



Facultad de
Ingeniería

Bases de datos

Año: 2021

Trabajo integrador final “MovieLens”

Salim Taleb, Nasim A.

Docentes: Hadad, Alejandro; Elias, Walter

Carrera: Lic. en Bioinformática

Base de datos MovieLens

Para este trabajo se importó la base de datos MovieLens, que puede ser encontrada en <https://relational.fit.cvut.cz/dataset/MovieLens>, consiste básicamente en una base de datos que almacena películas con sus respectivos directores y actores y también múltiples reseñas de usuarios. Como se puede evidenciar en los diagramas siguientes, la base de datos consiste en una tabla central llamada “movies”, la cual se relaciona con tablas intermedias a las demás tablas “actors”, “directors” y “users”. Las tablas “actors” y “directors” tiene un indicador de calidad y otros atributos pertinentes a cada uno, lo más destacable de la base de datos son las valoraciones de usuarios que serán utilizados para dar recomendaciones.

Diagrama de ER y tablas

Diagrama ER:

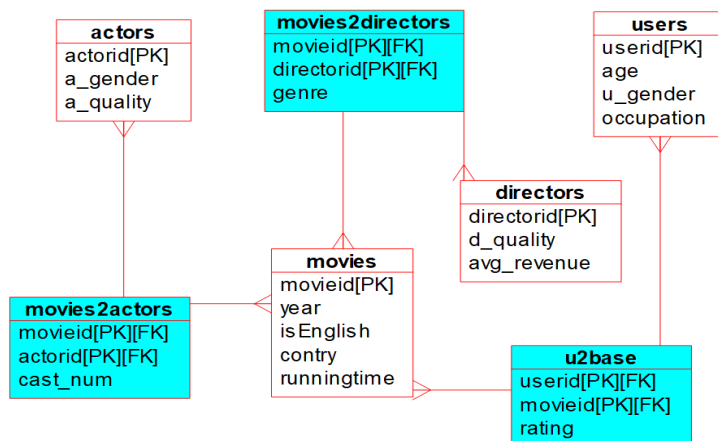
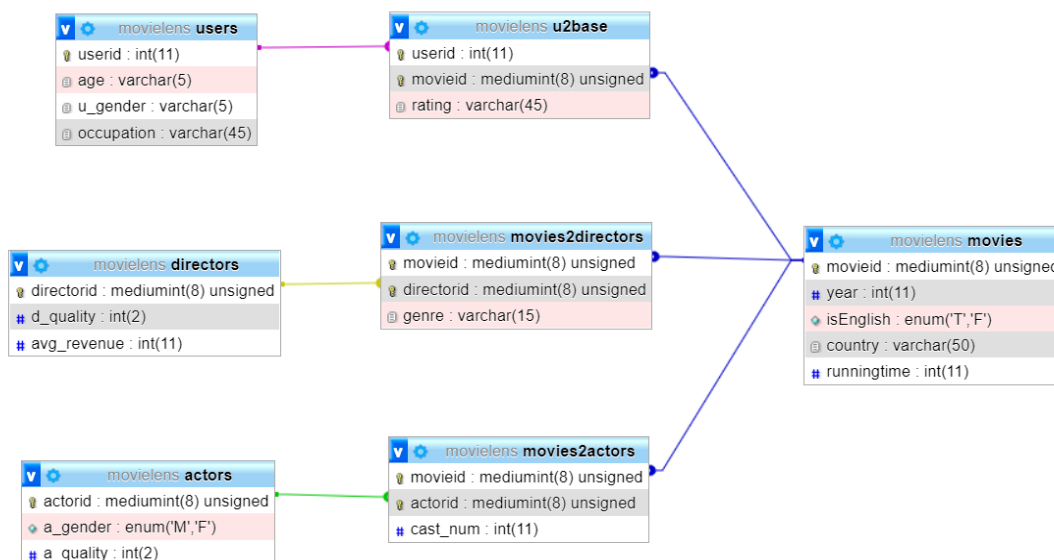


Diagrama de tablas:



Diccionario de la base de datos

actors

Modela actores que participan en películas

Columna	Tipo	Nulo	Predeterminado	Enlaces a	Comentarios
actorid (<i>Primaria</i>)	mediumint(8)	No			ID única de actor
a_gender	enum('M', 'F')	No			género del actor
a_quality	int(2)	No			(0-5) Dato que representa simbólicamente la calidad del actor

directors

Contiene datos acerca de directores que participan en películas

Columna	Tipo	Nulo	Predeterminado	Enlaces a	Comentarios
directorid (<i>Primaria</i>)	mediumint(8)	No			ID única de director
d_quality	int(2)	No			(0-5) Dato simbólico que representa la calidad del director
avg_revenue	int(11)	No			(0-4) Dato simbólico que representa el nivel de ingresos del director

movies

Contiene datos acerca de películas

Columna	Tipo	Nulo	Predeterminado	Enlaces a	Comentarios
movieid (<i>Primaria</i>)	mediumint(8)	No	0		ID única de cada película
year	int(11)	No			(1-4) Dato simbólico que representa que tan antigua es la película
isEnglish	enum('T', 'F')	No			Indica si la película está en inglés
country	varchar(50)	No			Indica de que país es la película
runningtime	int(11)	No			(0-3) Dato simbólico que expresa la duración de la película

