediante *phpmyadmin*, es preciso lanzar en el navegador y escribir en la barra de direcciones la dirección:

```
http://localhost/phpmyadmin
```

Para conectar desde terminal con el cliente MySql basta con ejecutar el comando:

```
%mysql
```

Dado que hemos adaptado el fichero de configuración /etc/mysql/my.cnf, automáticamente conectamos a través del usuario root con nuestro SGBD. Una vez accedido al cliente Mysql, aparece en el terminal el prefijo mysql>, tras el cual ya podemos comenzar a enviar órdenes y comandos a nuestro SGDB. Por ejemplo, podemos visualizar las bases de datos existentes en nuestro sistema:

```
mysql>show databases;
```

Ahora podemos escribir algunas sentencias:

```
mysql> SELECT NOW();
mysql> SELECT NOW(),USER(),VERSION();
mysql> use test;
mysql> SELECT DATABASE();
```

Sin embargo, escribir sentencias directamente desde el terminal resulta bastante incómodo por lo que nuestra forma de proceder consistirá en escribir nuestras sentencias en un archivo *.sql, utilizando para ello un editor de textos, y finalmente ejecutar la sentencia:

```
mysql> source ruta/filename.sql;
```

Si el archivo filename.sql lo almacenamos en el mismo directorio desde donde llamamos a mysql, entonces el fichero lo encontrará sin necesidad de escribir la ruta.

2. CREACIÓN DE LA BASE DE DATOS MYIMDB

Ahora que sabemos conectar con el servidor de base de datos, podemos ya comenzar a utilizar las sentencias del DDL para crear nuestra base de datos y comenzar a construir las tablas que hemos identificado en el modelo lógico construido para IMDB.

Antes de avanzar con el desarrollo de esta práctica es importante resaltar que existe un completo y preciso manual de MySql al cual se puede acceder desde la dirección www.mysql.com. La pestaña *Documentation* de esta página da acceso a los manuales para las distintas versiones de Mysql. Cualquier duda sobre la sintaxis de un comando concreto puede despejarse accediendo a este manual.

La sintaxis necesaria para crear una base de datos es

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
```

De modo que para crear nuestra base de datos ejecutamos:

```
mysql> CREATE DATABASE myimdb;
mysql> USE myimdb;
```

A partir de este punto podemos ya comenzar a crear nuestras tablas. A modo de ilustración vamos a crear la tabla *title*. Escribe en tu editor de textos la sentencia:

```
CREATE TABLE title (
    id INT(11) NOT NULL AUTO_INCREMENT,
    title TEXT NOT NULL,
    imbd_idex VARCHAR(12),
    kind_id INT(11) NOT NULL,
    production_year INT(11),
    imdb INT (11),
    phonetic_code VARCHAR(5),
    episode_of_id INT(11),
    season_nr   INT(11),
    episode_nr INT(11),
    series_years VARCHAR(49),
    md5sum VARCHAR(32),
    PRIMARY KEY (id)
    FOREING KEY (episode_of_id) REFERENCES title(id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREING KEY (kind_id) REFERENCES kind_type(id)
        ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=MyISAM;
```

Guarda el archivo con nombre *create_imdb.sql* y posteriormente ejecuta la sentencia

```
mysql>source ruta/create_imdb.sql;
```

Cuando se ejecuta el archivo, MySql nos reporta un error debido a la inexistencia de las tablas que se referencian en las cláusulas FOREING KEY. Para evitar este error podemos proceder de dos formas diferentes. La primera consiste en crear la tabla sin utilizar las cláusulas de clave externa hasta haber completado la colección de tablas de la base de datos. Posteriormente podremos modificar la definición de las tablas que contengan claves externas añadiendo sus declaraciones. Así por ejemplo, podríamos crear la tabla *title* sin las cláusulas FOREIGN KEY y posteriormente añadir:

```
ALTER TABLE title ADD
    FOREIGN KEY (episode_of_id) REFERENCES title(id)
        ON DELETE CASCADE ON UPDATE CASCADE,
```

Otra vía quizás más sencilla consiste en desactivar el chequeo de claves externas antes de ejecutar la sentencia y posteriormente activarlo de nuevo. Para ello debemos incluir en la cabecera del nuestro archivo la siguiente sentencia:

```
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
```

Y posteriormente reactivar el chequeo añadiendo como última sentencia:

```
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
```

Esta sentencia crea la tabla *title* con los atributos indicados en el esquema. La sintaxis para definir columnas es:

```
nombre_col tipo [NOT NULL | NULL] [DEFAULT valor_por_defecto]
[AUTO_INCREMENT] [[PRIMARY] KEY] [COMMENT 'string']
[FK_definición]
```

Veamos algunos aspectos interesantes que aparecen en esta simple definición de tabla.

Valores nulos

Cuando definimos una columna tras indicar su nombre debemos declarar el tipo de dato que se va a usar en dicha columna. Más adelante hablaremos sobre los tipos de datos permitidos en MySQL. Además, podemos indicarle si permitimos valores nulos o no sobre ella. Notemos que un valor NULL significa ausencia de datos, que no es lo mismo que tener una cadena vacía. Consideremos por ejemplo las siguientes sentencias:

```
mysql> INSERT INTO title (episode_nr) VALUES (NULL);
mysql> INSERT INTO title (episode_nr) VALUES ('');
```

Ambas sentencias insertan un valor en la columna *episode_nr*, la primera inserta un valor NULL y la segunda una cadena de caracteres vacía. El significado de la primera sentencia puede interpretarse como "se desconoce el número de episodio de la obra", la segunda se interpretaría como "se sabe que la obra en cuestión no es un episodio, por tanto no tiene número". Una cita que viene a colación con el uso de los valores nulos pertenece al secretario de defensa de los Estados Unidos en 2002 Donald Rumsfeld y la utiliza Stefan Faroult en su excelente libro "The Art of SQL":

> *Como sabemos hay conocimientos conocidos. Hay cosas que nosotros sabemos que las conocemos. También sabemos que hay desconocimientos conocidos. Es decir sabemos que hay algunas cosas que desconocemos. Pero hay también desconocimientos desconocidos, los únicos que nosotros no sabemos que desconocemos.*

El uso de valores NULL debería reservarse para, en palabras de Rumsfeld "desconocimientos conocidos", atributos que sabemos que existen y tienen algún valor que no conocemos en un momento dado. Finalmente, como bien indica Faroult, los nulos pueden ser peligrosos para tu lógica; si debes usarlos, asegúrate que entiendes las consecuencias de hacerlo en tu situación particular. A lo largo de presente curso, volveremos a insistir sobre el uso de valores nulos.

Valores por defecto

La cláusula DEFAULT nos permite definir opcionalmente un valor por defecto para la columna. El valor por defecto se asignará de forma automática a una columna cuando no se especifique un valor determinado al añadir filas a la tabla. El valor por defecto debe ser una constante, de modo que no se permiten funciones o expresiones, tales como NOW() o CURRENT_DATE. La única excepción a esta regla es que una columna de tipo TIMESTAMP puede asignársele como valor por defecto CURRENT_TIMESTAMP. Si una columna puede tener un valor nulo, y no se especifica un valor por defecto, se usará NULL como valor por defecto. En caso contrario, MySQL utiliza el valor por defecto implícito para cada tipo de columna.

Claves primarias

Como vemos, la columna *id* se declara como clave primaria de la tabla, para ello utilizamos la cláusula PRIMARY KEY. Sólo puede existir una clave primaria en cada tabla, y la columna sobre la que se define una clave primaria no puede tener valores *NULL*. Una PRIMARY KEY declara en realidad un índice único sobre la tabla en cuestión.

COLUMNAS AUTOINCREMENTADAS

MySQL permite que una columna declarada de algún tipo entero, tome un valor único asignado automáticamente por el sistema. Si al insertar una fila se omite el valor de la columna autoincrementada o si se inserta un valor nulo para esa columna, su valor se calcula automáticamente, tomando el valor más alto de esa columna y sumándole una unidad. Esto permite crear, de una forma sencilla, una columna con un valor único para cada fila de la tabla.

Generalmente, estas columnas se usan como claves primarias 'artificiales'. **MySQL** está optimizado para usar valores enteros como claves primarias, de modo que la combinación de clave primaria entera y autoincrementada es ideal para usarla como clave primaria artificial. En nuestra base de datos *imdb*, muchas tablas incluyen una clave primaria cuyo valor es asignado automáticamente mediante el uso de AUTO_INCREMENT. Ello nos permite no preocuparnos por la unicidad de clave. Una recomendación importante es que el uso de AUTO_INCREMENT para valores de clave primaria se debería considerar cuando no se precisa controlar el modo en que se le asignan valores a la columna, basta con que dichos valores sean únicos.

TIPOS DE DATOS BÁSICOS: ENTEROS Y CARACTERES

MySQL provee un muestrario rico de tipos de datos para las columnas. En una primera clasificación podríamos distinguir tipos numéricos, cadenas de caracteres, de fecha y tiempo, tipos enumerados y de conjunto y tipos contenedores para documentos grandes (imágenes, textos, etc.). Para no abordarlos todos desde un principio, iremos describiendo a lo largo del curso las características propias de los tipos particulares a medida que vayan surgiendo las necesidades. Dentro de los tipos numéricos el más utilizado es el tipo entero. La siguiente tabla muestra los tipos enteros que soporta MySQL y sus principales características.

Tipo		*Bytes*	*Valor mínimo*	*Valor máximo*
TINYIT	SIGNED	1	–128	127
	UNSIGNED		0	255
SMALLINT	SIGNED	2	–32768	32767
	UNSIGNED		0	65535
MEDIUMINT	SIGNED	3	–8388608	8388607
	UNSIGNED		0	16777215
INT	SIGNED	4	–2147483648	2147483647
	UNSIGNED		0	4294967295
BIGINT	SIGNED	8	–9223372036854775808	9223372036854775807
	UNSIGNED		0	18446744073709551615

Para visualizar más cómodamente los valores de un tipo entero podemos restringir el ancho de visualización de dicho valor escribiéndolo entre paréntesis, por ejemplo SMALLINT(3). En este caso el valor se visualiza con tres dígitos siempre y cuando su valor precise 3 o menos dígitos, pero la anchura de visualización no restringe el rango de valores de la columna en sí. En el caso anterior un valor de 4537 en el campo de tipo SMALLINT(3) se visualiza con los 4 dígitos requeridos.

Además de los anteriores existe un tipo entero especial BIT(M), donde el máximo valor de M es 64, cuyo propósito es facilitar el almacenamiento en binario. Para especificar valores de bit se usa una notación b'valor', como por ejemplo b'101'.

Por su parte los tipos cadena de caracteres básicos son CHAR Y VARCHAR. Difieren en la forma en que se almacenan y recuperan así como en su longitud máxima. Siempre se declaran con la longitud máxima permitida de caracteres. Por ejemplo CHAR(20) es un tipo que puede contener hasta 20 caracteres. El tipo CHAR es de longitud fija y ocupa el número de caracteres declarado en la sentencia CREATE TABLE. La longitud puede ser cualquier valor entre 0 y 255. Cuando se almacenan, se rellenan con espacios a la derecha hasta la longitud indicada. Cuando se recuperan esos espacios se ignoran. El tipo VARCHAR sirve para almacenar cadenas de longitud variable y su longitud puede tomar un valor entre 0 to 65535. Estas columnas se almacenan con uno o dos bytes de prefijo indicando su longitud actual.

La última parte de la sintaxis para definir columnas se refiere a la declaración de claves externas (*Foreign Key*). Para especificar una restricción referencial en una relación se utiliza la siguiente sintaxis:

```
[CONSTRAINT [symbol]]
FOREIGN KEY [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
reference_option: CASCADE | SET NULL | NO ACTION
```

Sin embargo esta sintaxis sólo tiene efecto cuando el motor de base de datos que se utiliza para almacenar la tabla es *Innodb*. En general el motor de almacenamiento por defecto es *MyISAM*, que no provee control para restricciones referenciales, por lo que se ignora la sentencia. Para utilizar el motor *Innodb* hay que indicarlo expresamente en la sentencia CREATE TABLE.

Imdb no es una base de datos típicamente transaccional, puesto que la principal operación que se realiza es la búsqueda de información, es decir operaciones de lectura. Tal circunstancia la hace una candidata perfecta a almacenarla en el motor MyIsam, el cual aporta mucha mayor rapidez para las lecturas, aunque por contra no da soporte a aspectos tales como las restricciones de integridad. Sea como sea, el esquema lógico de la base de datos *imdb* debe incluir las restricciones de clave externa.

3. EJERCICIOS

3.1. Inserción de datos

Una vez creada la tabla *title* en la base de datos *myimdb*, utiliza la sentencia INSERT INTO para realizar la inserción de las siguientes filas:

- Película: "The Matrix", año de producción: 1999, phonetic_code: M363.
- Serie: "Breaking Bad", año de producción: 2008, phonetic_code B6252, series_years: "2008-????".
- Capítulo de la serie "Breaking Bad", título: "Down", año de producción: 2009, phonetic_code: D5, season_nr: 2, episode_nr: 4.

3.2. Modificación/actualización de datos

Modifica el código fonético de la película "The Matrix" por B362.

3.3. Carga masiva de datos

El archivo *person_data.txt* contiene 100 líneas correspondientes a registros que debemos almacenar en la tabla *person_info*. Crea la tabla *person_info*. Analiza el archivo *person_data.txt* y utiliza la sentencia LOAD DATA INFILE para cargar los registros desde el archivo *person_data.txt* en la tabla *person_info*.

LABORATORIO 3.
CONSULTAS SOBRE IMDB

Objetivos. Tras cubrir esta sesión práctica, el alumno/a es capaz de utilizar la potencia del lenguaje SQL proporcionado por MySql para realizar las operaciones requeridas sobre la base de datos IMDB. La presente sesión reafirma las habilidades adquiridas en la asignatura Bases de Datos en cuanto a utilización del DML de MySql para resolver problemas específicos en un contexto realista. En concreto, se adquirirán habilidades para:

- ✓ Utilizar operadores y funciones en las consultas.
- ✓ Expresar consultas de agrupación y funciones agregadas.
- ✓ Expresar mediante subconsultas.
- ✓ Especificar consultas complejas mediante JOIN.

1. INTRODUCCIÓN

Para la realización de esta práctica trabajaremos con la maquina virtual "*Debian DABD imdb_no_index*". Esta máquina incorpora la base de datos IMDB siguiendo el esquema lógico obtenido en prácticas anteriores. Como en la sesión anterior, la máquina virtual tiene un usuario denominado "dabd" cuya contraseña es "**dabd_2013**", la cual es válida también para el usuario *root* de Debian y para el usuario *root* de Mysql.

2. CONSULTANDO EL DICCIONARIO

Antes de realizar consultas de usuario sobre IMDB, vamos a examinar alguna información relevante que se almacena en el diccionario de datos de MySQL.

La sentencia

```
mysql>show databases;
```

presenta las bases de datos que almacena nuestro servidor. Entre otras encontrarás *information_schema*. Se trata de la BD donde MySQL almacena el diccionario. Esta base de datos contiene una serie de tablas que nos aportan información de interés sobre las diferentes bases de datos del servidor. Veamos la información que sobre cada tabla almacena nuestro diccionario:

```
mysql>desc information_schema.tables;
```

Visualiza para cada tabla de imdb la cantidad de filas que contiene:

```
SELECT table_name, table_rows
FROM information_schema.tables
WHERE table_schema = 'imdb'
ORDER BY table_rows DESC;
```

EJERCICIO 1

Presenta una tabla como la que se muestra en la siguiente figura.

