

## Data engineering: Relational database optimization report

For this project, we were provided a database with plenty of issues that needed to be enhanced according to SQL database norms. The database MoviesDB contained initially 4 tables in total plus a tsv file which was later imported as a 5th table.

In this report, the process of optimizing and normalizing the database will be discussed through the different adjustments made, why and how they were executed.

### Major adjustments:

#### - Addition of tsv file

Firstly, a provided tsv file was inserted into the database under the name titleRatings, this was done using sFTP (Secure File Transfer Protocol), after connecting to the server via SSH, the following script was used to insert this data:

```
LOAD DATA LOCAL INFILE 'title.ratings.tsv'  
INTO TABLE titleRatings  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n'  
IGNORE 0 LINES  
(tconst, averageRating, numVotes)
```

#### - Conversion of tconst & nconst

First thing noticed are the tables titleBasics and nameBasics, these two form the core of the database, they contain the basic information of titles and individuals (actors, producers, directors, etc.). The columns tconst and nconst are thus unique identifiers to respectively titles and people, meaning they are primary keys.

The data type used for these two primary keys however is initially VARCHAR due to a mixture of letters and numbers, this data type is not ideal and primary keys are preferably of type INT, this results in easily manageable tables with faster runtime queries, integers are also easier to index.

The first two letters in tconst and nconst have no significance and are thus dropped and the rest is casted into an integer type using the following query:

```
UPDATE nameBasics  
SET nameID = (CAST(substring(nconst,3) AS unsigned));
```

```
UPDATE titleBasics  
SET titleID = (CAST(substring(tconst,3) AS unsigned));
```

The same query is run for each recurrence of tconst and nconst in other tables, the two columns are also relabeled as titleID and personID for more readability.

- **Split comma separated values in one row**

Furthermore, some columns such as, knownForTitles, primaryProfessions, genres and characters, consisted of a list of attributes separated by commas. This is a violation of the first norm.

Solving this issue would either be through enumeration or splitting these values into different rows (likely in a new table). Enumeration is not easily expandable in case the user wishes to add data, hence why we decided to split these values.

This was executed through a stepwise process:

Firstly we need to know the maximum number of possible values in one row, for knownForTitles, this is done with the query:

```
SELECT MAX(CHAR_LENGTH(knownForTitles) -  
CHAR_LENGTH(REPLACE(knownForTitles, ',', '')))  
FROM nameBasics;
```

This returns 7, meaning that there can be a max of 8 titles a person can be known for. Therefore, we create a temporary table containing 8 numbers, which will serve to iterate through each row or "list":

```
CREATE TEMPORARY TABLE numbers AS (  
SELECT 1 as n  
UNION SELECT 2 as n  
UNION SELECT 3 as n  
UNION SELECT 4 as n  
UNION SELECT 5 as n  
UNION SELECT 6 as n  
UNION SELECT 7 as n  
UNION SELECT 8 as n);
```

Next is we split the values and join the temporary table to store these values using the following query:

```
SELECT personID, knownForTitles, n FROM nameBasics  
JOIN numbers ON  
CHAR_LENGTH(knownForTitles) - CHAR_LENGTH(REPLACE(knownForTitles, ',', '')) >=  
n - 1  
ORDER BY personID, n;
```

```

1 • SELECT person_id, knownForTitles, n FROM nameBasics
2 JOIN numbers ON
3 CHAR_LENGTH(knownForTitles) - CHAR_LENGTH(REPLACE(knownForTitles, ",", "")) >= n - 1
4 ORDER BY person_id, n;

```

Result Grid		
Filter Rows: <input type="text"/>		
Export:		
Wrap Cell Content:		
Fetch rows:		
person_id	knownForTitles	n
1	tt0050419,tt0053137,tt0072308,tt0043044	1
1	tt0050419,tt0053137,tt0072308,tt0043044	2
1	tt0050419,tt0053137,tt0072308,tt0043044	3
1	tt0050419,tt0053137,tt0072308,tt0043044	4
2	tt0071877,tt0117057,tt0038355,tt0037382	1
2	tt0071877,tt0117057,tt0038355,tt0037382	2
2	tt0071877,tt0117057,tt0038355,tt0037382	3
2	tt0071877,tt0117057,tt0038355,tt0037382	4
3	tt0054452,tt0049189,tt0059956,tt0057345	1
3	tt0054452,tt0049189,tt0059956,tt0057345	2
3	tt0054452,tt0049189,tt0059956,tt0057345	3