users

Tiene datos sobre los usuarios que puntúan a las películas

Columna	Tipo	Nulo	Predeterminado	Enlaces a	Comentarios
userid (<i>Primaria</i>)	int(11)	No	0		ID única de cada usuario
age	varchar(5)	No			Edad aproximada del usuario
u_gender	varchar(5)	No			('M','F') género del usuario
occupation	varchar(45)	No			(1-5) Dato simbólico que representa la ocupación de los usuarios

movies2actors

Conecta a las películas con sus actores y viceversa

Columna	Tipo	Nulo	Predeterminado	Enlaces a	Comentarios
movieid (Primaria)	mediumint(8)	No		movies -> movieid	ID de película a vincular
actorid (Primaria)	mediumint(8)	No		actors -> actorid	ID de actor a vincular
cast_num	int(11)	No			Representa la división de reparto del actor en la película

movies2directors

Relaciona a los directores con sus películas y viceversa

Columna	Tipo	Nulo	Predeterminado	Enlaces a	Comentarios
movieid (Primaria)	mediumint(8)	No		movies -> movieid	Id de película a vincular
directorid (Primaria)	mediumint(8)	No		directors -> directorid	ID de director a vincular
genre	varchar(15)	No			Género de película

u2base

Relaciona a los usuarios con las películas que hayan puntuado

Columna	Tipo	Nulo	Predeterminado	Enlaces a	Comentarios
userid (Primaria)	int(11)	No	0	users -> userid	ID de usuario a vincular
movieid (Primaria)	mediumint(8)	No		movies -> movieid	ID de la película a vincular
rating	varchar(45)	No			(1-5) Valoración del usuario a la película

Sentencias DDL

A continuación, se presentan las sentencias necesarias para crear las tablas del esquema de la base de datos, recordando que se pueden obtener directamente exportando la base de datos original. Se utilizará como motor de base de datos a MySQL, por ser una de los motores más utilizados y difundidos gracias a su licencia de código abierto y a su gran performance para bases de datos grandes (Jesuïtes Educació, 2018; HostingPedia, 2019). Y como mecanismo de almacenamiento a InnoDB ya que la base de datos requerirá en mayor medida de sentencias del tipo SELECT que de INSERT, para lo cual es mejor utilizar InnoDB (Tellado, 2020; IONOS, 2020), debido a que es una base de datos de recomendaciones de películas a usuarios basadas en sus gustos, por lo que debe leer una gran cantidad de datos para realizar las mismas, además de que no se realizan una gran cantidad de inserciones por la naturaleza de la base de datos, no se crean datos en una gran medida.

Ejemplos de sentencias para realizar el insertado de datos de cada tabla se encuentran en el anexo, no se incluirán todas ya que algunas tablas tienen un número muy elevado de filas, si se desea se puede obtener el archivo con los insert exportando directamente desde la base de datos original.

actors

Creación de la tabla:

```
CREATE TABLE `actors` (  
  `actorid` mediumint(8) unsigned NOT NULL,  
  `a_gender` enum('M','F') NOT NULL,  
  `a_quality` int(2) NOT NULL,  
  PRIMARY KEY (`actorid`),  
  KEY `a_gender` (`a_gender`),  
  KEY `a_quality` (`a_quality`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

directors

Creación de la tabla:

```
CREATE TABLE `directors` (  
  `directorid` mediumint(8) unsigned NOT NULL,  
  `d_quality` int(2) NOT NULL,  
  `avg_revenue` int(11) NOT NULL,  
  PRIMARY KEY (`directorid`),  
  KEY `d_quality` (`d_quality`),  
  KEY `avg_revenue` (`avg_revenue`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

movies

Creación de la tabla:

```
CREATE TABLE `movies` (  
  `movieid` mediumint(8) unsigned NOT NULL DEFAULT 0,  
  `year` int(11) NOT NULL,  
  `isEnglish` enum('T','F') NOT NULL,  
  `country` varchar(50) NOT NULL,  
  `runningtime` int(11) NOT NULL,  
  PRIMARY KEY (`movieid`),  
  KEY `year` (`year`),  
  KEY `isEnglish` (`isEnglish`),  
  KEY `country` (`country`),  
  KEY `runningtime` (`runningtime`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

users

Creación de la tabla:

```
CREATE TABLE `users` (  
  `userid` int(11) NOT NULL DEFAULT 0,  
  `age` varchar(5) NOT NULL,  
  `u_gender` varchar(5) NOT NULL,  
  `occupation` varchar(45) NOT NULL,  
  PRIMARY KEY (`userid`),  
  KEY `index` (`age`,`u_gender`,`occupation`),  
  KEY `age` (`age`),  
  KEY `u_gender` (`u_gender`),  
  KEY `occupation` (`occupation`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

movies2actors

Creación de la tabla:

```
CREATE TABLE `movies2actors` (  
  `movieid` mediumint(8) unsigned NOT NULL,  
  `actorid` mediumint(8) unsigned NOT NULL,  
  `cast_num` int(11) NOT NULL,  
  PRIMARY KEY (`movieid`,`actorid`),  
  KEY `fk_m2a_mid_idx` (`movieid`),  
  KEY `fk_m2a_aid_idx` (`actorid`),  
  KEY `cast_num` (`cast_num`),  
  CONSTRAINT `fk_m2a_aid` FOREIGN KEY (`actorid`) REFERENCES `actors`  
  (`actorid`) ON DELETE CASCADE ON UPDATE CASCADE,  
  CONSTRAINT `fk_m2a_mid` FOREIGN KEY (`movieid`) REFERENCES `movies`  
  (`movieid`) ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

movies2directors

Creación de la tabla:

```
CREATE TABLE `movies2directors` (  
  `movieid` mediumint(8) unsigned NOT NULL,  
  `directorid` mediumint(8) unsigned NOT NULL,  
  `genre` varchar(15) NOT NULL,  
  PRIMARY KEY (`movieid`,`directorid`),  
  KEY `fk_m2d_mid_idx` (`movieid`),  
  KEY `fk_m2d_did_idx` (`directorid`),  
  KEY `genre` (`genre`),  
  CONSTRAINT `fk_m2d_did` FOREIGN KEY (`directorid`) REFERENCES  
`directors` (`directorid`) ON DELETE CASCADE ON UPDATE CASCADE,  
  CONSTRAINT `fk_m2d_mid` FOREIGN KEY (`movieid`) REFERENCES `movies`  
(`movieid`) ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

u2base

Creación de la tabla:

```
`userid` int(11) NOT NULL DEFAULT 0,  
`movieid` mediumint(8) unsigned NOT NULL,  
`rating` varchar(45) NOT NULL,  
PRIMARY KEY (`userid`,`movieid`),  
KEY `fk_u2base_uid_idx` (`userid`),  
KEY `fk_u2base_mid_idx` (`movieid`),  
KEY `rating` (`rating`),  
CONSTRAINT `fk_u2base_mid` FOREIGN KEY (`movieid`) REFERENCES  
`movies` (`movieid`) ON DELETE CASCADE ON UPDATE CASCADE,  
CONSTRAINT `fk_u2base_uid` FOREIGN KEY (`userid`) REFERENCES `users`  
(`userid`) ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Consultas básicas

Para ejemplificar la operación de producto cartesiano puede hacerse la siguiente consulta que muestra la relación de los actores con todos los directores, y en principio no representa información relevante:

```
SELECT *  
FROM actors,directors
```

Un ejemplo con las otras operaciones del algebra relacional básicas sería la siguiente consulta que consiste en obtener los ids de actrices femeninas y de la película donde actuó:

```
SELECT A.actorid,M2A.movieid  
FROM actors A JOIN movies2actors M2A USING (actorid)  
WHERE a_gender='F'
```

Donde SELECT representa la operación proyección (π) y WHERE a la operación de selección (σ). También vale aclarar que en este caso se realiza un JOIN que consiste en combinar una operación de selección con un producto cartesiano, y una operación de renombrar a la tabla 'actors' y 'movie2actors'.

Pasando a las operaciones de conjunto, una unión se representa de la siguiente forma, donde se une al conjunto de usuarios mayor a 10 y a los usuarios femeninas:

```
(
    SELECT *
    FROM users
    WHERE age>10
)
UNION
(
    SELECT *
    FROM users
    WHERE u_gender='F'
)
```

Similar al caso anterior, pero esta vez se seleccionan a los usuarios que sean mayores de 10 Y sean femeninos.

```
SELECT *
FROM users
WHERE age>10
INTERSECT
(
    SELECT *
    FROM users
    WHERE u_gender='F'
)
```

La última operación de conjuntos es la diferencia, esta vez se seleccionan a los usuarios que sean mayores de 10 pero que no sean mujeres.

```
SELECT *
FROM users
WHERE age>10
EXCEPT
SELECT *
FROM users
WHERE u_gender='F'
```

La operación de división podría ejecutarse mediante la siguiente consulta, que consiste en obtener todos los usuarios que hayan dado su opinión en la película de id 1711384.

```
SELECT *
FROM users
WHERE userid IN
(
    SELECT userid
    FROM u2base
    WHERE movieid=1711384
)
```

Finalmente, la operación de agrupamiento puede hacerse mediante la consulta que consiste en agrupar todas las películas según los géneros que puedan tener y contarlas.