```
+----------------+-----------+----------+
| table_name     | table_rows | Size     |
+----------------+-----------+----------+
| cast_info      |  35645119 | 1.14 Gb  |
| movie_info     |  14627341 | 901.45 Mb |
| person_info    |   2938437 | 369.94 Mb |
| name           |   4109319 | 304.89 Mb |
| char_name      |   3109986 | 211.75 Mb |
| title          |   2477301 | 196.22 Mb |
| movie_companies |   2550045 | 88.14 Mb  |
| aka_name       |    891626 | 68.07 Mb  |
| movie_keyword  |   4467778 | 55.39 Mb  |
| aka_title      |    358335 | 36.80 Mb  |
| movie_info_idx |   1357151 | 34.05 Mb  |
| movie_link     |   1578016 | 25.58 Mb  |
| company_name   |    232287 | 17.64 Mb  |
| keyword        |    133055 | 4.26 Mb   |
| complete_cast  |    135086 | 2.19 Mb   |
| info_type      |       113 | 2.68 Kb   |
| link_type      |        18 | 392 bytes |
| role_type      |        12 | 268 bytes |
| kind_type      |         7 | 144 bytes |
| company_type   |         4 | 124 bytes |
| comp_cast_type |         4 | 88 bytes  |
+----------------+-----------+----------+
21 rows in set (0.00 sec)
```

Para obtener esta tabla debemos capturar la información que nos aporta la columna *data_length* de la tabla *information_schema.tables*. Esta columna almacena el tamaño en bytes de cada tabla, pero en nuestro caso deseamos mostrar esa información procesada mediante el uso de algunas funciones de MySql.

En primer lugar junto con el dato referente al tamaño presentamos una cadena que nos muestra si es en bytes, kilobytes, etc. Para ello usamos la función CONCAT(), la cual concatena cadenas de caracteres o valores uno tras otro, por ejemplo:

```
SELECT table_name, CONCAT(data_length,' Bytes') AS 'size'
FROM information_schema.tables
WHERE table_schema = 'imdb'
AND table_name = 'comp_cast_type';
```

Además necesitamos convertir bytes a Kb, Mb o Gb en función del valor de *data_length*. Para hacer esto podemos dividir el valor de *data_length* por 1024^3 y preguntamos si el resultado es mayor que 1, si es así, colocamos el tamaño en Gb, de lo contrario dividimos por 1024^2 y si es mayor que 1, presentamos el resultado en Mb, etc. Para ello podemos usar la función IF(exp,true,false), por ejemplo:

```
SELECT table_name, IF(data_length/(1024*1024*1024) > 1, CONCAT(-
data_length/(1024*1024*1024),' Gb'),'otro') AS 'size'
FROM information_schema.tables
WHERE table_schema = 'imdb'
AND table_name = 'cast_info';
```

Alternativamente se puede usar la cláusula CASE, por ejemplo:

```
SELECT table_name, CASE
    WHEN data_length/(1024*1024*1024) > 1 THEN CONCAT(data_
length/(1024*1024*1024),' Gb')
    ELSE 'otro'
    END AS 'size'
FROM information_schema.tables
WHERE table_schema = 'imdb'
AND table_name = 'cast_info';
```

Sin embargo, como vemos en la figura correspondiente al ejercicio 1, el tamaño se presenta con un valor seguido por 2 decimales. Para ello se utiliza la función FORMAT(valor, dec), tal que así:

```
SELECT table_name, CASE
    WHEN data_length/(1024*1024*1024) > 1 THEN CONCAT(FORMAT(da-
ta_length/(1024*1024*1024),2),' Gb')
    ELSE 'otro'
    END AS 'size'
FROM information_schema.tables
WHERE table_schema = 'imdb'
AND table_name = 'cast_info';
```

3. CONSULTAS DE AGRUPACIÓN Y FUNCIONES AGREGADAS

A través de la cláusula GROUP BY podemos obtener valores agregados a partir de un número determinado de filas de una tabla. En este apartado vamos a trabajar con consultas que obtienen valores agregados.

Enfocamos en primer lugar nuestra necesidad de información en el diccionario de datos, por ello vamos a visualizar, para cada base de datos existente en nuestro servidor MySql, el número de tablas fueron creadas en abril de 2013. Para responder esta consulta nos centramos en la tabla *tables* de *information_schema*. Esta tabla incluye una columna *create_time* que almacena la fecha de creación de cada una de las tablas. Por otra parte la columna *table_schema* almacena el nombre de la base de datos a que pertenece la tabla. Así pues debemos realizar una agrupación por *table_schema*, pero primero restringiremos las filas a aquellas tablas que fueron creadas:

```
SELECT table_schema, count(*) AS 'tablas en abril 2013'
FROM information_schema.tables
WHERE YEAR(create_time) = 2013 AND MONTH(create_time)=4
GROUP BY table_schema;
```

```
+--------------+----------------------+
| table_schema | tablas en abril 2013 |
+--------------+----------------------+
| imdb         |                   21 |
| mysql        |                   12 |
| phpmyadmin   |                    9 |
+--------------+----------------------+
3 rows in set (0.00 sec)
```

EJERCICIO 2

Muestra tamaño medio de tabla, número de tablas y cantidad total de tuplas de las bases de datos salvo la del propio diccionario. Obtén una salida como ésta:

```
+--------------+----------------+-----------+------------+
| table_schema | table_size_avg | table_num | total_rows |
+--------------+----------------+-----------+------------+
| imdb         | 166 Mb         |        21 | 74,611,040 |
| mysql        | 25 Kb          |        23 | 2,022      |
| phpmyadmin   | 14 bytes       |         9 | 1          |
| prueba       | 16 Kb          |         3 | 7          |
+--------------+----------------+-----------+------------+
4 rows in set (0.00 sec)
```

□

EJERCICIO 3

Lista el número de series españolas producidas ente los años 2000 y 2005 con indicación del año y número de series de ese año.

```
+-----------------+----------+
| production_year | COUNT(*) |
+-----------------+----------+
|            2000 |      130 |
|            2001 |      101 |
|            2002 |      101 |
|            2003 |       97 |
|            2004 |      117 |
|            2005 |      153 |
+-----------------+----------+
6 rows in set (4.72 sec)
```

□

4. USO DE SUBCONSULTAS

En ocasiones para responder a una consulta es necesario utilizar datos devueltos por otra consulta. Este problema se resuelve utilizando subconsultas, es decir una consulta anidada dentro de otra. La forma más habitual de una subconsulta es incluirla como parte de la cláusula WHERE dentro de la consulta principal, aunque también es posible anidar una subconsulta dentro de la cláusula SELECT.

En su forma más sencilla tenemos las subconsultas escalares. Son subconsultas que retornan un único valor (escalar) y pueden utilizarse en cualquier posición en que un valor de columna o

literal sea legal. Supongamos que queremos conocer títulos de películas españolas producidas en 2012 cuya valoración (*ranking*) está por encima de la valoración media de las películas españolas producidas en 2012. Una forma sencilla de responder esta consulta sería obtener primero la duración media de las películas españolas producidas 2012. Teniendo en cuenta que la información de *ranking* se encuentra en la tabla *movie_info_idx* (código 101) y que la información sobre el país de la película se encuentra en *movie_info* (código 8), nos vemos obligados a realizar joins con las tablas *title*, *movie_info_idx* y *movie_info*. Además la información de *ranking* se almacena en el campo *info* de la tabla *movie_info_idx*, el cual es una cadena de caracteres, por lo que debemos convertirla a decimal usando la función CAST(). Así pues construimos una consulta que obtenga la valoración media de las películas españolas de 2012:

```
SELECT avg(CAST(mix.info as DECIMAL))
FROM movie_info_idx mix JOIN title t ON (t.id = mix.movie_id)
           JOIN movie_info mi ON (mi.movie_id = t.id)
WHERE kind_id = 1 AND production_year = 2012
AND mix.info_type_id = 101 AND mi.info_type_id = 8
AND mi.info = 'Spain';
```

Ahora esta consulta puede ser usada como subconsulta para obtener las películas españolas de 2012 con *ranking* superior a este valor:

```
SELECT title
FROM title t JOIN movie_info_idx mix ON (t.id = mix.movie_id)
        JOIN movie_info mi ON (mi.movie_id = t.id)
WHERE kind_id = 1 AND production_year = 2012
AND mix.info_type_id = 101 AND mi.info_type_id = 8
AND mi.info = 'Spain'
AND CAST(mix.info as DECIMAL) >
(SELECT avg(CAST(mix.info as DECIMAL))
FROM movie_info_idx mix JOIN title t ON (t.id = mix.movie_id)
           JOIN movie_info mi ON (mi.movie_id = t.id)
WHERE kind_id = 1 AND production_year = 2012
AND mix.info_type_id = 101 AND mi.info_type_id = 8
AND mi.info = 'Spain');
```

```
| Venuto al mondo                               |
| Vilamor                                       |
| WWW.                                          |
| Wilaya                                        |
| Wrath of the Titans                           |
| Zeinek Gehiago Iraun                          |
| Zuloak                                        |
| ¡Atraco!                                      |
| Ánima Buenos Aires                            |
+-----------------------------------------------+
115 rows in set (6.12 sec)
```

EJERCICIO 4

Obtén el género y número de películas españolas producidas en 2012 para el género que mayor número de películas produjo en 2012 en todo el mundo. Para obtener este resultado, construye primero una subconsulta que muestre el género que mayor número de películas produjo en 2012 en todo el mundo y después usa este resultado como escalar para compararlo en la consulta principal.

```
+-------+--------+
| info  | cuantas |
+-------+--------+
| Short |     750 |
+-------+--------+
1 row in set (9.22 sec)
```

□

Además de las subconsultas escalares, se pueden expresar subconsultas que retornan una colección de valores. En este caso los operadores ANY, IN y ALL son de utilidad en la cláusula WHERE. Por ejemplo, para los títulos de las 10 mejores películas en el top 250, podríamos generar una subconsulta que obtuviera los identificadores de las 10 mejores películas del top 250:

```
SELECT movie_id
FROM movie_info_idx
WHERE info_type_id = 112
AND info <= 10;
```

Y utilizar el resultado en la consulta principal:

```
SELECT title
FROM title
WHERE id IN (
    SELECT movie_id
    FROM movie_info_idx
    WHERE info_type_id = 112
    AND info <= 10);
```

Sin embargo esta consulta tarda muchísimo en ejecutarse y resulta mucho más eficiente construirla con un simple join:

```
SELECT title
FROM title t JOIN movie_info_idx mix ON (t.id = mix.movie_id)
WHERE info_type_id = 112
AND info <= 10;
```

En general conviene advertir que las consultas construidas mediante subconsultas, aunque a primera vista pueden resultar más legibles para el usuario tienen la desventaja de que son difícilmente optimizadas por el procesador de consultas, por lo que en general, resulta más eficiente construir consultas mediante joins que a través de subconsultas (siempre que ello sea posible).

5. CONSULTAS MEDIANTE JOIN

Recordemos que existen básicamente cuatro tipos de join. El más simple de todos es el CROSS JOIN que lleva a cabo el producto cartesiano de dos tablas, sin establecer ninguna condición sobre los atributos de las relaciones participantes. Se trata de una operación infrecuente y desde luego no existen ejemplos significativos de consultas en IMDB que precisen un CROSS JOIN. Por su parte, la operación JOIN más frecuente en los sistemas de bases de datos es el INNER JOIN, utilizado mayormente para mostrar información de tablas conectadas mediante claves externas en asociaciones uno a muchos y muchos a muchos.

La sintaxis para expresar un INNER JOIN es:

```
SELECT column_list
FROM table1 [INNER] JOIN table2 [join_condition]
<join_condition>: ON exp_conditional | USING (column_list)
```

Como puede verse la palabra INNER es opcional y existen dos posibles formas de expresar la condición del join, bien utilizando una cláusula ON o bien utilizando la fórmula USING. En la sintaxis anterior *table1* o *table2* pueden a su vez estar expresadas mediante la expresión JOIN, permitiendo así realizar el join de varias tablas.

Por ejemplo, mediante INNER JOIN podemos expresar una consulta que muestre los nombres de actores/actrices que participaron en la película “Blade Runner”, cuyo identificador es 1693255:

```
SET @blade_runner_id = 1693255;
SELECT name
FROM name n JOIN cast_info ci ON (n.id = ci.person_id)
WHERE ci.role_id IN (1,2)
AND ci.movie_id = @blade_runner_id;
```

```
| DeMirjian, Carolyn  |
| Hannah, Daryl       |
| Heising, Dawna Lee  |
| Hesky, Sharon       |
| Hine, Kelly         |
| Hiroshige, Kimiko   |
| Mascari, Rose       |
| Rhee, Alexis        |
| Young, Sean         |
+--------------------+
31 rows in set (10.74 sec)
```

La cláusula USING es cómoda cuando se trata de un join natural y el atributo de join de ambas tablas se denomina de la misma forma, sin embargo cuando esto no es así, la única posibilidad es utilizar la cláusula ON para expresar la condición del join. Por ejemplo, para seleccionar los géneros de las 5 películas en lo alto del Top 250, podríamos escribir:

```
SELECT DISTINCT mi.info
FROM movie_info_idx mix JOIN movie_info mi USING (movie_id)
WHERE mix.info_type_id =112 AND mix.info < 6
AND mi.info_type_id =3;
```

Para ilustrar el join entre más de dos tablas, supongamos que deseamos listar título y valoración de películas españolas de 2011 valoradas con una puntuación (*rating*) mayor que 8:

```
SELECT title, mix.info
FROM title t JOIN movie_info_idx mix ON (t.id = mix.movie_id)
    JOIN movie_info mi ON (mi.movie_id = t.id)
WHERE kind_id = 1 AND production_year = 2011
AND mix.info_type_id = 101 AND mi.info_type_id = 8
AND mi.info = 'Spain' AND mix.info > 8
```

Como hemos visto el INNER JOIN identifica filas coincidentes en las tablas que conecta. A veces, con objeto de mejorar la salida del JOIN, además de mostrar información sobre las filas coincidentes sobre una o más columnas deseamos también mostrar información sobre las filas de una tabla que no coinciden con las de la otra tabla. La operación OUTER JOIN además de presentar las filas coincidentes retiene también filas que no satisfacen la condición de join.

Esta operación tiene dos perspectivas, una a la izquierda y otra hacia la derecha. La primera de ellas presenta los datos de **todas las filas de la tabla de la izquierda** (la primera referenciada en el OUTER JOIN) y para aquellas filas que no satisfacen la condición de join, se muestran nulos los datos correspondientes a la tabla de la derecha. Este OUTER JOIN se expresa como LEFT [OUTER] JOIN. La perspectiva derecha, expresada como RIGHT [OUTER]

JOIN, representa la operación simétrica, es decir muestra todas las filas de la tabla de la derecha (la segunda referenciada en el OUTER JOIN) mostrando nulos en los atributos correspondientes a la tabla de la izquierda.

Para ilustrar el OUTER JOIN, supongamos que deseamos obtener un listado de las 5 mejores películas del Top 250 y para aquellas que tengan un título alternativo en español, indicar su título alternativo. La primera intuición nos lleva a escribir la consulta:

```
SELECT t.title, aka.title, aka.note
FROM title t JOIN movie_info_idx mix ON (t.id = mix.movie_id)
    LEFT JOIN aka_title aka ON (t.id=aka.movie_id)
WHERE info_type_id = 112
AND info <= 5 AND aka.note LIKE '(Spain%';
```

Sin embargo, esta consulta no aporta el resultado deseado, ya que en la cláusula WHERE estamos descartando aquellas filas cuyo título alternativo no está almacenado en español (*LIKE '(Spain%'*). Se hace pues necesario eliminar dicha restricción de la cláusula WHERE y restringirlo antes de hacer el LEFT JOIN. Para ello la solución que tenemos es utilizar una subconsulta en la cláusula FROM, de este modo:

```
SELECT t.title, aka.title, aka.note
FROM title t JOIN movie_info_idx mix ON (t.id = mix.movie_id)
    LEFT JOIN (SELECT movie_id,title,note
            FROM aka_title
            WHERE note LIKE '(Spain%') AS aka
    ON (t.id=aka.movie_id)
WHERE info_type_id = 112
AND info <= 5;
```

La cual ya nos aporta los resultados esperados:

```
+--------------------------------+---------------------------+-----------------------+
| title                          | title                     | note                  |
+--------------------------------+---------------------------+-----------------------+
| The Shawshank Redemption       | NULL                      | NULL                  |
| The Godfather                  | NULL                      | NULL                  |
| The Godfather: Part II         | NULL                      | NULL                  |
| Pulp Fiction                   | NULL                      | NULL                  |
| Il buono, il brutto, il cattivo. | El bo, el lleig i el dolent | (Spain: Catalan title) |
| Il buono, il brutto, il cattivo. | El bueno, el feo y el malo | (Spain)              |
+--------------------------------+---------------------------+-----------------------+
6 rows in set (0.44 sec)
```

6. EJERCICIOS FINALES. CONSULTAS SOBRE IMDB

Responde a las siguientes consultas.

1. ¿Cuantas películas españolas incluye la base de datos IMDB?

```
+----------+
| COUNT(*) |
+----------+
|    19451 |
+----------+
1 row in set (7.64 sec)
```

2. Para cada año, ¿cual es el número total de votos (código 100) a películas de ese año?