Finally, we created a new table with corresponding columns and insert each time the values from the temporary table, using the query:

```

INSERT INTO knownForTitleTable
SELECT personID, SUBSTRING_INDEX(SUBSTRING_INDEX(knownForTitles, ",", n), ",",
-1) AS titleID FROM nameBasics
JOIN numbers ON
CHAR_LENGTH(knownForTitles) - CHAR_LENGTH(REPLACE(knownForTitles, ",", "")) >=
n - 1;

```

The same procedure is repeated each time for genres, characters and primaryProfessions.

#### - Getting rid of excessive NULLI values

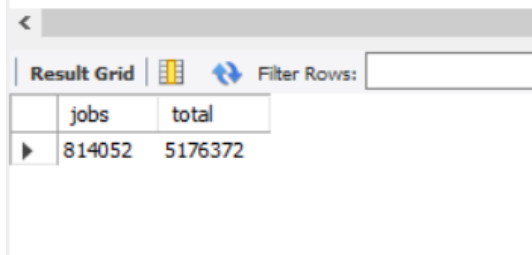
It is at first glance very remarkable that some tables have columns with many NULL value rows. In titlePrincipals for instance, the column "jobs" seems almost all sparse, this would make the column uselessly time consuming for the table.

Checking the ratio of the non NULL jobs to the total number of rows in the titlePrincipals:

```

1  SELECT count(job) AS jobs,
2  count(*) AS total
3  FROM titlePrincipals;

```



	jobs	total
▶	814052	5176372

This is a ratio of less than 16%, meaning that only 16% of the rows in titlePrincipals have an assigned value for “jobs”. It is better to have these jobs in a separate table referencing to titlePrincipals via a foreign key. This saves plenty of time and removes a lot of excessive NULL values from the original table.

```

INSERT INTO titlePrincipalsJobs (job, titlePrincipalsID)
SELECT job, titlePrincipalsID
FROM titlePrincipals
WHERE job IS NOT NULL;

```

The same procedure was executed for all columns with low ratios (<40%) of non NULL values, namely for attributes, language, originalTitles, titleEndYear, akaTypes.

Obviously not all NULL values were removed, it is for instance inevitable for column

“deathYear” in table nameBasics, as it is logical that not all individuals have a deathYear.

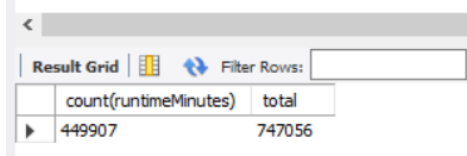
In some cases, it is also not very beneficial to create a new table when the ratio is around

50%, as adding an extra table only adds to the load of the database. This is the case for for example “runtimeMinutes” in titleBasics:

```

1  SELECT count(runtimeMinutes),
2  count(*) AS total
3  FROM titleBasics;

```



	count(runtimeMinutes)	total
▶	449907	747056

### - Primary keys

Primary keys are of significant importance in any table in a relational database, and making sure all tables have a unique identifier is part of the normalization process.

It was therefore one of the first things added, tables nameBasics and titleBasics have respectively titleID and personID as primary keys. New unique auto-increment integer IDs were added for the other tables to ensure they have primary keys. For some tables however, for example titlePrincipalsJobs, as titlePrincipalsID is unique in each row of the table, it is both a foreign key as a primary key for titlePrincipalsJobs. Same goes for originalTitles, titleRatings, language, attributes... where the foreign key referencing the old table also serves as the primary key because it's unique. For example, the table characters does have its own primary key. Since titlePrincipalsID within characters is not unique.