```
SELECT genre, COUNT(movieid) as cantidadDePelículas
FROM movies2directors
GROUP BY genre
```

Investigación

Problema

Se plantea como problema a la pregunta de si es más significativo para las críticas a una película realizar un cambio en sus actores, o, por otro lado, cambiar de directores.

Hipótesis

El hecho de cambiar de directores afectará de manera más significativa a las críticas que con un cambio de actores

Desarrollo

Como primer paso, sería útil crear una vista que contenga la valoración promedio de cada película, para esto se ejecuta la siguiente sentencia:

```
CREATE OR REPLACE VIEW avg_rating_movies
AS SELECT movieid, AVG(rating) as avg_rating
FROM u2base INNER JOIN users USING (userid)
GROUP BY movieid
```

Posteriormente se debería seleccionar que actores y directores elegir para el análisis, se elegirán directores y actores que participen en películas donde haya gran cantidad de actores o directores, respectivamente, y en lo posible que también tengan calidades distintas.

Para esto ejecutamos la siguiente consulta, para obtener el id de los 10 actores que participen en películas con la mayor cantidad de directores diferentes y la respectiva cantidad:


```
SELECT actorid, COUNT(DISTINCT(directorid)) as directores
FROM movies2actors INNER JOIN movies USING (movieid)
INNER JOIN movies2directors USING (movieid)
GROUP by actorid
ORDER BY directores DESC
LIMIT 0,10
```

actorid	directores
ID de actor a vincular	▼ 1
1694741	56
2092485	36
745232	32
775508	31
342571	31
2378921	31
1676554	30
362804	29
1960871	28
375938	28


Dando como resultado 10 actorids. Se verifican las calidades de los mismos mediante otra consulta y usando como entrada las ids obtenidas anteriormente:

```
SELECT actorid,a_quality
FROM actors
WHERE actorid in
(
  SELECT actorid
  FROM movies2actors INNER JOIN movies USING (movieid)
  INNER JOIN movies2directors USING (movieid)
  GROUP BY actorid
  ORDER BY COUNT(DISTINCT(directorid)) DESC
  LIMIT 0,10
)
```

Error

consulta SQL: [Copiar](#) 

```
SELECT actorid,a_quality
FROM actors
WHERE actorid in
(
  SELECT actorid
  FROM movies2actors INNER JOIN movies USING (movieid)
```

MySQL ha dicho: 

#1235 - Esta versión de MariaDB no soporta todavía 'LIMIT & IN/ALL/ANY/SOME subquery'

Esta consulta puede no funcionar en algunas versiones de MariaDB/SQL, como ocurrió en este caso, por lo tanto, existe una forma alternativa de hacerla:

```

SELECT actorid,a_quality
FROM actors
WHERE actorid in
(
    SELECT actorid
    FROM
    (
        SELECT actorid
        FROM movies2actors INNER JOIN movies USING (movieid)
        INNER JOIN movies2directors USING (movieid)
        GROUP by actorid
        ORDER BY COUNT(DISTINCT(directorid)) DESC
        LIMIT 0,10
    ) as a
)

```

actorid	a_quality
ID única de actor	(0-5)Dato que representa simbólicamente la calidad...
1694741	3
2092485	3
745232	4
342571	3
2378921	3
775508	4
1676554	3
362804	3
1960871	3
375938	4

Se observa que la calidad varía entre 3 y 4, no se encuentran actores de menor calidad entre los 10 primeros.

Se realiza un procedimiento similar al anterior, pero con los directores:

```

SELECT directorid, COUNT(DISTINCT(actorid)) as actores
FROM movies2directors INNER JOIN movies USING (movieid)
INNER JOIN movies2actors USING (movieid)
GROUP by directorid
ORDER BY actores DESC
LIMIT 0, 10

```

directorid	actores
ID de director a vincular	▼ 1
5185	1075
156165	885
151963	843
260322	780
121794	772
264169	768
70986	656
246973	656
158572	620
209221	587

Y para la calidad:

```
SELECT *
FROM directors
WHERE directorid IN
(
    SELECT directorid
    FROM
    (
        SELECT directorid
        FROM movies2actors INNER JOIN movies USING (movieid)
        INNER JOIN movies2directors USING (movieid)
        GROUP by directorid
        ORDER BY COUNT(DISTINCT(actorid)) DESC
        LIMIT 0,10
    ) AS a
)
```

Mostrando filas 0 - 24 (total de 2178, La consulta tardó 0,0499 segundos.)

```
SELECT * FROM directors WHERE directorid IN ( SELECT directorid FROM movies2actors INNER JOIN movies USING (movieid) INNER JOIN movies2directors USING (movieid) GROUP by directorid ORDER BY COUNT(DISTINCT(actorid)) DESC LIMIT 0,10 ) AS a );
```

☐ Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

1 > >> Número de filas: 25 Filtrar filas: Buscar en esta tabla Sort by key: Ninguna

+ Opciones

	directorid	d_quality	avg_revenue
	ID única de director	(0-5) (Dato simbólico que representa la calidad del	(0-4) (Dato simbólico que representa el nivel de ing...
<input type="checkbox"/> Editar Copiar Borrar	67	4	1
<input type="checkbox"/> Editar Copiar Borrar	92	2	3
<input type="checkbox"/> Editar Copiar Borrar	284	4	0
<input type="checkbox"/> Editar Copiar Borrar	708	4	1
<input type="checkbox"/> Editar Copiar Borrar	746	4	4
<input type="checkbox"/> Editar Copiar Borrar	836	2	3
<input type="checkbox"/> Editar Copiar Borrar	947	3	0
<input type="checkbox"/> Editar Copiar Borrar	1151	3	2
<input type="checkbox"/> Editar Copiar Borrar	1756	2	2
<input type="checkbox"/> Editar Copiar Borrar	1836	3	2
<input type="checkbox"/> Editar Copiar Borrar	1851	4	4
<input type="checkbox"/> Editar Copiar Borrar	3134	3	1
<input type="checkbox"/> Editar Copiar Borrar	3707	3	2
<input type="checkbox"/> Editar Copiar Borrar	4164	4	3

Consola

Por alguna razón esta consulta genera resultados erróneos, devolviendo más de 10 directores.

Una posible solución es intentar crear una vista con la subconsulta:

```
CREATE OR REPLACE VIEW top10dir AS
SELECT directorid
FROM movies2directors INNER JOIN movies USING (movieid)
INNER JOIN movies2actors USING (movieid)
GROUP by directorid
ORDER BY COUNT(DISTINCT(actorid)) DESC
LIMIT 0, 10
```

```
SELECT *
FROM directors
WHERE directorid IN
(
    SELECT *
    FROM top10dir
)
```

Pero tampoco da resultados...

Así que se procede a escribir manualmente las 10 IDs resultantes

```
SELECT *
FROM directors
WHERE directorid=5185 OR directorid=156165 OR directorid=151963
OR directorid=260322 OR directorid=121794 OR directorid=264160
OR directorid=70986 OR directorid=246973 OR directorid=158572 OR
directorid=209221
```

directorid	d_quality	avg_revenue
ID única de director	(0-5)Dato simbólico que representa la calidad del ...	(0-4)Dato simbólico que representa el nivel de ing...
5185	4	3
70986	4	4
121794	4	4
151963	3	4
156165	3	3
158572	4	4
209221	4	4
246973	4	4
260322	4	4

Similar al caso de los actores, hay muchos directores de calidad 4 y pocos de 3, y lo mismo con los ingresos promedios, entre los primeros 10 directores.

Ahora quedaría obtener el valor promedio de las notas promedio de las películas donde participa cada actor y director:

```
SELECT actorid, AVG(avg_rating) AS notaPromedio
FROM avg_rating_movies INNER JOIN movies2actors USING (movieid)
WHERE actorid IN
(
    SELECT actorid
    FROM
    (
        SELECT actorid
        FROM movies2actors INNER JOIN movies USING (movieid)
        INNER JOIN movies2directors USING (movieid)
        GROUP BY actorid
        ORDER BY COUNT(DISTINCT(directorid)) DESC
        LIMIT 0,10
    ) AS a
)
GROUP BY actorid
```

actorid	notaPromedio
342571	3.1081419660312513
362804	3.157426504368889
375938	3.546723282332204
667623	3.3389082359073177
745232	3.371899406158162
775508	3.4748731894673823
1676554	3.2499088417668327
1694741	2.9407819288932973
2092485	3.077891898552123
2378921	2.8825249999783416

Notar que esta consulta es muy costosa, puede tardar unos 40 segundos aproximadamente.

Con un agregado podemos obtener la media y el desvío estándar de estas notas:

```
SELECT AVG(notaPromedio) AS promedioNotas, STDDEV(notaPromedio) AS
desvioEstandar
FROM
(
  SELECT actorid, AVG(avg_rating) AS notaPromedio
  FROM avg_rating_movies INNER JOIN movies2actors USING (movieid)
  WHERE actorid IN
  (
    SELECT actorid
    FROM
    (
      SELECT actorid
      FROM movies2actors INNER JOIN movies USING (movieid)
      INNER JOIN movies2directors USING (movieid)
      GROUP BY actorid
      ORDER BY COUNT(DISTINCT(directorid)) DESC
      LIMIT 0,10
    ) AS a
  )
  GROUP BY actorid
) AS b
```

promedioNotas	desvioEstandar
3.21490802534558	0.20929184941849863

Ahora se realiza el mismo proceso con los directores:

```
SELECT directorid, AVG(avg_rating) AS notaPromedio
FROM movies2directors INNER JOIN avg_rating_movies USING (movieid)
WHERE directorid=5185 OR directorid=156165 OR directorid=151963
OR directorid=260322 OR directorid=121794 OR directorid=264160
OR directorid=70986 OR directorid=246973 OR directorid=158572 OR
directorid=209221
GROUP BY directorid
```

directorid	notaPromedio
ID de director a vincular	
5185	3.6795321218027213
70986	3.429785483554946
121794	3.4484427299033573
151963	3.234344396216297
156165	3.2620101255280143
158572	3.3698777582689483
209221	3.456077352614551
246973	3.7461493544291082
260322	3.8607539301781046