```
|              2004 | 14,698,404         |
|              2005 | 13,840,684         |
|              2006 | 15,549,551         |
|              2007 | 16,483,941         |
|              2008 | 16,749,304         |
|              2009 | 15,250,934         |
|              2010 | 14,851,752         |
|              2011 | 15,607,381         |
|              2012 | 11,293,534         |
|              2013 | 463,170            |
|              2014 | 24                 |
+-----------------+--------------------+
129 rows in set (2.87 sec)
```

3. Títulos de películas que han protagonizado juntos John Travolta (id=1550088) y Uma Thurman (id = 2563779).

```
+---------+----------------------------------------------------+
| id      | title                                              |
+---------+----------------------------------------------------+
| 2175869 | Pulp Fiction                                       |
| 2463497 | You're Still Not Fooling Anybody                   |
| 1676469 | Be Cool                                            |
| 2072242 | Midnight Movies: From the Margin to the Mainstream |
| 1698357 | Boffo! Tinseltown's Bombs and Blockbusters         |
+---------+----------------------------------------------------+
5 rows in set (21.56 sec)
```

4. Lista nombres y año de nacimiento (código 21) de todos los actores de "Pulp Fiction" (id= 2175869). Obtén el listado en orden creciente de año de nacimiento. (Sin indexar convenientemente esta consulta resulta altamente ineficiente).

```
| Plummer, Amanda         | 23 March 1957     |
| Plummer, Amanda         | 23 March 1957     |
| Rhames, Ving            | 12 May 1959       |
| Roth, Tim               | 14 May 1961       |
| Roth, Tim               | 14 May 1961       |
| Sekula, Andrzej         | 19 December 1954  |
| Sitka, Emil             | 22 December 1914  |
| Steers, Burr            | 8 October 1965    |
| Stoltz, Eric            | 30 September 1961 |
| Sweeney, Julia          | 10 October 1959   |
| Tarantino, Quentin      | 27 March 1963     |
| Tarantino, Quentin      | 27 March 1963     |
| Tarantino, Quentin      | 27 March 1963     |
| Tarantino, Quentin      | 27 March 1963     |
| Thurman, Uma            | 29 April 1970     |
| Travolta, John          | 18 February 1954  |
| Walken, Christopher     | 31 March 1943     |
| Weinstein, Bob          | 1954              |
| Weinstein, Harvey       | 19 March 1952     |
| Whaley, Frank           | 20 July 1963      |
| Whitaker, Duane         | 23 June 1959      |
| Willis, Bruce           | 19 March 1955     |
+-------------------------+-------------------+
54 rows in set (1 hour 13 min 6.93 sec)
```

5. Títulos y años de todas las películas que ha protagonizado John Travolta (id=1550088) desde 1980.

```
| Religulous                                          |            2008 |
| Old Dogs                                            |            2009 |
| The Taking of Pelham 1 2 3                          |            2009 |
| From Paris with Love                                |            2010 |
| Paul Williams Still Alive                           |            2011 |
| Unite for Japan                                     |            2011 |
| Casting By                                          |            2012 |
| Savages                                             |            2012 |
| Killing Season                                      |            2013 |
| Gotti: In the Shadow of My Father                   |            2014 |
| The Killer                                          |            2014 |
+-----------------------------------------------------+-----------------+
68 rows in set (10.58 sec)
```

6. Títulos de las 10 películas mejor valoradas (Cod 101) en 1992 y valoración de las mismas.

```
+---------------------------+------+
| title                     | info |
+---------------------------+------+
| Allein mit Dir            | 9.8  |
| Cîntarea cîntarilor       | 9.7  |
| Remember to Breathe       | 9.6  |
| CM101MMXI Fundamentals    | 9.5  |
| Girls Loving Girls        | 9.4  |
| Red Lodge                 | 9.4  |
| Tupac Shakur: Thug Angel 2 | 9.4  |
| The Shawshank Redemption  | 9.3  |
| Bingo                     | 9.0  |
| Pulp Fiction              | 9.0  |
+---------------------------+------+
10 rows in set (0.00 sec)
```

7. En 1994, ¿cual fue la valoración (cod 101) media de los usuarios de una película para cada idioma (cod 4) diferente? Lista el resultado en orden descendente del valor de valoración.

```
| Danish               | 5.17                    |
| Filipino             | 5.03                    |
| Hindi                | 5.03                    |
| Tagalog              | 4.96                    |
| Malay                | 4.50                    |
| Oriya                | 4.40                    |
| Croatian             | 4.30                    |
| Inuktitut            | 4.00                    |
| Catalan              | 3.70                    |
| Esperanto            | 3.50                    |
| Armenian             | 2.90                    |
+----------------------+-------------------------+
93 rows in set (3.39 sec)
```

8. ¿Qué actores/actrices de la película “Pulp Fiction” (id= 2175869) nunca protagonizaron conjuntamente ninguna otra película con otro actor/actriz de “Pulp Fiction”? (Extremadamente lenta sin indexar).

9. ¿Qué película protagonizada por John Travolta fue la mejor valorada (cod 101) por los usuarios? Muestra también la valoración.

```
+-------------+------+
| title       | info |
+-------------+------+
| Pulp Fiction | 9.0  |
+-------------+------+
1 row in set (13.30 sec)
```

10. Lista nombre de actores, fecha de nacimiento y nombre del personaje que interpretan en la película “Pulp Fiction” (si el personaje que interpretan no está almacenado, muestra un valor NULL). (Muy lenta sin indexar)

```
| 1562476 | Turner, Rich         | Sportscaster #2                    | NULL             |
| 2586599 | Valentino, Venessia  | Pedestrian                         | NULL             |
| 2586599 | Valentino, Venessia  | Bonnie Dimmick                     | NULL             |
| 1747168 | Arquette, Alexis     | Man #4                             | 28 July 1969     |
| 2141618 | Kaye, Linda          | Shot Woman                         | NULL             |
| 2038138 | Griffin, Kathy       | Hit-and-run Witness                | 4 November 1960  |
| 1510086 | Tarantino, Quentin   | Jimmie Dimmick                     | 27 March 1963    |
|  782985 | Keitel, Harvey       | Winston 'The Wolf' Wolfe           | 13 May 1939      |
| 2256312 | Maruyama, Karen      | Gawker #1                          | 29 May 1958      |
|  120919 | Bender, Lawrence     | Long Hair Yuppy Scum               | 17 October 1957  |
| 1426048 | Sitka, Emil          | Hold Hands You Lovebirds           | 22 December 1914 |
| 1033958 | Miller, Dick         | Monster Joe                        | 25 December 1928 |
+---------+----------------------+------------------------------------+------------------+
58 rows in set (20 min 49.17 sec)
```

11. Lista nombres de los capítulos y valoración de la serie “Breaking Bad” (id=175285) durante el año 2012.

```
+-----------------+------+
| title           | info |
+-----------------+------+
| Buyout          | 9.0  |
| Dead Freight    | 9.7  |
| Fifty-One       | 8.8  |
| Gliding Over All | 9.6  |
| Hazard Pay      | 8.9  |
| Live Free or Die | 9.3  |
| Madrigal        | 8.9  |
| Say My Name     | 9.4  |
+-----------------+------+
8 rows in set (31.18 sec)
```

12. Lista películas producidas en 2010 en las que el director haya participado también como actor. Presenta junto a la película el nombre de su director y su fechas de nacimiento (cod 21). (Extremadamente lenta sin indexar).

LABORATORIO 4.
CONSTRUCCIÓN DE ÍNDICES SOBRE IMDB

Objetivos. Tras cubrir esta sesión práctica, el alumno/a es capaz de diseñar una estrategia para la construcción de índices sobre una base de datos real con objeto de mejorar el rendimiento de consultas expresadas en SQL. En concreto el alumnado adquirirá habilidades para:

- ✓ Evaluar de modo efectivo el coste de una consulta.
- ✓ Aplicar el método three star indexing sobre consultas sencillas.
- ✓ Identificar columnas coincidentes y columnas de cribado.
- ✓ Construir índices óptimos para consultas que involucran una simple tabla.

1. INTRODUCCIÓN

A lo largo de esta práctica el alumno descubrirá por sí mismo los efectos de la indexación en el rendimiento de una base de datos. Para ello trabajaremos con la máquina virtual *imdb_no_index*, la cual dispone de la base de datos imdb en el que cada tabla incluye un único índice sobre su clave primaria. De este modo podremos ir valorando el efecto de incorporar nuevos índices para mejorar el rendimiento de las consultas.

Antes de comenzar vamos a desactivar el buffer de consultas de mysql para visualizar sin problemas el tiempo de ejecución de cada consulta particular:

```
SET SESSION query_cache_type = OFF;
```

Veamos qué sucede si emitimos una consulta simple que nos muestre las filas de la tabla *cast_info* correspondientes a todos los participantes en la película "Blade Runner" (movie_id = 1693255):

```
SELECT *
FROM cast_info
WHERE movie_id = 1693255;
```

En una máquina suficientemente potente, esta consulta realizada sobre la BD sin indexar viene a tardar alrededor de 15 segundos. La consulta se ha ejecutado de modo aislado, con un único cliente sobre la BD, imagina el impacto negativo que podría tener sobre los usuarios si la misma consulta es lanzada concurrentemente por una cantidad considerable de clientes. Como ves, se trata de una consulta simple, puedes intentar ejecutar ahora una consulta sólo un poco más exigente que muestre el nombre de los actores, su fecha de nacimiento y el personaje que interpreta en la película:

```
SELECT n.id, n.name, ch.name AS 'character name', pi.info AS
'birthdate'
FROM name n JOIN cast_info ci ON (ci.person_id = n.id)
 JOIN char_name ch ON (ch.id = ci.person_role_id)
 LEFT JOIN person_info pi ON (pi.person_id = n.id AND pi.in-
fo_type_id = 21)
WHERE ci.movie_id = 1693255
ORDER BY ci.nr_order;
```

¿Cuántos segundos invirtió el servidor para dar respuesta a esta "sencilla" consulta? (en mi máquina 9 minutos 30 segundos, ciertamente inasumible en un servidor en explotación).

```
| 1848923 | Cassidy, Joanna     | Zhora            | 2 August 1945    |
|  680487 | Hong, James         | Hannibal Chew    | 22 February 1929 |
| 1180309 | Paull, Morgan       | Holden           | 15 December 1944 |
| 1529222 | Thompson, Kevin     | Bear             | NULL             |
|   29043 | Allen, John Edward  | Kaiser           | NULL             |
| 1241701 | Pyke, Hy            | Taffey Lewis     | 2 December 1935  |
| 2081083 | Hiroshige, Kimiko   | Cambodian Lady   | 3 July 1912      |
| 1138508 | Okazaki, Bob        | Howie Lee        | 3 February 1902  |
| 1921727 | DeMirjian, Carolyn  | Saleslady        | NULL             |
+---------+---------------------+------------------+------------------+
31 rows in set (9 min 30.75 sec)
```

Es obvio que necesitamos indexar, veamos cómo hacerlo para conseguir mejorar el rendimiento de las consultas.

2. EL COSTE DE UNA CONSULTA SIMPLE

Antes de introducir un mecanismo sistemático que podamos aplicar para la construcción de índices sobre las tablas de nuestra BD, vamos a trabajar con algunos ejemplos que nos faciliten la comprensión del modo en que el DBMS utiliza el índice para responder a una consulta y aproximar el coste de responder a la consulta con y sin índice.

Considera la siguiente consulta, en la que mostramos títulos de series producidas en 1995:

```
SELECT title, production_year, kind_id
FROM title
WHERE production_year = 1995
AND kind_id = 2
ORDER BY title;
```

Supongamos que disponemos de un índice sobre *production_year*. El factor de filtrado o cardinalidad de este índice es de 0,75%, lo que significa que dado un valor concreto para *production_year*, el número aproximado de tuplas que obtenemos es de $0{,}0075 \times R(\text{title})$, un 0,75% del total de tuplas. Vamos a calcular el coste de responder a esta consulta a través de este índice.

Para responder esta consulta a través del índice, el procesador de consultas accederá al índice para obtener la rodaja correspondiente a 1995. Ahora, para cada entrada de dicha rodaja, el DBMS tendrá que examinar los valores de *kind_id* e *imbd_index*, para lo cual debe acceder a las tuplas de la tabla *title*. Pero tengamos en cuenta que este índice no es agrupado y por tanto las tuplas de *title* no están ordenadas por *production_year*, lo cual supone un acceso random a la tabla *title* para cada una de las entradas de la rodaja.

Supongamos que cada entrada de índice ocupa (incluyendo metadatos y espacio libre) unos 20 bytes. Dado que *title* tiene unos 2,5 millones de registros, el índice ocupará $2.500.000 \times 20$ bytes $\simeq$ 50 MB. Leer una rodaja del 0,75% supone leer unos 380KB. Teniendo en cuenta que una lectura aleatoria toma 10 ms y la lectura secuencial se hace a 40 MB/s, leer la rodaja de índice nos cuesta:

$$10 \text{ ms} + 380 \text{ Kb}/(40 \times 1024) \text{ Kb/s} = 20 \text{ ms}$$

El primer término de la ecuación (10 ms) es el tiempo invertido hasta localizar la primera entrada de índice. Suponemos pues que podemos hacerlo en un solo acceso, aunque si el índice en árbol B tiene más niveles, puede ser necesario invertir más de un acceso para llegar a la primera entrada.

El valor obtenido de 20 ms no es gran cosa, pero a este tiempo tenemos que añadir el acceso aleatorio a $0{,}75\% \times 2.500.000 = 18.750$ filas de la tabla base, lo cual supone un coste de:

$$18.750 \times 10 \text{ ms} = 187 \text{ s}$$

Veamos ahora cuál sería el coste de responder a esta misma consulta sin utilizar el índice sobre *production_year*. Si accedemos directamente a la tabla sin usar el índice, considerando que la tabla ocupa unos 200 MB el tiempo que invertimos será:

10 ms + 200 MB/40 MB/s = 5 s

dado que podemos llevar a cabo un acceso secuencial a la propia tabla. En este caso el procesador de consultas recorre cada fila de la tabla, una por una, chequeando si cumple el predicado de la consulta. Poder acceder de forma secuencial a la tabla para examinar el contenido de la fila resulta pues mucho más rentable que hacerlo a través de un índice. Es claro que una vez accedida la fila de la tabla hay un tiempo extra de procesamiento de CPU para chequear el predicado, pero aún así resulta mucho más rentable procesar directamente los datos de la tabla.

Como hemos visto, en este caso un full scan sería más rápido que usar un índice inadecuado sobre la tabla.

Pero ¿qué sucede si construimos el mejor índice posible para responder a esta consulta? ¿Conseguiríamos mejorar el rendimiento de un full scan de la tabla base? La respuesta es sí.

Un índice de tres estrellas (el mejor posible) para la consulta anterior estaría compuesto por las columnas (*production_year, kind_id, title*). Supongamos que existen 1600 filas de *title* que satisfacen el predicado de la consulta. Teniendo en cuenta que los predicados sobre las columnas *production_year* y *kind_id* son ambos de igualdad, la rodaja de índice que obtenemos finalmente es la más pequeña posible y contiene exactamente 1600 entradas. Considerando que el tamaño de una entrada de índice es de unos 60 bytes, la rodaja de índice ocupa 0,1 MB, por lo que el tiempo necesario para el acceso a la misma será:

10 ms + 0,1 MB /40 MB/s = 10 ms + 2,5 ms = 12,5 ms

Puesto que la información que debemos mostrar se encuentra en el propio índice no es necesario el acceso a la tabla. Adicionalmente, la consulta nos solicita mostrar los resultados ordenados por el título, pero dado que el atributo *title* aparece en el índice justo detrás de los atributos implicados en el predicado, los valores son mostrados en orden de *title*, por lo que tampoco se hace necesario realizar una operación extra de ordenación para finalmente responder a la consulta. Como vemos con este índice 3 estrellas conseguimos un speed-up de 416× (5000 ms/12 ms) más rapidez con índice que sin él.

3. CÓMO ELEGIR UN ÍNDICE

Antes de describir el método a utilizar para construir buenos índices para nuestra base de datos, conviene puntualizar que un índice puede ser óptimo para una consulta y completamente inútil para otra. En esta práctica aprenderemos a construir índices óptimos para *una* consulta. Mientras que el diseño del esquema relacional está basado en los datos, el diseño de índices está basado en las consultas.

El método que vamos a seguir se atribuye a Tapio Lahdenmäki y Michael Leach y está publicado en su libro *Relational Database Index Design and the Optimizer* (Wiley, 2005). Este método se conoce con el nombre de "Three Star Index Design" y está basado en la calificación de un índice con una dos o tres estrellas en función de su eficiencia para responder una consulta. En una primera aproximación podemos decir que el índice obtiene la primera estrella si las filas referenciadas por la consulta están agrupadas juntas en el índice, consigue la segunda estrella si las filas referenciadas por la consulta están ordenadas en el índice en la forma en que se desean mostrar y finalmente se cualifica con la tercera estrella si el índice contiene todas las columnas referenciadas por la consulta. En este caso se dice que tenemos un índice de cobertura o *covering index*.

Supongamos que queremos mostrar identificadores de personas junto a identificadores de personajes que interpretan los actores (no actrices) de la película "Pulp Fiction" (id=2175869) ordenados en función de la relevancia del papel en la misma.

Ejercicio 1

Escribe y ejecuta la consulta anterior.

□

Para responder esta consulta, el optimizador no cuenta con índice alguno, de modo que debe acceder a cada fila de la tabla *cast_info* (35,6 millones de filas) y para cada fila ver si cumple el predicado. Una vez obtenidas todas las filas que satisfacen el predicado debe realizar una ordenación de ese conjunto de filas respecto a la columna nr_order. En total como puede apreciarse en la imagen anterior, en una máquina bastante potente toma algo más de 12 segundos. Veamos cómo podemos aplicar el método de las tres estrellas para construir un índice que mejore sensiblemente el tiempo de ejecución de esta consulta.

Para cualificar con la primera estrella, hay que tomar todas las columnas con un predicado de igualdad en la cláusula WHERE y poner éstas como las primeras columnas de nuestro índice (en cualquier orden). De este modo se consigue que la rodaja de índice sea lo más delgada posible. En nuestro ejemplo tomaríamos todas las columnas que tienen un predicado de igualdad y colocaríamos dichas columnas en el índice (no ejecute esta sentencia):