#### - Other small changes

In some new tables such as genres, it is remarkable that there is a set number of, for this case, genres which a title could take. By putting these distinct genres in a separate table genreID and allocate an ID to each of them, we can refer to this ID in the genres table, instead of repeating the VARCHAR data type genre. After creating a new table and insert the distinct genres into it, we update the table genres using the following query:

```
UPDATE genres
SET genreID = (SELECT genreID FROM genreID WHERE
genreID.genre = genres.genre);
```

By having more columns of type INT and very limited columns of type VARCHAR, one takes much load off the database and saves time. The same procedure is done for titleTypes and category.

In table titlePrincipals column "characters", the data is enclosed between brackets [ ] and ellipsis " ", these symbols do not necessarily have any significance so it was decided to remove them using the query:

```
UPDATE titlePrincipals
SET charactersNew = (REPLACE (REPLACE (characters, "[", ""), "[", " "));
```

In titleBasics, "isAdult" data type was changed from true/false to TINYINT (0/1) for memory and time saving.

```
UPDATE titleBasics
SET isAdultNew = CASE
WHEN isAdult = 'true' THEN 1 ELSE 0 END
```

#### - Foreign keys

To complete a correct relational database, foreign keys must be added to relate the different tables accordingly. There were many foreign key constraints added, where the new tables would either refer to titleBasics, nameBasics, titleAkas and so on.

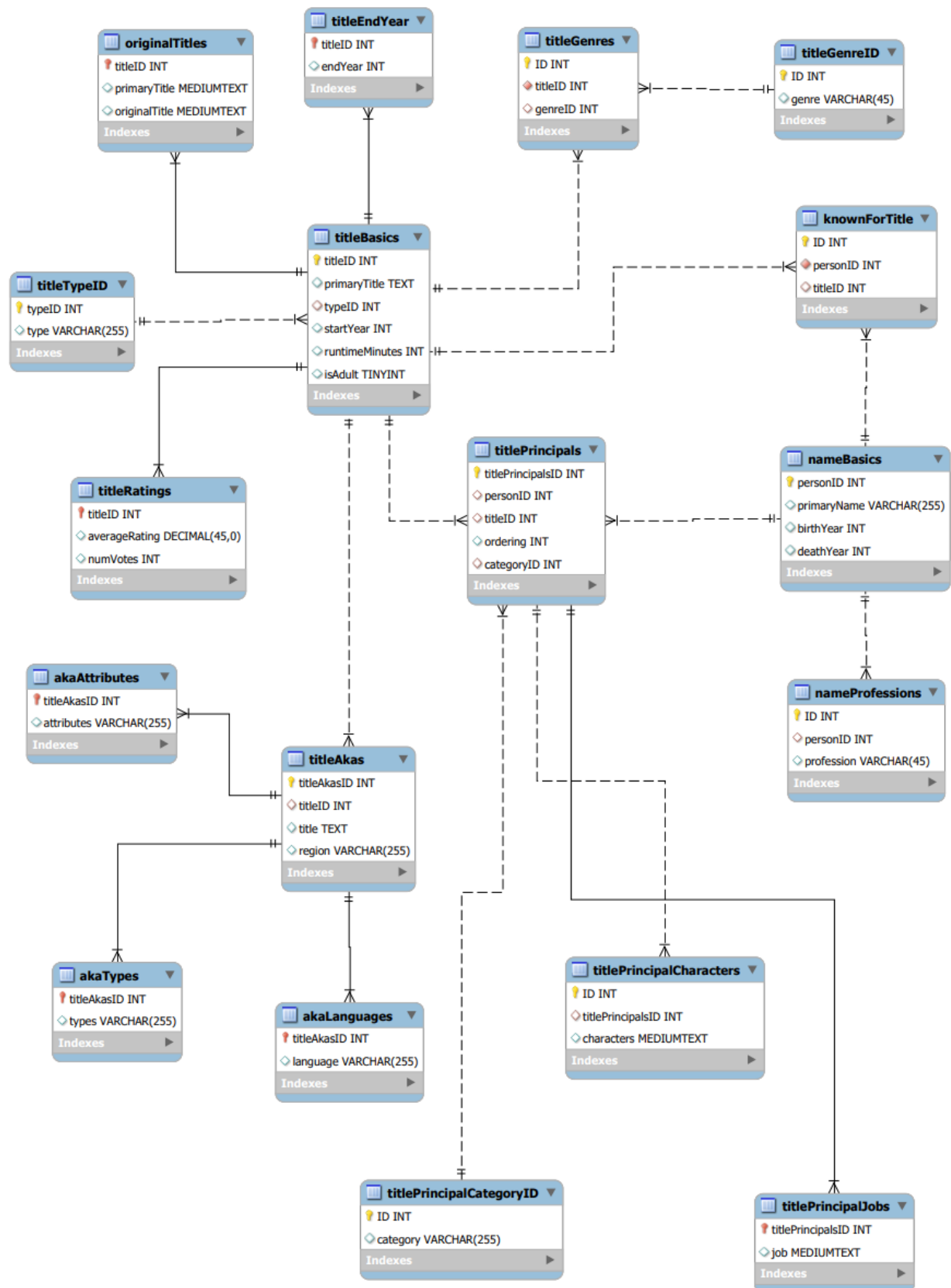
Due to the database not having full integrity, where we basically have titleIDs of titles that aren't in titleBasics, same for titleAkas and titlePrincipals, linking these tables back to titleBasics through foreign key "titleID" wasn't possible, since this defies the principal of a foreign key. This can either be solved by `SET FOREIGN_KEY_CHECKS = 0` or adding these values to the original table.