```

SELECT AVG(notaPromedio) AS promedioNotas, STDDEV(notaPromedio) AS
desvioEstandar
FROM
(
  SELECT directorid, AVG(avg_rating) AS notaPromedio
  FROM movies2directors INNER JOIN avg_rating_movies USING(movieid)
  WHERE directorid=5185 OR directorid=156165 OR directorid=151963
  OR directorid=260322 OR directorid=121794 OR directorid=264160
  OR directorid=70986 OR directorid=246973 OR directorid=158572 OR
directorid=209221
  GROUP BY directorid
) as b

```

promedioNotas	desvioEstandar
3.498552583610672	0.20456108508889986

Conclusión

Para comparar ambas medias se realiza un test t:

Unpaired t test results

P value and statistical significance:
The two-tailed P value equals 0,0067
By conventional criteria, this difference is considered to be very statistically significant.

Confidence interval:
The mean of Actors minus Directors equals -0,2836445800
95% confidence interval of this difference: From -0,4780773392 to -0,0892118208

Intermediate values used in calculations:
t = 3,0649
df = 18
standard error of difference = 0,093

Learn more:
GraphPad's web site includes portions of the manual for GraphPad Prism that can help you learn statistics. First, review the meaning of [P values](#) and [confidence intervals](#). Then learn how to interpret results from an [unpaired](#) or [paired](#) t test. These links include GraphPad's popular [analysis checklists](#).

Review your data:

Group	Actors	Directors
Mean	3,2149080000	3,4985525800
SD	0,2092918000	0,2045610000
SEM	0,0661838784	0,0646878680
N	10	10

Se puede ver que por lo general un cambio de directores genera películas con menor puntaje. Respecto a como varían al elegir si cambiar directores o actores no existe gran diferencia observando los desvíos de cada apartado.

También cabe recalcar que en ambos grupos la calidad era alta, lo que indica que por lo general los actores y directores de mayor calidad e ingresos son los que trabajan en proyectos variados.

Interfaz de acceso a datos para usuarios

Para este apartado se utilizó una página HTML con PHP, ya que resulta muy simple su utilización directamente mediante XAMPP, además PHP es un lenguaje open source, rápido, requiere de poco mantenimiento y tiene una muy buena integración al uso de bases de datos (Mino, 2021).

Enter user:

Enter movie genre:

Recomendar

La página consiste en dos apartados de texto donde el usuario puede, opcionalmente, ingresar su número de usuario y un género de película el cuál desee una recomendación, al presionar el botón “Recomendar” se generar las 10 ids de películas con mayor puntaje las cuales pertenecerán al género que el usuario ingreso, si es que lo hizo, y que no haya puntuado según la id que proveyó o no.

El código de la página se encuentra en el anexo.

Consultas adicionales

Obtener la cantidad de películas, por género, que han puntuado usuarios mayores a 40 años y ordenarlas por esa cantidad.

```
SELECT genre, COUNT(movieid) as cantidadPelículas
FROM u2base INNER JOIN users USING (userid)
INNER JOIN movies USING (movieid)
INNER JOIN movies2directors USING (movieid)
WHERE age>40
GROUP BY genre
ORDER BY cantidadPelículas DESC
```

Obtener los datos de los directores que hayan dirigido una película con más de 4.5 de valoración promedio

```
SELECT *
FROM directors D
WHERE EXISTS
(
    SELECT *
    FROM movies2directors M2D INNER JOIN avg_rating_movies ARM ON
    ARM.movieid=M2D.movieid
    WHERE avg_rating>4.5 AND D.directorid = M2D.directorid
)
```

Obtener los usuarios y la cantidad de películas puntuadas que pertenezcan a las 10 mejores películas según su puntaje promedio.

```
SELECT userid, COUNT(movieid) as pelisPuntuadas
FROM u2base
WHERE movieid IN
(
    SELECT *
    FROM
    (
        SELECT movieid
        FROM avg_rating_movies
        ORDER BY avg_rating DESC
        LIMIT 0,10
    ) as A
)
GROUP BY userid
ORDER BY pelisPuntuadas DESC
```

Referencias

- HostingPedia. (24 de Enero de 2019). *HostingPedia*. Recuperado el 28 de Octubre de 2021, de HostingPedia: <https://hostingpedia.net/mysql.html>
- IONOS. (19 de Octubre de 2020). *Ionos by 1&1 Digital Guide*. Recuperado el 28 de Octubre de 2021, de Ionos by 1&1 Digital Guide: <https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/que-es-innodb/>
- Jesuïtes Educació. (26 de Febrero de 2018). *Blog Jesuïtes Educació*. Recuperado el 28 de Octubre de 2021, de Blog Jesuïtes Educació: <https://fp.uoc.fje.edu/blog/por-que-elegir-el-gestor-de-base-de-datos-mysql/>
- Mino, S. (15 de Abril de 2021). *JOBSITY*. Recuperado el 10 de Noviembre de 2021, de JOBSITY: <https://www.jobsity.com/blog/8-reasons-why-php-is-still-so-important-for-web-development>
- Tellado, F. (03 de Marzo de 2020). *AyudaWP*. Recuperado el 28 de Octubre de 2021, de Ayuda WordPress: <https://ayudawp.com/myisam-innodb/>

Anexo

Sentencias INSERT:

actors

```
INSERT INTO `actors` VALUES
(4,'M',4),(16,'M',0),(28,'M',4),(566,'M',4),(580,'M',4),(636,'M',0),(7
69,'M',4),(797,'M',4),(802,'M',4),(808,'M',3),(817,'M',0),(851,'M',3),
(888,'M',4),(894,'M',3),(923,'M',4),(962,'M',4),(978,'M',3),(989,'M',4),
(1143,'M',3),(1276,'M',3),(1304,'M',3),(1348,'M',0),(1391,'M',4),(15
17,'M',4),(1535,'M',3),(1554,'M',4),(1556,'M',4);
```

directors