```
CREATE INDEX pfroles_ndx ON cast_info (movie_id,role_id)
```

Para otorgar la segunda estrella a nuestro índice debemos asegurarnos de que las entradas de la rodaja de índice están en el orden en que deseamos mostrar los valores, de modo que no sea necesario aplicar un proceso adicional de ordenación. Esto significa que debemos añadir al índice las columnas presentes en la cláusula ORDER BY, en el mismo orden en que aparecen en dicha cláusula, aunque sin repetir aquellas que ya estuvieran presentes en el índice.

En nuestro ejemplo, dado que las filas se muestran en orden de relevancia del papel que interpretan los actores, debemos añadir a nuestro índice la columna *nr_order (*no ejecute esta sentencia*)*:

```
CREATE INDEX pfroles_ndx ON cast_info (movie_id,role_id,nr_
order)
```

Finalmente la tercera estrella se consigue cuando las entradas de índice contienen todas las columnas referidas por la cláusula SELECT, lo cual evita la necesidad de acceder a la tabla base. En nuestro ejemplo, para conseguir la tercera estrella para nuestro índice *pfroles_ndx* debemos añadir las columnas *person_id* y *person_role_id* (ejecute finalmente esta sentencia):

```
CREATE INDEX pfroles_ndx
    ON cast_info (movie_id,role_id,nr_order,person_id, person_
role_id)
```

Como habrá podido observar, la sentencia anterior consume un tiempo considerable en el servidor (en el caso de mi máquina con procesador intel i7 a 4,2 Ghz y 8 Gb de RAM, ha consumido 10 seg):

```
|   1562476 |         435983 |
|   1510086 |        1923000 |
|    782985 |        1108347 |
|    120919 |         205436 |
|   1426048 |        1833730 |
|   1033958 |         664620 |
+-----------+----------------+
37 rows in set (10.11 sec)
```

Si volvemos a lanzar la consulta anterior con el índice anteriormente construido, ahora el procesador de consultas invierte 0,01 seg:

```
|   1332235 |        1734407 |
|   1562476 |         435983 |
|   1510086 |        1923000 |
|    782985 |        1108347 |
|    120919 |         205436 |
|   1426048 |        1833730 |
|   1033958 |         664620 |
+-----------+----------------+
37 rows in set (0.01 sec)
```

Consiguiendo así un speed-up de 1011× más rápidez.

La consulta que hemos utilizado para nuestro ejemplo contenía en la cláusula WHERE exclusivamente predicados de igualdad, pero ¿qué sucede con los predicados por rango? ¿Es posible conseguir índices de tres estrellas para consultas con predicados por rango? La respuesta es desafortunadamente no, precisamente porque la segunda estrella no se puede garantizar en caso de predicado por rango, pero eso no significa que no podamos construir índices con la primera y tercera estrella. Veamos un ejemplo.

Supongamos que queremos visualizar título y código de la clase de obra (*kind_id*) para telefilmes y series de televisión producidos en 2005 y ordenados alfabéticamente por el título.

Ejercicio 2

Escribe y ejecuta la consulta anterior.

En esta consulta, para conseguir la primera estrella añadimos la columna *production_year*, la cual tiene un predicado de igualdad que nos permite reducir la rodaja de índice. ***La primera estrella obliga a que las columnas con predicado de igualdad se coloquen siempre al principio del índice, por delante de cualesquiera otras***. Si además añadimos la columna *kind_id*, sobre la cual se realiza un predicado por rango, estamos consiguiendo reducir aún más nuestra rodaja.

production_year	*kind_id*	*title*
1995	1	Watch Me
1995	2	Inka Connection
1995	2	Sailor Moon
1995	3	Bloodknot
1995	3	In the Heat of the Night
1996	1	101 Dalmatians

En la tabla anterior marcamos en verde las filas que satisfacen el predicado. En este ejemplo podemos apreciar con claridad que dado que la primera estrella obliga a ordenar el índice por (*production_year, kind_id*) no resulta posible obtener las filas ordenadas por título, lo cual impide que el índice tenga la segunda estrella. En este caso las estrellas uno y dos son incompatibles. Si deseáramos la segunda estrella para nuestro índice, la columna *title* debería aparecer ***antes*** que la columna *kind_id* sobre la cual hemos establecido un predicado por rango, pero esto haría que la rodaja de índice fuera más grande, consumiendo pues más tiempo en la ejecución de la consulta.

La conclusión es que con un predicado por rango, no podemos disponer de un índice 3 estrellas y habrá que optar por tener un índice con la primera o la segunda estrella, ya que ambas no son posibles. En términos generales conviene decir que la primera estrella resulta normalmente más impactante sobre el rendimiento de la consulta que la segunda estrella, por lo que en general optaremos siempre por la primera.

Del ejemplo anterior se extraen dos reglas importantes:

1. Ante casos de incompatibilidad entre primera y segunda estrella, optamos siempre por la primera estrella.
2. Para conseguir la primera estrella, en caso de predicados por rango, primero se colocan en el índice las columnas con predicados de igualdad y posteriormente las columnas con predicados por rango.

EJERCICIO 3

Construye el mejor índice posible para la consulta anterior e indica la asignación de estrellas del mismo. Ten en cuenta que la columna *title* es de tipo TEXT y MySql no permite indexar columnas BLOB/TEXT completas (ocupan un máximo de 64 Kb). Este tipo de columnas se indexan mediante un prefijo suficientemente discriminativo para dicha columna. Para examinar la capacidad de discriminación de un prefijo podemos contar los valores distintos que nos encontramos con dicho prefijo y dividirlo por el total de filas. En nuestro caso, probamos con un prefijo de 30 caracteres:

```
SELECT 100*COUNT(DISTINCT SUBSTR(title,1,30))/COUNT(DISTINCT title)
FROM title
```

```
+----------------------------------------------------------+
| 100*COUNT(DISTINCT SUBSTR(title,1,30))/COUNT(DISTINCT title) |
+----------------------------------------------------------+
|                                                  99.1919 |
+----------------------------------------------------------+
1 row in set (29.03 sec)
```

El 99% de discriminación del prefijo de 30 caracteres sobre *title* parece suficiente para conseguir el orden pretendido.

□

4. LA IMPORTANCIA DEL ACCESO A LA TABLA BASE

Un aspecto importante a tener en cuenta en la construcción del índice es el coste en que se incurre tras el acceso al índice para recuperar finalmente las filas de la tabla que satisfacen la consulta. Hasta ahora hemos hablado de la importancia de conseguir una rodaja de índice lo más delgada posible, ya que mientras más pequeña sea la rodaja, más rápido se accederá al índice (este acceso es secuencial una vez alcanzada la primera entrada). Así pues, el coste de la consulta depende en gran medida del grosor de la rodaja de índice que consigamos, pero el factor que más afecta no es precisamente el coste de acceso a la rodaja, sino el coste de las lecturas síncronas que hacen falta para alcanzar las filas de la tabla base. Por tanto, ***una rodaja***

fina reducirá el coste de acceso al índice, pero lo verdaderamente importante es que ***reducirá el número de lecturas síncronas a la tabla base***.

Dicho esto, un concepto importante que tiene implicaciones sobre el grosor de la rodaja es el número de *columnas coincidentes* que se usan en el índice, lo que se conoce como *matching columns*. Una columna se define como coincidente si tiene capacidad de reducir el grosor de la rodaja. Es claro que columnas con predicado de igualdad son coincidentes, pero también aquellas que aparecen en predicados por rango lo son. Sin embargo existen otras columnas no coincidentes que no tienen capacidad para reducir la rodaja de índice. Consideraremos el siguiente ejemplo.

Supongamos una tabla con columnas A, B, C, D con los siguientes valores:

A	*B*	*C*	*D*
a	99	c	d1
a	100	f	d2
b	110	g	d1
a	104	c	d3
a	105	b	d1
a	110	c	d8
b	103	d	d3
a	104	g	d4

Y consideremos la consulta:

```
SELECT A,B,C
FROM tabla
WHERE A = a AND C IN (c,g) AND B BETWEEN 103 AND 110;
```

¿Cómo abordamos la construcción de un índice óptimo para esta consulta?

Tal y como hemos estudiado con anterioridad, la columna A con predicado de igualdad debe ir la primera en el índice. Ordenando la tabla por esta columna, la rodaja de índice que conseguimos (marcada en verde) es:

A	*B*	*C*	*D*
a	99	c	d1
a	100	f	d2
a	104	c	d3
a	105	b	d1
a	110	c	d8
a	104	g	d4
b	110	g	d1
b	103	d	d3

Es claro que A es una columna coincidente. Por otra parte tenemos en la cláusula WHERE un predicado por rango sobre B, añadimos esta columna a nuestro índice para conseguir una rodaja más fina. Ordenando ahora por A,B, tenemos la siguiente rodaja:

A	*B*	*C*	*D*
a	99	c	d1
a	100	f	d2
a	104	g	d4
a	104	c	d3
a	105	b	d1
a	110	c	d8
b	103	d	d3
b	110	g	d1

Lo que nos indica que la columna B también es coincidente. Por último tenemos un predicado que no es de igualdad ni rango sobre la columna *C*, por lo que no nos permite reducir más la rodaja. Pero ¿qué sucede si añadimos *C* a nuestro índice? En este caso, ordenando por (A, B, C) tenemos:

A	*B*	*C*	*D*
a	99	c	d1
a	100	f	d2
a	104	c	d3
a	104	g	d4
a	105	b	d1
a	110	c	d8
b	103	d	d3
b	110	g	d1

En este caso tenemos las filas que satisfacen el predicado ya no definen una rodaja. La columna C no tiene capacidad de reducir la rodaja, por lo que ya no es una columna coincidente. Sin embargo esto no significa que no sea útil la introducción de la columna C en nuestro índice. El procesador de consulta accederá a la rodaja de índice constituida por las filas:

a	104	c	d3
a	104	g	d4
a	105	b	d1
a	110	c	d8

Y posteriormente utilizará el valor de la columna C para *cribar* los accesos a la tabla base, ahorrándose en este caso un acceso a la tabla base correspondiente a la tercera entrada de la rodaja.

Así pues, las columnas que no tienen capacidad de reducir la rodaja pero sí evitan accesos a la tabla base, se denominan columnas de *cribado* o *screening columns*.

Ahora que conocemos el método *three star indexing*, intenta aplicarlo en las siguientes consultas.

5. EJERCICIOS FINALES

En una franja determinada de tiempo, el interés de los usuarios se concentra en obtener determinados valores que el abd ha decidido materializar en una tabla nueva llamada *rgc*. Para cada obra almacenada en la tabla *title*, la nueva tabla *rgc* registra su identificador, la clase de obra que es, su año de producción, información de episodio, temporada y número, además de tres columnas adicionales que indican la puntuación otorgada por los usuarios a la obra, la nacionalidad de la obra y el género de la misma. Dado que tanto la nacionalidad como el género puede no ser único para una obra, la tabla anterior puede incluir bastante replicación. Por otra parte, algunos de los valores de las columnas adicionales pueden ser nulos, pero al menos uno de los tres campos adicionales debe tener un valor no nulo.

Ahora sobre la tabla *rgc* realiza las consultas que se indican. Para cada una de ellas, registra el tiempo de ejecución y construye un índice óptimo siguiendo el método *three star indexing*. Explica detalladamente cómo el índice creado ayuda a ejecutar la consulta de modo eficiente. Crea finalmente el índice y ejecuta de nuevo la consulta registrando el tiempo de ejecución y el speed-up que se consigue.

EJERCICIO 4

Obtén un listado de identificadores junto a su año de producción y calificación otorgada de películas de cine y televisión españolas valoradas por los usuarios con una calificación igual o superior a 7.0. Ordena la salida por año de producción.

□

EJERCICIO 5

Lista identificadores de series españolas junto con la máxima puntuación otorgada por los usuarios a cualquiera de sus capítulos. Si los usuarios no han otorgado calificación a ninguno de sus capítulos, la serie no se presenta en la lista.

□

Ejercicio 6

Lista identificadores de películas de terror españolas y portuguesas de 2010 ordenadas en orden descendente de rating. Muestra en el listado el identificador, nacionalidad, año y la puntuación de la película. Las películas no puntadas no aparecen en el listado.

□

LABORATORIO 5.
OPTIMIZACIÓN DE CONSULTAS

Objetivos. Tras cubrir esta sesión práctica, el alumno/a habrá adquirido habilidades específicas para mejorar el rendimiento de consultas complejas y más concretamente de consultas que contienen JOINs.

1. INTRODUCCIÓN

En la práctica anterior se introdujo el método *three star indexing* para construir índices capaces de mejorar el rendimiento de las consultas. En el desarrollo de esta práctica profundizaremos en la aplicación de este método, experimentando sobre una colección de consultas complejas, con el fin de obtener planes de ejecución óptimos. Igual que en prácticas anteriores trabajaremos con la máquina virtual *imdb_no_index*.

Tal y como hicimos en la práctica anterior, vamos a desactivar el buffer de consultas de mysql, pero en este caso lo haremos modificando el fichero de configuración my.cnf para hacer permanente esta característica. Para ello, editamos desde la cuenta root del sistema el archivo /etc/mysql/my.cnf. Accedemos a la sección [mysqld], comentamos la línea query_cache_size y añadimos las siguientes dos líneas:

```
#query_cache_size          = 16M
query_cache_size           = 0
query_cache_type           = 0
```

2. EJECUCIÓN DE JOINS EN MYSQL

Si bien es responsabilidad del programador sacar el máximo partido al lenguaje de consultas expresando convenientemente las necesidades de información de los usuarios, el encargado de conseguir un alto nivel de eficiencia por parte del SGBD en la respuestas a dichas es sin duda el administrador de la base de datos. Para dotar de eficiencia al sistema, el ABD debe conocer el funcionamiento del procesador de consultas, disponer todos los recursos, tanto de indexación como de estructura de la consulta y conseguir que el optimizador pueda desplegar su máxima potencia.

El JOIN es la operación por excelencia en las bases de datos relacionales, por ello resulta imprescindible conocer el modo en que nuestro SGBD procede para llevar a cabo esta operación. En el caso concreto que nos ocupa, MySQL utiliza un algoritmo de bucles anidados (*nested loop*) también conocido como barrido simple (*single sweep*). Esto significa que MySQL lee una fila de la primera tabla y encuentra una fila coincidente de la segunda tabla y una de la tercera tabla y así sucesivamente con todas las tablas del JOIN. Cuando se finaliza el barrido sobre todas las tablas del JOIN, se mandan a la salida las columnas seleccionadas y se procede hacia atrás en la lista de tablas hasta que se encuentre una tabla para la cual existen más filas coincidentes. Entonces se lee la siguiente fila de esa tabla y el proceso continúa con la siguiente tabla.

3. OPTIMIZACIÓN DEL JOIN MEDIANTE ÍNDICES

Deseamos obtener un listado de identificadores de películas y telefilmes españoles producidos en el año 2012. Para expresar esta consulta en SQL necesitamos realizar una operación JOIN entre las tablas *title* y *movie_info*:

```
SELECT t.id
FROM title t JOIN movie_info mi ON (t.id=mi.movie_id)
WHERE t.production_year = 2012 AND t.kind_id IN (1,3)
AND mi.info_type_id = 8 AND mi.info ='Spain'
ORDER BY t.id
```

El número de filas y el tiempo invertido en la misma en un ordenador con procesador Intel i7 a 4,2 Ghz y 8 Gb de memoria RAM es el que muestra la figura:

```
|  2471948 |
|  2471976 |
|  2472568 |
|  2472672 |
|  2473789 |
|  2474817 |
+----------+
1204 rows in set (13.74 sec)
```

El optimizador tiene dos opciones para responder esta consulta. La primera es hacer un full scan de la tabla *title*. Si la fila no satisface el predicado *"t.production_year = 2012 AND t.kind_id IN (1,3)"* entonces puede descartarse, en caso contrario, tomamos el valor de *title.id*. Con ese valor, llevamos a cabo un full scan de la tabla *movie_info*, si la fila correspondiente tiene un valor de *movie_id* diferente al obtenido previamente, podemos descartarla, en caso contrario comprobamos si además satisface el predicado *"mi.info_type_id = 8 AND mi.info ='Spain'"*. En caso afirmativo sacamos el valor de *t.id* a la salida y proseguimos.

La segunda opción comienza haciendo un full scan por la tabla *movie_info*. Si la fila satisface el predicado *"mi.info_type_id = 8 AND mi.info ='Spain'"*, entonces tomamos el valor correspondiente de *movie_id*. Ahora dado que la tabla *title* tiene un índice por la columna *id* (es clave primaria), podemos acceder al índice, buscando el valor obtenido en *movie_id*. La entrada del índice nos dirige rápidamente a la tabla base, donde examinamos si la fila correspondiente satisface el predicado *"t.production_year = 2012 AND t.kind_id IN (1,3)"*, en caso positivo sacamos el valor de *t.id* a la salida y proseguimos con la siguiente fila de *movie_info*.

Es claro que la primera opción es bastante más costosa en tiempo que la segunda, ya que aquella conlleva un full scan de *movie_info* para cada fila de *title*, lo cual supone la friolera de $2{,}5 \times 10^6 \times 14{,}6 \times 10^6 = 36{,}5 \times 10^{12}$ accesos y en el segundo caso sólo hacemos un full scan de *movie_info* y un único acceso a cada tupla de *title* a través de su índice.

Para comprobar de modo efectivo el tiempo que conllevan ambas alternativas, podemos reescribir la consulta mediante el uso de USE INDEX y STRAIGHT JOIN. La primera cláusula obliga a considerar una lista de índices concretos y la segunda obliga al procesador de consultas a realizar el join en el orden en que aparecen en la consulta. De este modo, si deseamos ejecutar la consulta siguiendo la primera opción escribimos:

```
SELECT t.id
FROM title t USE INDEX (PRIMARY) STRAIGHT JOIN movie_info mi
USE INDEX (PRIMARY) ON (t.id=mi.movie_id)
WHERE t.production_year = 2012 AND t.kind_id IN (1,3)
AND mi.info_type_id = 8 AND mi.info ='Spain'
ORDER BY t.id;
```

El resultado es este:

```
| 2472568 |
| 2472672 |
| 2473789 |
| 2474817 |
+---------+
1203 rows in set (5 min 50.87 sec)
```

Optando por la segunda vía:

```
SELECT t.id
FROM movie_info mi USE INDEX (PRIMARY) STRAIGHT JOIN title t
USE INDEX (PRIMARY) ON (t.id=mi.movie_id)
WHERE t.production_year = 2012 AND t.kind_id IN (1,3)
AND mi.info_type_id = 8 AND mi.info ='Spain'
ORDER BY t.id;
```

obtenemos este resultado:

```
| 2472568 |
| 2472672 |
| 2473789 |
| 2474817 |
+---------+
1203 rows in set (22.80 sec)
```

El método *three star indexing* es aplicable sobre una tabla concreta, pero ¿cómo podemos sacar partido de este método en una consulta conteniendo un JOIN como la anterior?

En este caso al tener un join, debemos analizar la indexación de cada tabla participante en el join. Existen dos vías para ejecutar esta consulta.

PRIMERA VÍA

Comencemos analizando el acceso desde la tabla *title*. En la cláusula WHERE de nuestra consulta aparece un predicado de igualdad sobre *production_year* (columna coincidente), de modo que esta sería la primera columna de nuestro índice sobre *title*. Por su parte, *kind_id* es una columna de cribado ya que no permite reducir la rodaja, pero según vimos en el apartado 4, también conviene incluirla en el índice. De momento el índice (*production_year,*

kind_id) sobre *title*, tendría la primera estrella. Para dotarle de la segunda estrella, nuestro índice debe obtener los valores por orden de *title.id*. Esto supone añadir la columna *id* en el índice, pero si la incorporamos detrás de *kind_id*, dado que esta última no es una columna coincidente, las entradas de nuestro índice no aparecerían ordenadas por *id*. Así pues, para conseguir la segunda estrella debemos incorporar la columna *id* por delante de *kind_id*. Démonos cuenta que esta incorporación no afecta a la primera estrella ya que *kind_id*, como hemos dicho ya no es una columna coincidente. Así pues el índice (*production_year, id, kind_id*) tiene de momento dos estrellas. Teniendo en cuenta además que no necesitaremos acudir a la tabla base para obtener los valores necesarios de la fila para presentarlos en el resultado, nuestro índice es en realidad de 3 estrellas. Con este índice que llamaremos *pik_ndx,* accederíamos a la rodaja dada por *production_year*=2012 y para cada entrada que tuviera un valor *kind_id* de 1 ó 3, tomaríamos su *id* para acceder a los datos de *movie_info*.

Veamos ahora cómo podemos indexar óptimamente la tabla *movie_info* para conseguir reducir el tiempo de ejecución de nuestra consulta. Comenzando la ejecución por el acceso al índice *pik_ndx*, el cual contiene toda la información necesaria, una vez obtenido los valores de *title.id*, debemos ser capaz de localizar rápidamente las tuplas candidatas a la concatenación. La única vía de hacerlo es construyendo un índice (*movie_id*) sobre la tabla *movie_info*. Este índice nos evita un full scan sobre la tabla base *movie_info* para cada valor obtenido anteriormente de *title.id*. En este caso la operación join manda sobre el resto de predicados en los que aparecen columnas de *movie_info*. De nada sirve construir un índice en *movie_info* que no tenga como primera columna *movie_id*, ya que no sería de utilidad para mejorar el tiempo de ejecución del join. Dicho esto, si incorporamos en este índice las columnas *info_type_id* e *info*, evitaríamos accesos a la tabla base, ya que tendríamos en el índice toda la información necesaria para evaluar el predicado. Sin embargo, la columna *info* es de tipo Text y no puede ser indexada como tal, se precisa un prefijo sobre el que indexar. Siguiendo el método descrito en la página 50, estudiamos la capacidad de discriminación de un prefijo no excesivamente largo:

```
SELECT 100*COUNT(DISTINCT SUBSTR(info,1,20))/COUNT(DISTINCT
info)
FROM movie_info
WHERE info_type_id = 8;
```

```
+--------------------------------------------------------+
| 100*COUNT(DISTINCT SUBSTR(info,1,20))/COUNT(DISTINCT info) |
+--------------------------------------------------------+
|                                                99.5652 |
+--------------------------------------------------------+
1 row in set (13.28 sec)
```

Como vemos un prefijo de 20 cubre el 99,5% de los nombres de países, lo cual es suficiente.

Construimos entonces un índice sobre la tabla *movie_info* que denominamos *mii_ndx* (*movie_id, info_type_id, info(20)*). En este caso, no hemos podido seguir el método *three star indexing* debido a que el acceso a *movie_info* venía marcado por el join.

Emitimos las sentencias para construir ambos índices:

```
CREATE INDEX pik_ndx ON title (production_year, id, kind_id);
CREATE INDEX mii_ndx ON movie_info (movie_id, info_type_id,
info(30));
```

Y ejecutamos la consulta mirando los resultados y el tiempo de ejecución:

```
|  2471948 |
|  2471976 |
|  2472568 |
|  2472672 |
|  2473789 |
|  2474817 |
+----------+
1204 rows in set (0.54 sec)
```

Con lo que conseguimos un 8,9 de speed-up.

SEGUNDA VÍA

La segunda vía para la indexación consiste en acceder primeramente a la tabla *movie_info*. En este caso tenemos dos columnas coincidentes, por lo que comenzaremos añadiendo a nuestro índice las columnas (*info_type_id,info(20)*), que conseguirán reducir la rodaja al menor grosor posible. Puesto que utilizamos el valor *movie_id* para acceder a la tabla *title*, incorporamos *movie_id* como tercera columna de nuestro índice. Como además se piden los resultados ordenados por *t.id*, el cual es coincidente con *movie_id*, el índice *itimid_ndx* (*info_type_id,info(20), movie_id*) es un índice tres estrellas (podemos modificar la cláusula SELECT de nuestra consulta para que en lugar de mostrar *t.id* muestre *mi.movie_id*, puesto que ambos valores coincidirán tras el join).

Una vez accedida la rodaja del índice *itimid_ndx*, a partir del valor de *movie_id* debemos buscar en la tabla *title* las filas cuyo *id* coincida con este valor y examinar los valores de *production_year* y *kind_id* para chequear si satisfacen o no el predicado. Aunque sobre *t.id* ya tenemos un índice (por ser clave primaria), si integramos en un nuevo índice los valores de *production_year* y *kind_id,* nos ahorraremos el acceso a la tabla base. De este modo construimos el índice *idpk_ndx* (*id, production_year, kind_id*). Igual que antes, este segundo

índice no puede constituirse según las reglas del three star indexing, ya que su acceso está motivado por la operación de join. Veamos la ganancia que obtenemos con la utilización de ambos índices.

```
CREATE INDEX itimid_ndx ON movie_info (info_type_id,info(20),
movie_id);
CREATE INDEX idpk_ndx ON title (id, production_year, kind_id);
SELECT mi.id
FROM title t USE INDEX (idpk_ndx)
JOIN movie_info mi USE INDEX (itimid_ndx) ON (t.id=mi.movie_id)
WHERE t.production_year = 2012 AND t.kind_id IN (1,3)
AND mi.info_type_id = 8 AND mi.info ='Spain'
ORDER BY mi.id;
```

```
|  2471976 |
|  2472568 |
|  2472672 |
|  2473789 |
|  2474817 |
+----------+
1204 rows in set (0.58 sec)
```

Presentamos a modo de resumen una tabla que recoge los tiempos de ejecución invertidos en la consulta mediante la utilización de los diferentes índices creados.

Consulta	*Índices*	*Tiempo*	*Speed-up*
SELECT mi.movie_id FROM title t IGNORE INDEX (py_kid_id_ndx,idpk_ndx,pik_ndx) JOIN movie_info mi IGNORE INDEX (movie_id_ndx,itimid_ndx,mii_ndx) ON (t.id=mi.movie_id) WHERE t.production_year = 2012 AND t.kind_id IN (1,3) AND mi.info_type_id = 8 AND mi.info ='Spain' ORDER BY mi.movie_id;	Clave primaria	13,74 seg	1

<table>
<tr><th>Consulta</th><th>Índices</th><th>Tiempo</th><th>Speed-up</th></tr>
<tr><td><pre>SELECT mi.movie_id
FROM title t FORCE INDEX (pik_ndx)
    JOIN movie_info mi FORCE INDEX
(mii_ndx) ON (t.id=mi.movie_id)
WHERE t.production_year = 2012 AND
t.kind_id IN (1,3)
AND mi.info_type_id = 8 AND mi.info
=’Spain’
ORDER BY mi.movie_id;</pre></td><td>title (production_ year, id, kind_id) movie_info (movie_id, info_type_id, info(20))</td><td>0,54</td><td>25,4</td></tr>
<tr><td><pre>SELECT mi.movie_id
FROM title t FORCE INDEX (idpk_ndx)
    JOIN movie_info mi FORCE INDEX
(itimid_ndx) ON (t.id=mi.movie_id)
WHERE t.production_year = 2012 AND
t.kind_id IN (1,3)
AND mi.info_type_id = 8 AND mi.info
=’Spain’
ORDER BY mi.movie_id;</pre></td><td>title (id, production_year, kind_id) movie_info (info_type_ id,info (20), movie_id)</td><td>0,58</td><td>23,6</td></tr>
</table>

A partir de este ejemplo y siguiendo las técnicas que hemos introducido, vamos a experimentar con la creación de índices óptimos sobre consultas que conlleven la ejecución de JOINs.

4. EJERCICIOS FINALES

Ejercicio 1

Construye índices que permitan optimizar la consulta “Lista nombres de actores/actrices junto al nombre del personaje que interpretaron en la película ‘Blade Runner’, cuyo identificador es 1693255. Obtén el listado ordenado según la importancia del papel representado en la película”. Presenta una tabla que muestre los tiempos de ejecución sin índice y con índice, indicando el speed-up conseguido.

□

Ejercicio 2

Optimiza mediante índices la consulta “Listado de películas producidas en 2010 en las que el director participa como actor, incluyendo también el nombre del director”.

□

LABORATORIO 6.
LA SENTENCIA EXPLAIN

Objetivos. Tras cubrir esta sesión práctica, el alumno/a es capaz de interpretar el modo en que el optimizador de consultas de MySql ejecuta una sentencia SELECT. Este conocimiento le permitirá medir posteriormente el impacto de la creación de un nuevo índice sobre las tablas de la sentencia SELECT. El estudiante adquirirá habilidades para utilizar la sentencia EXPLAIN como herramienta básica de análisis del rendimiento de una consulta.

1. INTRODUCCIÓN A EXPLAIN

A lo largo de esta práctica el alumno/a aprenderá a interpretar la salida de la sentencia EXPLAIN, la cual como su nombre indica explica el plan de ejecución de una sentencia SELECT. Para ello trabajaremos con la máquina virtual *imdb_no_index*, la cual dispone de la base de datos *imdb* en el que cada tabla incluye un único índice sobre su clave primaria.

Como en sesiones anteriores, desactivamos el buffer de consultas de mysql.

La sentencia EXPLAIN informa sobre el modo en que MySql va a ejecutar una sentencia SELECT. Es por ello que representa una buena herramienta para optimizar y mejorar el rendimiento en la ejecución de nuestras consultas sobre la base de datos. A través de la sentencia EXPLAIN, el administrador de la BD puede apreciar si el optimizador utiliza o no de la forma esperada determinados índices definidos sobre las tablas que participan en la consulta y consecuentemente decidir la incorporación de nuevos índices sobre las tablas o sencillamente modificar la expresión de las consultas para conseguir mejores tiempos de búsqueda.

EXPLAIN muestra como resultado una tabla conteniendo una línea de información para cada tabla utilizada en la sentencia SELECT. Las tablas se muestran en el mismo orden en el que MySQL las leería al procesar la consulta.

Comprobemos el resultado de la sentencia

```
EXPLAIN
SELECT *
FROM cast_info
WHERE movie_id = 1693255;
```

Nos aparece una tabla como ésta:

```
+----+-------------+-----------+------+---------------+------+---------+------+----------+-------------+
| id | select_type | table     | type | possible_keys | key  | key_len | ref  | rows     | Extra       |
+----+-------------+-----------+------+---------------+------+---------+------+----------+-------------+
|  1 | SIMPLE      | cast_info | ALL  | NULL          | NULL | NULL    | NULL | 35645119 | Using where |
+----+-------------+-----------+------+---------------+------+---------+------+----------+-------------+
1 row in set (0.00 sec)
```

La primera información que nos muestra la tabla anterior es que contiene una única fila, correspondiente a la única tabla que precisa accederse para responder a la consulta. Veamos uno a uno el significado de los diferentes campos de esta tabla.

2. EL CAMPO ID

El primer campo *id* es el identificador de la sentencia SELECT dentro de la consulta, de modo que si la consulta incluye más de una SELECT cada una tendrá un identificador diferente. Por ejemplo:

```
EXPLAIN
SELECT *
FROM cast_info
WHERE movie_id =
    (SELECT id
    FROM title
    WHERE title = 'Calabuch');
```