This is why the missing titles were added to titleBasics as "unknown", this results in a rather sparse table with many NULL values but we think that respecting foreign key constraints is more important. The number of NULL values was after all not of big ratio to titles with actual data.

- Adding indexes

At the very end, indexes were added to the central tables for their primary keys and other important columns. Indexes improve the performance of retrieving data and made the 5 queries run faster.

### **Result & ERD:**



Overview of all tables:

## TitleBasics

titleID	primaryTitle	typeID	startYear	runtimeMinutes	isAdult
1	unknown	NULL	NULL	NULL	0
2	unknown	NULL	NULL	NULL	0
3	unknown	NULL	NULL	NULL	0
4	unknown	NULL	NULL	NULL	0
5	unknown	NULL	NULL	NULL	0
6	unknown	NULL	NULL	NULL	0
7	unknown	NULL	NULL	NULL	0
8	unknown	NULL	NULL	NULL	0
9	Miss Jerry	1	1894	45	0
10	unknown	NULL	NULL	NULL	0
11	unknown	NULL	NULL	NULL	0
12	unknown	NULL	NULL	NULL	0

Ignoring the unknown titles that were added for database integrity:

titleID	primaryTitle	typeID	startYear	runtimeMinutes	isAdult
9	Miss Jerry	1	1894	45	0
147	The Corbett-Fitzsimmons Fight	1	1897	20	0
335	Soldiers of the Cross	1	1900	NULL	0
502	Bohemios	1	1905	100	0
574	The Story of the Kelly Gang	1	1906	70	0
615	Robbery Under Arms	1	1907	NULL	0
630	Hamlet	1	1908	NULL	0
675	Don Quijote	1	1908	NULL	0
676	Don Álvaro o la fuerza del sino	1	1908	NULL	0
679	The Fairylogue and Radio-Plays	1	1908	120	0
739	El pastorcito de Torrente	1	1908	NULL	0
793	Andreas Hofer	1	1909	NULL	0

## OriginalTitles

Not all titles have an original title, non NULL ratio = 4%

So whenever a title has an original title different from its primary title, it was put in a new table, the new table contains primary title just for comparison and better user experience, it is however not adhering to the second norm since primaryTitle is completely dependent on titleID.



titleID	primaryTitle	originalTitle
630	Hamlet	Amleto
886	Hamlet, Prince of Denmark	Hamlet
891	Cycle Rider and the Witch	Heksen og cyklisten
1122	The Red Inn	L'auberge rouge
1175	Camille	La dame aux camélias
1258	The White Slave Trade	Den hvide slavehandel
1338	A Night in May	Maiskaya noch, ili utoplennitsa
1530	The Bushranger's Bride	Captain Midnight, the Bush King

### TitleEndYear

Only a few titles had an endyear, so to get rid of excessive NULL values, they were put in a new table.