```
INSERT INTO `directors` VALUES
(67,4,1),(92,2,3),(284,4,0),(708,4,1),(746,4,4),(836,2,3),(947,3,0),(1
151,3,2),(1756,2,2),(1836,3,2),(1851,4,4),(3134,3,1),(3707,3,2),(4164,
4,3),(4451,3,3),(4621,4,4),(4637,3,2),(4930,2,3),(4997,2,0),(5049,3,1),
(5088,3,1),(5185,4,3),(5201,4,4),(5308,3,0),(5425,3,2),(5442,4,2),(57
63,4,2),(5798,4,3),(6057,2,4),(6301,2,2),(6429,4,2),(6458,4,1),(6523,3
,4),(6981,3,2),(7117,3,2),(7212,3,0),(7309,4,0),(7335,4,3),(7365,4,4),
(7366,3,4),(7387,0,0),(7456,4,3),(7682,3,1),(8157,4,2),(8168,4,1),(817
1,4,2),(8343,4,3),(8349,4,4),(8468,4,3),(8685,4,1),(8702,4,3);
```

movies

```
INSERT INTO `movies` VALUES
(1672052,3,'T','other',2),(1672111,4,'T','other',2),(1672580,4,'T','US
A',3),(1672716,4,'T','USA',2),(1672946,4,'T','USA',0),(1673647,3,'F','
France',3),(1673658,4,'T','USA',2),(1673848,4,'T','USA',2),(1674388,4,
'T','USA',2),(1674737,3,'T','USA',2),(1677011,4,'T','USA',2),(1677258,
3,'F','France',1),(1677346,4,'T','USA',1),(1677472,1,'T','USA',1),(167
7516,4,'T','USA',2),(1677631,3,'T','USA',3),(1678036,3,'T','USA',3),(1
678591,4,'T','UK',2),(1679029,4,'T','USA',2),(1679461,4,'T','USA',1),(
1679507,4,'T','USA',1),(1679920,4,'F','other',2),(1681633,4,'T','USA',
3),(1681655,4,'T','USA',2),(1682630,3,'F','USA',1),(1682657,4,'T','USA
',1),(1682692,4,'T','USA',2),(1682746,4,'T','UK',3),(1682922,4,'T','US
A',3),(1684201,4,'T','USA',2),(1684430,4,'T','USA',3),(1684442,4,'T','
USA',1),(1684486,4,'T','USA',1),(1684910,4,'T','USA',1),(1685241,2,'T'
,'USA',1),(1685309,3,'T','USA',1),(1685405,4,'T','USA',3),(1685463,3,'
T','UK',3),(1685493,4,'T','UK',1),(1685815,4,'T','USA',2),(1686071,4,'
T','other',2),(1686209,2,'T','USA',2),(1687181,4,'T','USA',2),(1687191
,1,'T','USA',0),(1687544,3,'T','USA',3),(1687685,4,'T','USA',2),(16877
30,4,'T','USA',2),(1687738,2,'T','USA',1);
```

users

INSERT INTO `users` VALUES

```
(1,'1','F','2'),(51,'1','F','2'),(75,'1','F','2'),(86,'1','F','2'),(99,'1','F','2'),(119,'1','F','2'),(194,'1','F','2'),(210,'1','F','2'),(468,'1','F','2'),(470,'1','F','2'),(484,'1','F','2'),(606,'1','F','2'),(629,'1','F','2'),(634,'1','F','2'),(888,'1','F','2'),(1045,'1','F','2'),(1088,'1','F','2'),(1154,'1','F','2'),(1195,'1','F','2'),(1348,'1','F','2'),(1365,'1','F','2'),(1421,'1','F','2'),(1434,'1','F','2'),(1508,'1','F','2'),(1563,'1','F','2'),(1568,'1','F','2'),(1929,'1','F','2'),(2133,'1','F','2'),(2137,'1','F','2'),(2155,'1','F','2'),(2162,'1','F','2'),(2202,'1','F','2'),(2285,'1','F','2'),(2346,'1','F','2'),(2418,'1','F','2'),(2657,'1','F','2'),(2810,'1','F','2');
```

movies2actors

INSERT INTO `movies2actors` VALUES

```
(1672580,981535,0),(1672946,1094968,0),(1673647,149985,0),(1673647,261595,0),(1673647,781357,0),(1673647,893869,0),(1673647,1806262,0),(1673647,1944214,0),(1673647,2464501,0),(1673658,573391,0),(1673658,601871,0),(2211336,1826900,2),(2211336,1941919,2),(2211336,1950861,2),(2211336,2191576,2),(2211336,2516171,2),(2211336,2680475,2),(2211336,2719749,2);
```

movies2directors

INSERT INTO `movies2directors` VALUES