```
+----+-------------+-----------+------+---------------+------+---------+------+----------+-------------+
| id | select_type | table     | type | possible_keys | key  | key_len | ref  | rows     | Extra       |
+----+-------------+-----------+------+---------------+------+---------+------+----------+-------------+
|  1 | PRIMARY     | cast_info | ALL  | NULL          | NULL | NULL    | NULL | 35645119 | Using where |
|  2 | SUBQUERY    | title     | ALL  | NULL          | NULL | NULL    | NULL |  2477301 | Using where |
+----+-------------+-----------+------+---------------+------+---------+------+----------+-------------+
```

En este caso, el resultado muestra dos filas, cada una con un identificador diferente. El identificador 1 hace referencia a la tabla correspondiente a la sentencia SELECT principal (la más exterior), mientras que el identificador 2 hace referencia a la tabla que se accede en la subconsulta.

EJERCICIO 1

Construye una consulta que liste el nombre de la compañía productora de la película situada en la posición 1 del top 250. Utiliza al menos una subconsulta y muestra la salida de la sentencia explain, para que presente diferentes identificadores. Ejecuta finalmente la consulta para averiguar la salida.

□

3. EL CAMPO SELECT_TYPE

Este campo hace referencia al tipo de select en que participa la tabla correspondiente a la fila en cuestión. La siguiente tabla muestra los posibles valores que puede tomar esta columna.

Posibles valores para la columna select_type en la salida de EXPLAIN. De MySQL 5.0 Reference Manual (dev.mysql.com/doc/refman/5.0/en/explain-output.html)	
select_type **Value**	**Meaning**
SIMPLE	Simple **SELECT** (not using **UNION** or subqueries)
PRIMARY	Outermost **SELECT**
UNION	Second or later **SELECT** statement in a **UNION**
DEPENDENT UNION	Second or later **SELECT** statement in a **UNION** dependent on outer query
UNION RESULT	Result of a **UNION**
SUBQUERY	First **SELECT** in subquery
DEPENDENT SUBQUERY	First **SELECT** in subquery, dependent on outer query
DERIVED	Derived table **SELECT** (subquery in **FROM** clause)
UNCACHEABLE SUBQUERY	A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query

El tipo SIMPLE hace referencia a una SELECT sencilla, la cual no forma parte de una operación UNION ni contiene subconsultas. Un ejemplo lo vimos al principio de la sesión.

El tipo PRIMARY identifica tablas pertenecientes a la SELECT más externa, mientras que el tipo SUBQUERY referencia tablas en sentencias SELECT anidadas dentro de otras como subconsultas. Ambos tipos lo hemos tratado en el apartado anterior.

El tipo DEPENDENT SUBQUERY aparece cuando la SELECT correspondiente a la subconsulta contiene alguna referencia a tablas no utilizadas en dicha subconsulta. Este tipo de SELECT aparece por ejemplo en la siguiente consulta. Queremos saber cuantos actores y cuantas actrices protagonizan obras, es decir aparecen como primer actor o actriz en las diferentes obras almacenadas en imdb.

```
explain SELECT role, (
    SELECT COUNT(*)
    FROM cast_info ci
    WHERE ci.role_id = rt.id
    AND nr_order = 1) as 'numero'
FROM role_type rt
WHERE id IN (1,2);
```

La salida que obtenemos es:

```
+----+--------------------+-------+-------+---------------+---------+---------+------+----------+-------------+
| id | select_type        | table | type  | possible_keys | key     | key_len | ref  | rows     | Extra       |
+----+--------------------+-------+-------+---------------+---------+---------+------+----------+-------------+
|  1 | PRIMARY            | rt    | range | PRIMARY       | PRIMARY | 4       | NULL |        2 | Using where |
|  2 | DEPENDENT SUBQUERY | ci    | ALL   | NULL          | NULL    | NULL    | NULL | 35645119 | Using where |
+----+--------------------+-------+-------+---------------+---------+---------+------+----------+-------------+
```

Podemos observar cómo aparece una subconsulta dependiente ya que en su cláusula WHERE se referencia a una tabla de la SELECT más exterior.

El tipo UNION se muestra cuando se utiliza una operación UNION entre dos o más tablas. Por ejemplo, deseamos mostrar el títulos original y todos los títulos alternativos de la película cuyo identificador es 1930235:

```
EXPLAIN SELECT title
FROM title
WHERE id = 1930235
UNION
SELECT title
FROM aka_title
WHERE movie_id = 1930235
```

La salida que muestra el operador es:

```
+----+--------------+------------+-------+-----------------+----------+---------+-------+------+-------+
| id | select_type  | table      | type  | possible_keys   | key      | key_len | ref   | rows | Extra |
+----+--------------+------------+-------+-----------------+----------+---------+-------+------+-------+
|  1 | PRIMARY      | title      | const | PRIMARY,idpk_ndx | PRIMARY  | 4       | const |    1 |       |
|  2 | UNION        | aka_title  | ref   | movie_id        | movie_id | 4       | const |    7 |       |
| NULL | UNION RESULT | <union1,2> | ALL  | NULL  | NULL | NULL | NULL | NULL |
+----+--------------+------------+-------+-----------------+----------+---------+-------+------+-------+
```

La primera fila hace referencia al SELECT escrito en primer lugar. El identificador número 2 referencia a la cláusula SELECT tras el operador UNION y finalmente la última fila tiene identificador nulo y hace referencia a la tabla resultante de la unión, mostrándola del tipo UNION RESULT.

Similar a DEPENDENT SUBQUERY, el tipo DEPENDENT UNION aparece cuando en la operación UNION aparece dentro de una subconsulta. Por ejemplo, queremos obtener un listado de los títulos alternativos de películas españolas de 2013 así como de las cinco películas mejores del top 250. Podemos escribir:

```
EXPLAIN SELECT title
FROM aka_title
WHERE movie_id IN (
    SELECT t.id FROM title t JOIN movie_info mi
                ON (t.id = mi.movie_id)
        WHERE t.production_year = 2013 AND t.kind_id = 1
        AND mi.info_type_id = 8 AND mi.info='Spain'
    UNION
    SELECT t.id FROM title t JOIN movie_info_idx mix
                ON (t.id = mix.movie_id)
        WHERE t.kind_id = 1 AND mix.info_type_id = 112
        AND info < 5);
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	aka_title	ALL	NULL	NULL	NULL	NULL	358335	Using where
2	DEPENDENT SUBQUERY	t	eq_ref	PRIMARY	PRIMARY	4	func	1	Using where
2	DEPENDENT SUBQUERY	mi	ALL	NULL	NULL	NULL	NULL	14627337	Using where; Using join buffer
3	DEPENDENT UNION	mix	ALL	NULL	NULL	NULL	NULL	1357151	Using where
3	DEPENDENT UNION	t	eq_ref	PRIMARY	PRIMARY	4	func	1	Using where
NULL	UNION RESULT	<union2,3>	ALL	NULL	NULL	NULL	NULL	NULL	

Como vemos en la salida de la sentencia EXPLAIN, las filas correspondientes a la tercera sentencia SELECT aparecen del tipo DEPENDENT UNION.

El tipo DERIVED hace referencia a una subconsulta que aparece en la cláusula FROM. Para ilustrarlo, consideremos que deseamos restringir la información a películas españolas de 2013. En tal caso podemos aislar este resultado en una subconsulta:

```
SELECT *
FROM title t JOIN movie_info mi ON (t.id=mi.movie_id)
WHERE kind_id = 1 AND production_year = 2013
AND mi.info_type_id = 8 AND mi.info = 'Spain'
```

Y ahora podemos ya trabajar con esta tabla para consultar por ejemplo los títulos alternativos de estas películas (si los tienen):

```
EXPLAIN SELECT sp13t.title, aka.title
FROM (SELECT t.*
    FROM title t JOIN movie_info mi ON (t.id=mi.movie_id)
    WHERE kind_id = 1 AND production_year = 2013
    AND mi.info_type_id = 8 AND mi.info = 'Spain') sp13t
    LEFT JOIN aka_title aka ON (aka.movie_id = sp13t.id);
```

Como podemos ver en la salida, el SELECT identificado como 2 tiene dos filas catalogadas del tipo DERIVED, ya que se trata de tablas que aparecen en una subconsulta de la cláusula FROM:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	358	
1	PRIMARY	aka	ALL	NULL	NULL	NULL	NULL	358335	
2	DERIVED	mi	ALL	NULL	NULL	NULL	NULL	14627337	Using where
2	DERIVED	t	eq_ref	PRIMARY	PRIMARY	4	imdb.mi.movie_id	1	Using where

Finalmente el tipo UNCACHEABLE SUBQUERY referencia una subconsulta que debe ser evaluada para cada fila de la consulta más externa, por lo que resultan normalmente lentas. Son subconsultas que usan por ejemplo variables de sesión que pueden cambiar con frecuencia y por tanto deben ser reevaluadas cada vez que se requiere alguna fila de la salida generada por la subconsulta.

EJERCICIO 2

Muestra la salida de la sentencia EXPLAIN para una colección de consultas semánticamente relevantes y diferentes de las expresadas con anterioridad, en las que se aparezcan todos los tipos de SELECT introducidos anteriormente, excepto el tipo UNCACHEABLE SUBQUERY.

4. EL CAMPO TABLE

Este campo presenta el nombre de la tabla a la cual se refiere la fila en cuestión. Aparte del nombre de la tabla base, este campo puede también presentar dos posibles valores:

- <unionM, N>. En este caso se refiere la tabla que almacena la unión de las filas con identificadores M y N correspondientes a la salida de EXPLAIN.
- <derivedN>. La fila se refiere al resultado de la tabla derivada para la(s) fila(s) con identificador N del resultado de EXPLAIN.

5. EL CAMPO TYPE

La columna Type de la salida de EXPLAIN es uno de los más informativos sobre el rendimiento de la consulta. Este campo describe la estrategia de acceso que MySql elige para responder a la consulta. La tabla siguiente presenta todos los posibles valores (en orden de peor a mejor rendimiento) que pueden aparecer en este campo y un pequeño resumen de su significado.

Valores posibles de la columna Type en la salida de EXPLAIN	
Valor campo TYPE	*Significado*
All	Full Table Scan. Realiza una búsqueda secuencial por la tabla entera.
Index	Full Index Scan. Realiza una búsqueda por el conjunto secuencia del B-tree.
Range	Accede a una o varias rodajas de índice en predicados con rango.
Index_Subquery/ Unique_Subquery	Accede a través del índice a una tabla que aparece en una subconsulta con el predicado IN. Si el índice es único muestra *Unique*, si no, muestra *Index*.
Index_merge	Accede a la tabla a través de varios índices.
Fulltext	Se accede a través de un índice fulltext.
Ref	Accede a través de un índice no único para coleccionar un conjunto de tuplas que tienen el valor indicado.
Eq_ref	Aparece en joins cuando se accede a través de un índice único.
Const	La tabla posee una única fila con un valor solicitado y se lee al comienzo de la consulta y una sola vez.
System	La tabla tiene una única fila. Es un caso especial del tipo const.

Con objeto de apreciar la importancia de este campo en el análisis del plan de ejecución de la consulta, considera la necesidad de listar filas de la tabla *name* cuyo apellido tenga un código fonético A1452:

```
EXPLAIN
SELECT * FROM name WHERE name_pcode_cf = 'A1452';
```

cuya salida es:

```
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|  1 | SIMPLE      | name  | ALL  | NULL          | NULL | NULL    | NULL | 4109319 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
```

Dado que no tenemos ningún índice sobre el atributo *name_pcod_cf*, la única posibilidad es realizar un full table scan (ALL) sobre la tabla, y para cada fila comprobar si cumple o no el predicado.

Si creamos ahora un índice *pcod_cf_ndx* sobre la columna *name_pcod_cf*, ¿qué tipo de acceso utilizará el procesador de consultas para ejecutar la consulta anterior? Teniendo en cuenta que se trata de un índice no único y coleccionamos un conjunto de tuplas con un valor indicado el tipo será *ref*. Comprobemos:

```
CREATE INDEX pcod_cf_ndx ON name (name_pcod_cf);

EXPLAIN
SELECT * FROM name WHERE name_pcode_cf = 'A1452';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	name	ref	name_pcode_cf	name_pcode_cf	8	const	341	Using where

Supongamos ahora que deseamos listar todos los códigos fonéticos presentes en la tabla para los apellidos. Considerando que ahora tenemos un índice por dicho valor, haciendo un recorrido completo por el conjunto secuencia del árbol B+ correspondiente daríamos respuesta a dicha consulta, por tanto el tipo de acceso en la respuesta a dicha consulta debería ser index. Veamos:

```
EXPLAIN
SELECT name_pcode_cf FROM name;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	name	index	NULL	name_pcode_cf	8	NULL	4109319	Using index

EJERCICIO 3

¿Qué tipo de salida se muestra si en lugar de listar exclusivamente el código fonético se presenta también el nombre? Explica el por qué de la salida obtenida.

Si ahora queremos listar las filas de la tabla *name* correspondientes a un rango delimitado de códigos, el tipo de acceso sería *range:*

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	name	range	name_pcode_cf	name_pcode_cf	8	NULL	170486	Using where

Ejercicio 4

Amplía el rango que se pretende seleccionar a ['A14','F04'] y muestra la salida de explain, explicando por qué cambia el tipo de acceso.

Supongamos a continuación que deseamos mostrar los títulos de las películas dirigidas por el director con id=29911. Lo correcto sería escribir la consulta como un join, pero podríamos hacerlo a través de la subconsulta

```
EXPLAIN

SELECT title
FROM title
WHERE id IN (
    SELECT movie_id
    FROM cast_info
    WHERE person_id = 29911
    AND role_id = 8);
```

Si construimos un índice *mrp_ndx* en la tabla *cast_info* sobre (*movie_id*, *role_id, person_id*), entonces el tipo que utilizará el procesador para acceder a *cast_info* será *index_subquery*.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	title	ALL	NULL	NULL	NULL	NULL	2477301	Using where
2	DEPENDENT SUBQUERY	cast_info	index_subquery	mrp_ndx	mrp_ndx	12	func,const,const	1	Using index; Using where

En consultas en las que es posible utilizar más de un índice para acelerar al acceso a los datos solicitados, el optimizador puede mezclar el uso de dichos índices para conseguir mayor eficiencia. En este caso la salida de EXPLAIN muestra un tipo *index_merge*. Esto sucede por ejemplo en consultas en que aparezca un operador OR sobre columnas indexadas:

```
EXPLAIN
SELECT *
FROM name
WHERE id = 45678 OR name_pcode_cf='a432'\G
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: name
         type: index_merge
possible_keys: PRIMARY,name_pcode_cf
          key: PRIMARY,name_pcode_cf
      key_len: 4,8
          ref: NULL
         rows: 67
        Extra: Using union(PRIMARY,name_pcode_cf); Using where
```

En la ejecución de joins, cuando el acceso se realiza a través de un índice único a la tabla del bucle más interno, entonces el tipo de acceso que se utiliza es *eq_ref*. Por ejemplo, supongamos que queremos listar el título de la película correspondiente a la fila con identificador id de 1 a 10 en la tabla *cast_info*:

```
EXPLAIN
SELECT title
FROM title JOIN cast_info ci ON (title.id = ci.movie_id)
WHERE ci.id BETWEEN 1 AND 10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	ci	range	PRIMARY,midrid_ndx,mrp_ndx,mpr_ndx	PRIMARY	4	NULL	11	Using where
1	SIMPLE	title	eq_ref	PRIMARY,idpk_ndx	PRIMARY	4	imdb.ci.movie_id	1	

En el caso particular en que la rodaja de índice contenga una única entrada, entonces el tipo de acceso que se muestra es *const*. Se trata de un acceso muy rápido y eficiente en que la entrada sobre el índice se lee una única vez al comienzo de la consulta y el valor leído se considera constante durante la ejecución de la misma. Esto sucede si modificamos la consulta anterior y en lugar de acceder a las 10 primeras entradas del índice primario sobre *cast_info*, accedemos a una única entrada, por ejemplo:

```
EXPLAIN
SELECT title
FROM title JOIN cast_info ci ON (title.id = ci.movie_id)
WHERE ci.id = 123456;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	ci	const	PRIMARY,midrid_ndx,mrp_ndx,mpr_ndx	PRIMARY	4	const	1	
1	SIMPLE	title	const	PRIMARY,idpk_ndx	PRIMARY	4	const	1	

El tipo de acceso *system* es un tipo especial reservado para tablas que tienen una única fila.

6. KEY Y ROWS

Además del tipo de acceso, las columnas *key* y *row* son interesantemente informativas sobre el costo que puede conllevar nuestra consulta. En el capítulo anterior estudiamos la vía de construcción de índices para acelerar las consultas en las que se involucran joins. Ahora, con la introducción de la herramienta EXPLAIN disponemos de un recurso valioso que nos permite conocer el plan de ejecución que va a desplegar el optimizador de consultas. En tal sentido, los campos *key* y *row* son extremadamente informativos para hacernos una idea del coste que puede suponer la consulta.