Ratio of non NULL values was 1,15%.

titleID	endYear
53538	1960
59993	1966
60329	1965
61292	1967
62586	1969
62607	1968
65292	1969
65322	2005
65341	1970
66678	1971

### TitleTypeID

Used for typeID column in tileBasics since there are only 6 types and using integers makes life easier.

typeID	type
1	movie
2	tvMovie
3	tvShort
4	tvMiniSeries
5	tvSpecial
6	videoGame

### Genres & genreID

Is is the genres a title can have, split the comma separated genres into different rows and set IDs for the limited number of genres recurring to refer to them as integers.

ID	titleID	genreID	ID	genre
1	9	1	1	Romance
2	147	22	2	Documentary
3	147	26	3	Biography
4	147	2	5	Drama
5	335	5	6	Adventure
6	335	3	7	Comedy
7	574	5	8	Crime
8	574	8	9	War
9	574	3	10	Sci-Fi
10	615	5	11	History
			12	Western

**NameBasics**: contains the name, birth year and ID of different people working in the movie industry.

personID	primaryName	birthYear	deathYear
1	Fred Astaire	1899	1987
2	Lauren Bacall	1924	2014
3	Brigitte Bardot	1934	NULL
4	John Belushi	1949	1982
5	Ingmar Bergman	1918	2007
6	Ingrid Bergman	1915	1982
7	Humphrey Bogart	1899	1957
8	Marlon Brando	1924	2004
9	Richard Burton	1925	1984
10	James Cagney	1899	1986
11	Gary Cooper	1901	1961
12	Bette Davis	1908	1989

### Professions

Shows the professions of individuals from personID, was initially a list of comma separated values in titleBasics, since a person can take on multiple professions. So to adhere to the first norm, they were put in a new table referring back to the person via foreign key personID.

ID	personID	profession
1	286006	writer
2	1705708	actress
3	1	miscellaneous
4	1705714	actress
5	286006	actor
6	1705715	actress
7	1705717	actress
8	1	actor
9	286006	director
10	1	soundtrack
11	286010	writer

### KnownForTitle

Shows what titles a person is known for, it was initially a set of titles in one row, so it was split into different rows in a new table and got a unique primary key for each combination of personID and titleID.

ID	personID	titleID
1	1	43044
2	1	72308
3	1	53137
4	1	50419
5	2	37382
6	2	38355
7	2	117057
8	2	71877
9	3	57345
10	3	59956

### TitleAkas

This table contains the different titles of the same movie based off either region or language, these columns also show what regions, languages, types a movie is available in, and the attributes of each unique title of the movie.

They were however put in new tables that refer to the akaTitles through a foreign key because a lot of the values in these columns were NULL.

TitleAkasID represents a unique identifier of a combination of a titleID and its title, which could be in different languages, so each language gets a unique ID.

titleAkasID	titleID	title	region
1	9	Miss Jerry	NULL
2	9	Miss Jerry	HU
3	9	Miss Jerry	US
4	147	The Corbett-Fitzsimmons Fight	US
5	147	Бой Корбетта и Фитцсиммонса	RU
6	147	The Corbett-Fitzsimmons Fight	NULL
7	335	Early Christian Martyrs	AU
8	335	Soldiers of the Cross	AU
9	335	Soldiers of the Cross	NULL
10	502	Bohemios	ES
11	574	Kelly bandájának története	HU

### AkaTypes

Non NULL value ratio was: 42%

titleAkasID	types
1	original
2	imdbDisplay
6	original
7	working
9	original
11	imdbDisplay
12	original
17	original
18	imdbDisplay
20	imdbDisplay

### AkaAttributes

Non NULL value ratio was: 6%

titleAkasID	attributes
33	literal English title
46	informal English title
53	informal English title
57	informal English title
59	transliterated ISO-LATIN-1 title
72	informal literal title
97	literal English title
126	transliterated ISO-LATIN-1 title
162	transliterated ISO-LATIN-1 title
189	complete title