```
(1672111,54934,'Action'),(1672946,188940,'Action'),(1679461,179783,'Action'),(1691387,291700,'Action'),(1693305,14663,'Action'),(1698667,98630,'Action'),(1704645,98630,'Action'),(1705417,19141,'Action'),(1706887,221541,'Action'),(1707277,52953,'Action'),(1709238,108073,'Action'),(1711133,260603,'Action'),(1711175,213822,'Action'),(1711393,128558,'Action'),(1711398,256455,'Action'),(1715380,130253,'Action'),(1715404,39629,'Action'),(1715607,172775,'Action'),(1717455,196811,'Action'),(1719502,69968,'Action'),(1722327,56159,'Action'),(1731582,13745,'Action'),(1736268,19189,'Action'),(1736362,158089,'Action'),(1736794,54543,'Action'),(1740732,123402,'Action'),(1740752,70986,'Action'),(1740811,41939,'Action'),(1744811,230882,'Action'),(1745530,4997,'Action');
```

u2base

INSERT INTO `u2base` VALUES

```
(2,1964242,'1'),(2,2219779,'1'),(3,1856939,'1'),(4,2273044,'1'),(5,1681655,'1'),(5,1695219,'1'),(5,1915498,'1'),(5758,2253332,'1'),(5758,2371786,'1'),(5758,2457464,'1'),(5758,2462958,'1'),(5758,2462959,'1'),(5758,2558138,'1'),(2820,2407502,'2'),(2820,2408078,'2'),(2820,2409105,'2'),(2820,2411035,'2'),(2820,2412502,'2'),(2820,2414225,'2'),(5777,1781518,'2'),(5777,1792713,'2'),(5777,2006702,'2'),(5777,2034166,'2'),(5777,2355625,'2'),(5777,2371786,'2'),(3626,1831593,'3'),(3626,1832852,'3'),(3626,1833175,'3'),(3626,1836375,'3'),(3626,1837469,'3'),(3626,1838160,'3'),(999,2258328,'4'),(999,2262464,'4'),(999,2263670,'4'),(999,2273174,'4'),(999,2276841,'4'),(999,2280997,'4'),(5488,2286681,'4'),(5488,2310159,'4'),(5488,2338963,'4'),(5488,2357080,'4'),(5488,2363738,'4'),(5488,2371045,'4'),(3476,2069192,'5'),(3476,2075122,'5'),(3476,2094683,'5'),(3476,2103975,'5'),(3476,2116072,'5'),(3476,2117874,'5'),(4826,2048619,'5'),(4826,2109093,'5'),(4826,2172808,'5'),(4826,2178276,'5'),(4826,2324123,'5'),(4826,2325897,'5');
```

Código de la página:

```
<html>
<body>
<form name="Opciones" action="index.php" method="POST">
    Enter user:<input name="Usuario" type="text">
    <p>Enter movie genre:<input name="Genero" type="text"></p>
    <p><input type="submit" name="buscar" value="Recomendar"
type="button"></p>
</form>
<?php
/*
Autor: Salim Nasim
Fecha: 10/11/2021
*/

error_reporting(0);

//Realizo la conexión con la base de datos.
$db = new mysqli("localhost","root","","movielens");

//Ejecuto si se presiona el botón
$action = $_POST['buscar'];
if($accion=='Recomendar')
{
    //Cargo las variables por POST
    $user = $_POST['Usuario'];
    $genero = $_POST['Genero'];

    //Muestro el encabezado
    echo "Top 10 recommendations";
    if(!empty($user)) echo " for $user";
    if(!empty($genero)) echo " of $genero";
    echo "<br>";

    //Genero la consulta sql en base a los datos que tengo o no
    $consulta;

    if(empty($user))
    {
        if(empty($genero))
            $consulta= "SELECT movieid
                        FROM avg_rating_movies
                        ORDER BY avg_rating DESC
                        LIMIT 0,10";
        else
            $consulta= "SELECT movieid
                        FROM avg_rating_movies INNER JOIN movies2di-
rectors USING(movieid)
                        WHERE genre LIKE '$genero'
                        ORDER BY avg_rating DESC
                        LIMIT 0,10";
    }
}
```

```

else if(empty($genero))
{
    $consulta= "SELECT movieid
                FROM avg_rating_movies
                WHERE movieid NOT IN
                (
                    SELECT movieid
                    FROM u2base
                    WHERE userid=$user
                )
                ORDER BY avg_rating DESC
                LIMIT 0,10";
}
else
    $consulta= "SELECT movieid
                FROM avg_rating_movies INNER JOIN
movies2directors USING(movieid)
                WHERE genre LIKE '$genero'
                AND movieid NOT IN
                (
                    SELECT movieid
                    FROM u2base
                    WHERE userid=$user
                )
                ORDER BY avg_rating DESC
                LIMIT 0,10";

//Ejecuto la consulta SQL
$q = $db->query($consulta);

//Creo un array asociativo con las tuplas y lo recorro con un
while
while($r=$q->fetch_assoc())
{
    //Asigno cada columna a una variable
    $name = $r['movieid'];

    //Muestro los valores
    echo "$name<br>";
}

}

?>
<br>
</body>
</html>

```