El campo *key* nos indica qué índice va a utilizar el procesador sobre la tabla en cuestión para llevar a cabo el acceso a la misma, mientras que la columna *row* nos informa de las filas a las que tendrá que acceder sobre dicha tabla para responder a la consulta.

Queremos listar los identificadores de películas españolas de 2012 con las valoraciones (cod 101) que les han otorgado los usuarios. Los únicos índices disponibles para realizar esta consulta son los índices primarios definidos sobre las correspondientes tablas.

EJERCICIO 5

Muestra el resultado de EXPLAIN para esta consulta. Si dispones de más índices ya creados sobre las tablas, obliga a que sólo se utilicen los índices primarios.

□

La salida explain para esta consulta es:

```
+----+-------------+-------+--------+---------------+---------+---------+-------------------+----------+------------------------------+
| id | select_type | table | type   | possible_keys | key     | key_len | ref               | rows     | Extra                        |
+----+-------------+-------+--------+---------------+---------+---------+-------------------+----------+------------------------------+
|  1 | SIMPLE      | mix   | ALL    | NULL          | NULL    | NULL    | NULL              |  1357151 | Using where                  |
|  1 | SIMPLE      | t     | eq_ref | PRIMARY       | PRIMARY | 4       | imdb.mix.movie_id |        1 | Using where                  |
|  1 | SIMPLE      | mi    | ALL    | NULL          | NULL    | NULL    | NULL              | 14627337 | Using where; Using join buffer |
+----+-------------+-------+--------+---------------+---------+---------+-------------------+----------+------------------------------+
```

Según esta salida, el join comenzará por la tabla *movie_info_idx* sobre la cual realizará un table full scan sin utilizar ningún índice (observa que la columna key es NULL, lo que significa que no utiliza ningún índice para acceder). El número de filas al que accederá en esta tabla será 1,3 millones. Para cada fila de esta tabla accederá a través del índice primario de la tabla *title* para localizar exactamente una fila (*rows* = 1, y además el tipo de acceso es *eq_ref*) y una vez concatenados los valores, realizará un nuevo table full scan sobre la tabla *movie_info* para encontrar las filas que cumplen los predicados de la consulta y concatenarlos con los valores anteriores para llevarlo a la salida. Si prestamos atención a los valores de la columna *rows* nos haremos una idea del coste de esta consulta. En el bucle externo del nested loop join tenemos 1,3 millones en el bucle anidado sólo se realiza un acceso y en el bucle interior se realizan 14,6 millones de lecturas, por tanto el número total de lecturas será de $1{,}3 \times 14{,}6 \times 10^{12}$, lo cual resulta inaceptablemente ineficiente.

En vista de los valores obtenidos a partir de la sentencia EXPLAIN, tiene sentido actuar con la creación de índices que nos permitan mejorar la eficiencia de esta consulta. Para ello aplicamos los mecanismos estudiados en prácticas anteriores y analizamos el plan de ejecución obtenido por EXPLAIN. En este caso, tenemos varias opciones de mejora para el acceso a las diferentes tablas. Veamos algunas de ellas y comprobemos cual es la que presenta mejores resultados.

Podemos comenzar el join por la tabla *title* utilizando un índice sobre (*production_year, kind_id*), de modo que obtengamos una rodaja suficientemente pequeña. A continuación, necesitamos obtener el valor *title.id* para localizar las filas en *movie_info* cuyo valor de *movie_id* coincida con *title.id*. Teniendo esto en cuenta podemos ahorrarnos el acceso a la tabla *title* si incorporamos en el índice el valor *title.id*, así pues construimos sobre *title* el índice (*production_year, kind_id, id*). Para evitar un full scan en *movie_info* creamos aquí un índice (*movie_id*) y finalmente realizamos el join con la tabla *movie_info_idx*, sobre la cual debemos también localizar el valor *movie_id* para obtener finalmente la información *movie_info_idx.info*, por tanto obtendremos ventaja si construimos un índice (*movie_id*) sobre la tabla *movie_info_idx*. Ejecutamos la operaciones y valoramos la salida de EXPLAIN.

```
CREATE INDEX pry_kid_id_ndx ON
    title (production_year, kind_id, id)
CREATE INDEX mid_ndx ON movie_info (movie_id)
CREATE INDEX mid_inf_ndx ON movie_info_idx (movie_id)

EXPLAIN
SELECT mi.id,mix.info
FROM title t
    STRAIGHT_JOIN movie_info mi ON (t.id = mi.movie_id)
    STRAIGHT_JOIN movie_info_idx mix
                ON (mi.movie_id = mix.movie_id)
WHERE t.production_year = 2012 AND t.kind_id = 1
AND mi.info_type_id = 8 AND mi.info ='Spain'
AND mix.info_type_id = 101;
```

```
+----+-------------+-------+------+---------------------------------------+----------------+---------+--------------------+-------+--------------------------+
| id | select_type | table | type | possible_keys                         | key            | key_len | ref                | rows  | Extra                    |
+----+-------------+-------+------+---------------------------------------+----------------+---------+--------------------+-------+--------------------------+
|  1 | SIMPLE      | t     | ref  | PRIMARY,py_kid_id_ndx,idpk_ndx,pik_ndx | py_kid_id_ndx | 9       | const,const        | 39120 | Using where; Using index |
|  1 | SIMPLE      | mi    | ref  | movie_id_ndx                          | movie_id_ndx   | 4       | imdb.t.id          |     6 | Using where              |
|  1 | SIMPLE      | mix   | ref  | mid_ndx                               | mid_ndx        | 4       | imdb.mi.movie_id   |     3 | Using where              |
+----+-------------+-------+------+---------------------------------------+----------------+---------+--------------------+-------+--------------------------+
```

Observando la salida anterior podemos extraer algunas conclusiones importantes. En primer lugar los accesos a las tres tablas son de tipo *ref*, lo que significa que a todas ellas se accede a través de un índice no único. Los índices utilizados aparecen en la columna *key*. Por su parte la columna *rows* nos indica que ahora el número de accesos ha caído hasta un total de $39.120 \times 6 \times 3$, unas 700.000 filas en total. Como puede verse, en la consulta hemos obligado a hacer el join en un orden concreto, a saber, en el bucle más externo aparece la tabla *title*, al cual se anida el acceso a la tabla *movie_info* y finalmente en el bucle más interior aparece la tabla *movie_info_idx*.

¿Podemos mejorar aún más el rendimiento de nuestra consulta? Veamos qué sucede si en lugar de comenzar el acceso por *title*, lo hacemos por *movie_info*. En este caso precisaríamos un índice por (*info_type,info*), lo cual nos generaría una rodaja pequeña. Sin embargo la columna *info* es de tipo Text y MySql no nos permite incorporarla directamente al índice, sino que tendríamos que considerar la incorporación de un prefijo. Veamos qué cantidad de caracteres de *info* nos permite diferenciar el país de la película:

```
SELECT 100*COUNT(DISTINCT SUBSTR(info,1,25))/COUNT(DISTINCT
info) as 'capacidad de filtrado 25'
FROM movie_info
WHERE info_type_id = 8;
```

```
+-------------------------+
| capacidad de filtrado 25 |
+-------------------------+
|                100.0000 |
+-------------------------+
```

Como vemos 25 caracteres son suficientes para filtrarnos todos los países posibles, por lo que construimos un índice (*info_type, info(25)*) sobre *movie_info*. Ya que necesitamos el campo *movie_id* para acceder a *title*, podemos incorporarlo al índice evitando así un acceso a la tabla base, por tanto nuestro índice final sobre *movie_info* será (*info_type, info(25), movie_id*). Con el valor de *movie_id* nos vamos a la tabla *title* para acceder por *id*, el cual al ser clave primaria nos evita la creación de un nuevo índice. Finalmente debemos acceder a *movie_info_idx* para obtener los valores de *movie_id* coincidentes y acceder al campo *info*. Este acceso puede hacerse por medio del índice (*movie_id*) ya existente sobre *movie_info_idx*. Observemos el plan:

```
CREATE INDEX itid_inf25_mid_ndx ON movie_info (info_type_id,in-
fo(25),movie_id);

EXPLAIN
SELECT mi.id,mix.info
FROM movie_info mi USE INDEX (itid_inf25_mid_ndx)
    STRAIGHT_JOIN title t ON (t.id = mi.movie_id)
    STRAIGHT_JOIN movie_info_idx mix USE INDEX (mid_ndx)
            ON (mi.movie_id = mix.movie_id)
WHERE t.production_year = 2012 AND t.kind_id = 1
AND mi.info_type_id = 8 AND mi.info ='Spain'
AND mix.info_type_id = 101;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	mi	ref	itid_inf25_mid_ndx	itid_inf25_mid_ndx	31	const,const	7920	Using where
1	SIMPLE	t	eq_ref	PRIMARY,py_kid_id_ndx,idpk_ndx,pik_ndx	PRIMARY	4	imdb.mi.movie_id	1	Using where
1	SIMPLE	mix	ref	mid_ndx	mid_ndx	4	imdb.t.id	3	Using where

Como vemos ahora hemos ganado rendimiento ya que observando la columna *rows* vemos que el número de filas totales a acceder es ahora de $7.920 \times 1 \times 3$, algo menos de 24.000 filas, que comparadas con las 700.000 anteriores suponen un ahorro considerable.

EJERCICIO 6

¿Es posible mejorar el rendimiento de esta consulta? Razona tu respuesta analizando la salida de EXPLAIN de las diferentes consultas. Utiliza la opción STRAIGHT_JOIN y USE INDEX para llevar a cabo este análisis.

□

7. POSSIBLE_KEYS, KEY_LEN Y REF

El campo *possible_keys* muestra los índices existentes que están a disposición del optimizador para ejecutar la consulta. Un valor NULL para este campo significa que el optimizador no puede encontrar índices relevantes para responder la consulta, lo cual no indica ausencia total de índices sobre la tabla. Los campos *key* y *posible_keys* son por tanto los primeros que hay que analizar para iniciar la estrategia de indexación.

El campo *key_len* muestra el número de bytes que MySql utilizará para el índice que aparece referenciado en el campo *key*. Esta columna permite concluir si el índice entero o sólo parte de él se va a utilizar para responder a la consulta.

Por ejemplo, en el apartado anterior construimos el índice *itid_inf25_mid_ndx* compuesto por las columnas (*info_type_id,info(25),movie_id*) sobre *movie_info_idx*. Consideremos ahora una consulta que pretende listar los identificadores de obras con una valoración por encima de 9.0:

```
EXPLAIN
SELECT movie_id
FROM movie_info_idx
WHERE info_type_id = 101
AND info > "9";
```

```
+----+-------------+----------------+-------+--------------------+--------------------+---------+------+------+-------------+
| id | select_type | table          | type  | possible_keys      | key                | key_len | ref  | rows | Extra       |
+----+-------------+----------------+-------+--------------------+--------------------+---------+------+------+-------------+
|  1 | SIMPLE      | movie_info_idx | range | itid_inf25_mid_ndx | itid_inf25_mid_ndx | 31      | NULL |  841 | Using where |
+----+-------------+----------------+-------+--------------------+--------------------+---------+------+------+-------------+
```

Como podemos ver el valor de *key_len* es de 31, lo que indica que ha utilizado 31 bytes de la entrada de índice. La entrada de índice está compuesto por dos enteros (4 bytes) y una cadena de 25 caracteres, cuya suma es 33. Entonces ¿de dónde sale este valor? Hay que indicar

que, por una parte para cada tipo que pueda ser NULL hay que añadir un byte más, y por otra que el tipo TEXT necesita 2 bytes adicionales para almacenar su longitud, por tanto la columna *info* ocupa 27 bytes. Teniendo en cuenta que ninguno de los campos referenciados en el índice pueden ser nulos, el total de bytes de la entrada serán 4+4+27 = 35 bytes. Así pues el optimizador está utilizando sólo dos campos, a saber *info_type_id* (4 bytes) e *info* (27 bytes) que dan los 31 de la salida.

La columna *ref* en la salida de explain nos muestra sobre qué columna obtiene MySql el valor para comparar cuando el tipo de acceso a la tabla es *ref* o *eq_ref*. Considera la siguiente consulta con su correspondiente salida:

```
EXPLAIN SELECT straight_join ci.movie_id
FROM title t JOIN cast_info ci
    ON (t.id=ci.movie_id AND ci.person_id = 494761
                    AND ci.role_id=1)
    JOIN movie_info mi ON (mi.movie_id = ci.movie_id)
WHERE mi.info_type_id = 3 AND mi.info = "thriller"
AND t.kind_id=1;
```

Esta consulta presenta identificadores de películas en las que ha intervenido como actor Harrison Ford (id = 494761). La salida del comando es:

```
+-------+-------+--------------------------------------+--------------+---------+----------------------------+
| table | type  | possible_keys                        | key          | key_len | ref                        |
+-------+-------+--------------------------------------+--------------+---------+----------------------------+
| t     | index | PRIMARY,idpk_ndx                     | py_kid_id_ndx | 13     | NULL                       |
| ci    | ref   | midrid_ndx,mrp_ndx,mpr_ndx,pidrid_ndx | mrp_ndx      | 12      | imdb.t.id,const,const      |
| mi    | ref   | movie_id_ndx,mii_ndx,itid_inf25_mid_ndx | mii_ndx    | 40      | imdb.ci.movie_id,const,const |
+-------+-------+--------------------------------------+--------------+---------+----------------------------+
```

En la primera fila, correspondiente al acceso a la tabla *title*, dado que no existe ningún valor que pueda usarse como referencia para buscar en el índice *py_kid_id_ndx*, el valor que nos muestra la columna *ref* es NULL. Sin embargo la tabla *cast_info* se accede a través del índice *mrp_ndx* (*movie_id, role_id, person_id*) y en este caso se utilizan dos valores constantes, a saber *ci.person_id*= 494761 y *ci.role_id* = 1 y el valor dado por el atributo *id* de la tabla *title* (*imdb.t.id*). Similarmente sucede en el caso de la tabla *movie_info*, el cual utiliza para su acceso el índice *mii_ndx* (*movie_id, info_type_id, info*(*30*)) y utiliza para buscar en el índice dos valores constantes que son *mi.info_type_id* = 3 y *mi.info* = "thriller" y el valor *movie_id* obtenido a partir de la tabla *cast_info* (*imdb.ci.movie_id*).

8. EL CAMPO EXTRA

Este campo muestra información adicional sobre el modo en que MySql ejecuta la consulta, aunque la diversidad de notas que pueden aparecer en este campo es amplia, las observaciones más interesantes que merecen conocerse son las siguientes:

- *Using Index*. Se usará un *covering index*, es decir un índice del que se obtienen todos los datos de la consulta, sin necesidad de acceder a la tabla base. Se corresponde a la tercera estrella en el método *three star indexing* que estudiamos en prácticas anteriores.
- *Using where*. Nos indica que una cláusula WHERE se utiliza para filtrar tuplas que bien se envían al cliente o bien se trasladan a la siguiente tabla (en una operación JOIN).
- *Using temporary*. En algún punto de la ejecución se precisa construir una tabla temporal. Para mantenerla en RAM es preciso asegurar ciertos parámetros del sistema, tales como *tmp_table-size* y *max_heap_table_size*. Cuando se muestra esta información en la salida de la sentencia EXPLAIN, seguramente hay posibilidad de mejorar el rendimiento de la consulta, tratando de evitar la necesidad de construir estas tablas temporales, las cuales consumen tiempo y recursos del sistema.
- *Using filesort*. Se precisa aplicar un proceso de ordenación adicional para retornar el resultado al cliente. Este proceso se lleva a cabo normalmente en memoria, pero si el fichero es grande, posiblemente suponga accesos a disco para llevar a cabo la ordenación. Igual que en la opción anterior, cuando nos aparece esta información, existe normalmente la posibilidad de mejorar el rendimiento de la consulta.

Veamos algunos ejemplos donde las anteriores indicaciones aparecen en la salida de la sentencia EXPLAIN y cómo pueden mejorarse.

Deseamos listar identificadores de series de 2012. Planteamos la consulta:

```
EXPLAIN SELECT id
FROM title t
WHERE kind_id = 2
AND production_year = 2012;
```

```
+----+-------------+-------+------+----------------------+--------------+---------+-------------+------+--------------------------+
| id | select_type | table | type | possible_keys        | key          | key_len | ref         | rows | Extra                    |
+----+-------------+-------+------+----------------------+--------------+---------+-------------+------+--------------------------+
|  1 | SIMPLE      | t     | ref  | py_kid_id_ndx,pik_ndx | py_kid_id_ndx | 9       | const,const | 5394 | Using where; Using index |
+----+-------------+-------+------+----------------------+--------------+---------+-------------+------+--------------------------+
```

La salida de EXPLAIN indica que *py_kid_id_ndx* (*production_year, kind_id, id*) actúa como covering index (*using index*), por lo que nos ahorramos el acceso a la tabla base. Además utiliza el predicado de la cláusula WHERE para filtrar las entradas del índice, construyendo una rodaja fina.

Pero supongamos ahora que deseamos mostrar identificadores y año de series de 2013 y 2013 obtenidas en orden de id.