### AkaLanguages

Non NULL value ratio was: 12%

titleAkasID	language
20	SV
33	EN
42	BG
44	EN
46	EN
53	EN
55	EN
57	EN
58	EN
62	RU

### TitlePrincipals

This table contains all unique combinations of a person and a title, what their job in the title was, characters, category and so on.

titlePrincipalsID	personID	titleID	ordering	categoryID
1	1	25164	1	1
2	1	26942	2	1
3	1	27125	1	1
4	1	27630	1	1
5	1	28333	1	1
6	1	28757	1	1
7	1	29546	1	1
8	1	29971	1	1
9	1	31983	1	1
10	1	32284	1	1

### TitlePrincipalCategoryID

All 15 categories were put in one table and allocated one integer ID to use as reference in titlePrincipals instead of their VARCHAR representation.

ID	category
1	actor
2	self
3	archive_footage
4	actress
5	writer
6	director
7	producer
8	cinematographer
9	archive_sound
10	composer

### [TitlePrincipalCharacters](#)

Characters were put on a new table due to it having multiple characters in one row. It represents what character(s) a person played in a certain title.

ID	titlePrincipalsID	characters
1	1176061	Dorothy Lee
2	1176062	Grandma
3	1176063	Sadie Johnson
4	1176064	Mrs. G
5	1176065	Mrs. Thomas
6	1176066	Grandma Annie
7	1176067	Big Ma Mazie
8	1176068	Georgia Mae Jackson
9	1176090	Colonel Hooker
10	1176092	Harrigan Blood

### [TitlePrincipalJobs](#)

TitlePrincipalJobs shows what job a certain individual did in a title, identifying by titlePrincipalID.

It was a sparse column. Non NULL ratio was: 15%

titlePrincipalsID	job
363	screenplay
368	idea
388	written by
405	suggested by: a film by "Sommarnattens leende"
407	producer
411	producer
413	producer
438	producer
447	play
459	play

### TitleRatings

titleID	averageRating	numVotes
1	6	1550
2	6	186
3	7	1207
4	6	113
5	6	1934
6	5	102
7	6	615
8	5	1667
9	5	81
10	7	5545

### Optimizing the 5 queries:

#### Query1: **99.99% lower query cost**

```

SELECT tb.titleID, titleTypeID.type, tb.primaryTitle, cat.category, cha.characters
FROM titleBasics AS tb
LEFT JOIN titleTypeID
ON titleTypeID.typeID = tb.typeID
LEFT JOIN titlePrincipals AS tp
ON tp.titleID = tb.titleID
LEFT JOIN titlePrincipalCategoryID AS cat
ON cat.ID = tp.categoryID
LEFT JOIN titlePrincipalCharacters AS cha
ON cha.titlePrincipalsID = tp.titlePrincipalsID

```

WHERE tb.primaryTitle LIKE 'The S%';

Below screenshots of the output of the old vs the new query, since we couldn't get an identical result:

### Old query:

```
1 • SELECT tb.tconst, tb.titleType, tb.primaryTitle, tp.category, tp.characters
2 FROM titleBasics as tb
3 LEFT JOIN titlePrincipals as tp
4 ON tb.tconst=tp.tconst WHERE tb.primaryTitle LIKE 'The S%';
```