```
EXPLAIN SELECT id, production_year
FROM title t
WHERE kind_id = 2
AND production_year BETWEEN 2012 AND 2013
ORDER BY id;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	range	py_kid_id_ndx,pik_ndx	py_kid_id_ndx	9	NULL	143796	Using where; Using index; Using filesort

Ahora, dado que la columna *id* aparece tras *production_year* en el índice, sobre el cual se realiza un predicado por rango, para ordenar la salida por *id* es preciso llevar a cabo un proceso de ordenación extra (Using filesort). Si, en lugar de ello, hubiéramos pedido la salida ordenada por año e id ¿cual habría sido el resultado?

Si queremos listar identificadores de obras que tienen títulos alternativos, la consulta tendría la siguiente salida:

```
EXPLAIN SELECT DISTINCT movie_id
FROM aka_title;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	aka_title	ALL	NULL	NULL	NULL	NULL	358335	Using temporary

Como vemos, para poder llevar a cabo una eliminación de duplicados, MySql construye una tabla temporal con los valores diferentes que va encontrando de los correspondiente identificadores. ¿Podemos evitar la utilización de una tabla temporal? La respuesta es sí, para ello construimos un índice sobre *movie_id* y repetimos la sentencia, ahora la salida es:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	aka_title	range	NULL	movie_id	4	NULL	179168	Using index for group-by

Nos aparece en el campo Extra una información adicional que no conocíamos, la cual nos indica que utiliza un tipo de acceso conocido como *"loose index scan"*, el cual recorre el índice para recuperar valores distintos de clave, evitando el acceso a los valores repetidos de clave. Este tipo de acceso, cuando es posible, se utiliza en la operación de agrupamiento (por eso aparece el término group-by en el campo extra) y aporta muy buen rendimiento fundamentalmente cuando el índice tiene muchas entradas iguales de clave.

Ejercicio 7

Ahora que ya dispones de toda la información sobre EXPLAIN, vuelve sobre el ejercicio 6 y responde, ¿Podríamos ganar ventaja si creamos un índice (*movie_id, info(n)*) sobre la tabla *movie_info_idx* para evitarnos accesos a la tabla base? Razona tu respuesta analizando la salida.

LABORATORIO 7.
PRACTICANDO LA OPTIMIZACIÓN DE CONSULTAS EN IMDB

Objetivos. Tras cubrir esta sesión práctica, el alumno/a es capaz de aplicar los métodos de optimización para mejorar el rendimiento de consultas sobre una base de datos. Además de aplicar la metodología introducida a lo largo del curso, será capaz de utilizar las herramientas que le proporciona el SGBD para analizar el plan de ejecución de la misma y modificarlo convenientemente para conseguir el objetivo de mejora del rendimiento.

1. INTRODUCCIÓN

Esta práctica servirá de experimentación de los mecanismos introducidos en las dos prácticas anteriores para conseguir un rendimiento óptimo de las diferentes consultas que plantearemos sobre la base de datos IMDB. Para ello trabajaremos con la máquina virtual *imdb_no_index*, la cual dispone de la base de datos *imdb* en el que cada tabla incluye un único índice sobre su clave primaria.

Como en sesiones anteriores, desactivamos el buffer de consultas de mysql.

En esta práctica iremos planteando una serie de consultas que el alumno debe resolver. Se le aportará en número de filas resultantes con objeto de comprobar que la consulta está correctamente planteada. Para todas las consultas, se obtendrá la salida del comando EXPLAIN, sobre la cual se escribirá un pequeño informe describiendo qué aspectos del plan van a intentar mejorarse, tanto con la introducción de nuevos índices, como con la reescritura, si fuera necesario, de la propia consulta. Con cada conjunto de cambios que se efectúen, se obtendrá la salida de EXPLAIN, se comentarán las mejoras conseguidas y se repetirá el proceso hasta obtener un plan que entendamos es suficientemente eficiente. Finalmente se comparan

los tiempos de ejecución antes y después de nuestra intervención. Algunas de las consultas que plantearemos ya han sido tratadas con anterioridad en las sesiones correspondientes al Laboratorio 03.

2. INFORMES DE RENDIMIENTO

Para ilustrar el modo en que se solicita realizar los diferentes ejercicios que plantearemos, vamos a desarrollar a modo de ejemplo la respuesta a una consulta.

Lista nombre de actores, fecha de nacimiento y nombre del personaje que interpretan en la película "Pulp Fiction" (si el personaje que interpretan no está almacenado, muestra un valor NULL). Esta consulta ya se trató en una práctica anterior y una posible expresión en SQL de la misma es:

```
SET @pulpid = 2175869;
SELECT  n.id, n.name, ch.name AS 'character name',
        pi.info AS 'birthdate'
FROM name n JOIN cast_info ci ON (ci.person_id = n.id)
    JOIN char_name ch ON (ch.id = ci.person_role_id)
    LEFT JOIN person_info pi
        ON (pi.person_id = n.id AND pi.info_type_id = 21)
WHERE ci.movie_id = @pulpid
ORDER BY ci.nr_order;
```

Vamos a analizar la salida del comando EXPLAIN para esta consulta sin utilizar ningún índice más que los índices de clave primaria definidos. Para ello reescribimos la consulta (por si acaso hemos construido con anterioridad algún índice sobre las tablas) y obtenemos el resultado:

```
EXPLAIN SELECT n.id, n.name, ch.name AS 'character name',
        pi.info AS 'birthdate'
FROM name n USE INDEX (PRIMARY)
    JOIN cast_info ci USE INDEX (PRIMARY)
        ON (ci.person_id = n.id)
    JOIN char_name ch USE INDEX (PRIMARY)
        ON (ch.id = ci.person_role_id)
    LEFT JOIN person_info pi USE INDEX (PRIMARY)
        ON (pi.person_id = n.id AND pi.info_type_id = 21)
WHERE ci.movie_id = @pulpid
ORDER BY ci.nr_order;
```

```
+----+-------------+-------+--------+---------------+---------+---------+-----------------------+----------+----------------------------------------------+
| id | select_type | table | type   | possible_keys | key     | key_len | ref                   | rows     | Extra                                        |
+----+-------------+-------+--------+---------------+---------+---------+-----------------------+----------+----------------------------------------------+
|  1 | SIMPLE      | ci    | ALL    | NULL          | NULL    | NULL    | NULL                  | 35645119 | Using where; Using temporary; Using filesort |
|  1 | SIMPLE      | n     | eq_ref | PRIMARY       | PRIMARY | 4       | imdb.ci.person_id     |        1 |                                              |
|  1 | SIMPLE      | pi    | ALL    | NULL          | NULL    | NULL    | NULL                  |  2938437 |                                              |
|  1 | SIMPLE      | ch    | eq_ref | PRIMARY       | PRIMARY | 4       | imdb.ci.person_role_id |        1 |                                              |
+----+-------------+-------+--------+---------------+---------+---------+-----------------------+----------+----------------------------------------------+
```

La información que se obtiene de esta salida nos habla de una consulta extremadamente lenta. Primeramente la tabla más voluminosa *cast_info* se accede a través de un table full scan (ALL) y para cada fila, una vez unida a la correspondiente fila de *name* se realiza un table full scan sobre la tabla char_name, por lo que se precisan un total de $35 \times 10^6 \times 3 \times 10^6$ o aproximadamente 105 billones de accesos, lo cual supone un coste inaceptable. Además el campo Extra nos indica que MySql precisa crear una tabla temporal y realizar una operación de ordenación adicional para finalmente responder a la consulta.

Es claro que un sitio para comenzar a mejorar el rendimiento de esta consulta es evitar el full scan de *cast_info (ci)*, ya que de esta tabla sólo estamos interesados en aquellas filas cuyo *movie_id* corresponda al de la película "Pulp Fiction" (2175869), así pues podemos considerar la necesidad de un índice conteniendo (*movie_id*) sobre *ci*. Dado que también deseamos mostrar las tuplas en orden del campo *nr_order*, podemos incorporar esta columna al índice anterior (*movie_id, nr_order*). Una vez restringido el acceso a la rodaja correspondiente del índice, necesitamos acceder a la tabla *name* a partir del correspondiente valor de *ci.person_id*. Si incorporamos *person_id* en nuestro índice evitaremos el acceso a la tabla base *ci*. Exactamente igual sucede con el campo *person_role_id*, el cual se utiliza para hacer el join con la tabla char_name. Podríamos plantearnos incorporar también este campo al índice. Sin embargo, si lo hacemos así, el índice va a ocupar mucho espacio en disco de modo que su acceso puede verse seriamente ralentizado y jugar en nuestra contra. Así pues, ya que finalmente necesitaremos acceder a la tabla base *ci*, podemos pensar en crear definitivamente un índice mov_order_ndx (*movie_id, nr_order*) sobre la tabla *ci*:

```
CREATE INDEX mov_order_ndx ON cast_info (movie_id, nr_order)
```

En la salida de explain vemos que el join con la tabla *person_info* conlleva también un full scan. En este caso buscamos filas cuyo valor de *person_id* coincidan con *name.id* y que tengan sobre i*nfo_type_id* un valor constate (21). Dado que MySql utiliza el algoritmo de bucles anidados para efectuar el join y que la tabla *person_info* está en un bucle interno, de nada sirve construir un índice por i*nfo_type_id,* ya que el algoritmo precisa encontrar tuplas buscando sobre el valor de *person_id*. Es por ello que sería de utilidad un índice que nos facilitara la búsqueda del valor correspondiente. En este caso de nada sirve incorporar al índice el valor de *info_type_id*, ya que es completamente inevitable acceder a la tabla base para obtener el valor de info, el cual se solicita en la cláusula SELECT. Así pues un índice *persid_ndx* (*person_id)* sobre *person_info* nos ayudaría a mejorar el rendimiento de nuestra consulta:

```
CREATE INDEX persid_ndx ON person_info (person_id)
```

Veamos ahora el efecto que causa nuestra intervención en la consulta.

```
EXPLAIN SELECT n.id, n.name, ch.name AS 'character name',
        pi.info AS 'birthdate'
FROM name n
    JOIN cast_info ci ON (ci.person_id = n.id)
    JOIN char_name ch ON (ch.id = ci.person_role_id)
    LEFT JOIN person_info pi
        ON (pi.person_id = n.id AND pi.info_type_id = 21)
WHERE ci.movie_id = 2175869
ORDER BY ci.nr_order;
```

```
+----+-------------+-------+--------+--------------------------+---------------+---------+------------------------+------+-------------+
| id | select_type | table | type   | possible_keys            | key           | key_len | ref                    | rows | Extra       |
+----+-------------+-------+--------+--------------------------+---------------+---------+------------------------+------+-------------+
|  1 | SIMPLE      | ci    | ref    | pidrid_ndx,mov_order_ndx | mov_order_ndx | 4       | const                  |  171 | Using where |
|  1 | SIMPLE      | ch    | eq_ref | PRIMARY                  | PRIMARY       | 4       | imdb.ci.person_role_id |    1 |             |
|  1 | SIMPLE      | n     | eq_ref | PRIMARY                  | PRIMARY       | 4       | imdb.ci.person_id      |    1 |             |
|  1 | SIMPLE      | pi    | ref    | persid_ndx               | persid_ndx    | 4       | imdb.ci.person_id      |    5 |             |
+----+-------------+-------+--------+--------------------------+---------------+---------+------------------------+------+-------------+
```

Como podemos ver, la mejora ha sido espectacular. El optimizador utiliza nuestros índices *move_order_ndx* y *persid_ndx* para evitar los costosos full scans sobre las tablas *cast_info* y *person_info* que teníamos anteriormente. Ahora el join se reduce al acceso a $171 \times 5 = 855$ tuplas y el procesador de consultas no requiere de la creación de tablas temporales ni la utilización de procesos adicionales de ordenación que ralentizan la consulta. Los tipos de acceso son *ref* o *eq_ref*, los mejores posibles en la ejecución del join. Aunque el campo *key_len* para la tabla *cast_info* muestra la utilización de sólo una parte del índice (la correspondiente a los 4 bytes que ocupa el campo *movie_id*), en realidad, la incorporación de la columna *nr_order* nos ha evitado el que aparezca en el campo Extra la información *"using filesort"*, la cual incrementa el tiempo de ejecución de la consulta.

El speed-up que conseguimos con el análisis del plan de ejecución y la creación de los índices anteriores es considerable. Si bien la consulta sin indexar invierte casi 21 minutos, como muestra esta salida:

```
| 1562476 | Turner, Rich         | Sportscaster #2                   | NULL             |
| 2586599 | Valentino, Venessia  | Pedestrian                        | NULL             |
| 2586599 | Valentino, Venessia  | Bonnie Dimmick                    | NULL             |
| 1747168 | Arquette, Alexis     | Man #4                            | 28 July 1969     |
| 2141618 | Kaye, Linda          | Shot Woman                        | NULL             |
| 2038138 | Griffin, Kathy       | Hit-and-run Witness               | 4 November 1960  |
| 1510086 | Tarantino, Quentin   | Jimmie Dimmick                    | 27 March 1963    |
|  782985 | Keitel, Harvey       | Winston 'The Wolf' Wolfe          | 13 May 1939      |
| 2256312 | Maruyama, Karen      | Gawker #1                         | 29 May 1958      |
|  120919 | Bender, Lawrence     | Long Hair Yuppy Scum              | 17 October 1957  |
| 1426048 | Sitka, Emil          | Hold Hands You Lovebircs          | 22 December 1914 |
| 1033958 | Miller, Dick         | Monster Joe                       | 25 December 1928 |
+---------+----------------------+-----------------------------------+------------------+
58 rows in set (20 min 49.17 sec)
```

el tiempo de ejecución siguiendo el plan marcado por la salida de EXPLAIN es de 5 segundos, tal y como muestra la siguiente imagen:

```
| 1747168 | Arquette, Alexis      | Man #4                            | 28 July 1969      |
| 2141618 | Kaye, Linda           | Shot Woman                        | NULL              |
| 2038138 | Griffin, Kathy        | Hit-and-run Witness               | 4 November 1960   |
| 1510086 | Tarantino, Quentin    | Jimmie Dimmick                    | 27 March 1963     |
|  782985 | Keitel, Harvey        | Winston 'The Wolf' Wolfe          | 13 May 1939       |
| 2256312 | Maruyama, Karen       | Gawker #1                         | 29 May 1958       |
|  120919 | Bender, Lawrence      | Long Hair Yuppy Scum              | 17 October 1957   |
| 1426048 | Sitka, Emil           | Hold Hands You Lovebirds          | 22 December 1914  |
| 1033958 | Miller, Dick          | Monster Joe                       | 25 December 1928  |
+---------+---------------------+----------------------------------+------------------+
58 rows in set (5.19 sec)
```

Así pues el speed-up que conseguimos es de 240.6, una mejora espectacular que da idea de la importancia del papel del ABD en una organización.

3. EJERCICIOS DE OPTIMIZACIÓN

Realiza un informe similar al anterior para optimizar cuanto sea posible las siguientes consultas:

1. Identificadores, títulos y año de producción de películas protagonizadas conjuntamente por Spencer Tracy (1548437) y Katharine Hepburn (2073327). Obtén el listado por orden creciente de año de producción.
2. Lista nombres y año de nacimiento (código 21) de todos los actores de "Blade Runner" (id= 1693255).
3. Lista la valoración media de los capítulos de la serie "Breaking Bad" (id=175285) para cada año de los que se emitió esta serie.
4. Lista identificadores y nombre de actores/actrices que han actuado tanto en la película "Kill Bill Vol. 1" (id = 1974761) como en "Kill Bill Vol. 2" (id=1974762), obteniendo el listado ordenado por la importancia de sus papeles (lo valores nulos al final).
5. Obtén un listado de películas producidas en 2010 en las que su director/a participe también como actor/actriz en la obra. Presenta junto al nombre de la película su director-actor y su fecha de nacimiento.