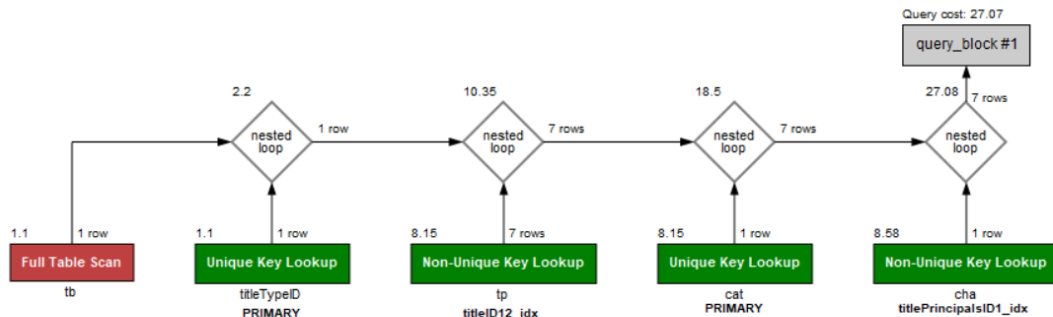
tconst	titleType	primaryTitle	category	characters
tt0019375	movie	The Shopworn Angel	actor	["William Tyler"]
tt0020371	movie	The Ship of Lost Men	actress	["Ethel Marley"]
tt0021412	movie	The Spoilers	actor	["Roy Glenister"]
tt0022395	movie	The Skin Game	director	[NULL]
tt0024598	movie	The Song of Songs	actress	["Lily Czepanek"]
tt0025746	movie	The Scarlet Empress	actress	["Princess Sophia Frederica", "Catherine II"]
tt0025826	movie	The St. Louis Kid	actor	["Eddie Kennedy"]
tt0029543	movie	The Shadow	actress	["Mary Gillespie"]
tt0030755	movie	The Sisters	actress	["Louise Elliott"]
tt0031981	movie	The Story of Alexander Graham Bell	actor	["Thomas Watson"]
tt0031983	movie	The Story of Vernon and Irene Castle	actor	["Vernon Castle"]
tt0034236	movie	The Strawberry Blonde	actress	["Virginia Brush"]
tt0034236	movie	The Strawberry Blonde	actress	["Amy Lind"]

### New query:

```
1 • SELECT tb.titleID, titleTypeID.type, tb.primaryTitle, cat.category, cha.characters
2 FROM titleBasics AS tb
3 LEFT JOIN titleTypeID
4 ON titleTypeID.typeID = tb.typeID
5 LEFT JOIN titlePrincipals AS tp
6 ON tp.titleID = tb.titleID
7 LEFT JOIN titlePrincipalCategoryID AS cat
8 ON cat.ID = tp.categoryID
9 LEFT JOIN titlePrincipalCharacters AS cha
10 ON cha.titlePrincipalsID = tp.titlePrincipalsID
11 WHERE tb.primaryTitle LIKE 'The S%';
12
```

titleID	type	primaryTitle	category	characters
574	movie	The Story of the Kelly Gang	actress	[NULL]
574	movie	The Story of the Kelly Gang	producer	[NULL]
574	movie	The Story of the Kelly Gang	producer	[NULL]
574	movie	The Story of the Kelly Gang	cinematographer	[NULL]
574	movie	The Story of the Kelly Gang	director	[NULL]
574	movie	The Story of the Kelly Gang	actress	Kate Kelly
574	movie	The Story of the Kelly Gang	actor	School Master
574	movie	The Story of the Kelly Gang	producer	[NULL]
574	movie	The Story of the Kelly Gang	composer	[NULL]
574	movie	The Story of the Kelly Gang	actor	Steve Hart
2483	movie	The Sergeant's Boy	actor	[NULL]
2483	movie	The Sergeant's Boy	actress	[NULL]





### Query2: 99.99% lower query cost

```

SELECT tb.titleID, titleTypeID.type, tb.primaryTitle, cat.category, char.characters
FROM titleBasics AS tb
LEFT JOIN titleTypeID
ON titleTypeID.ID = tb.typeID
LEFT JOIN genres
ON genres.titleID = tb.titleID
LEFT JOIN titlePrincipals AS tp
ON tp.titleID = tb.titleID
LEFT JOIN titlePrincipalsCategoryID AS cat
ON cat.ID = tp.categoryID
LEFT JOIN titlePrincipalCharacters AS char
ON char.titlePrincipalsID = tp.titlePrincipalsID
WHERE genres.genreID = 5;
  
```

Below screenshots of the output of the old vs the new query, since we couldn't get an identical result:

#### Old query:

```

1 • SELECT tb.tconst, tb.titleType, tb.primaryTitle, tp.category, tp.characters
2 FROM titlePrincipals as tp
3 LEFT JOIN titleBasics as tb
4 ON tp.tconst=tb.tconst WHERE tb.genres = 'Drama'
5

```

Result Grid				
Filter Rows: <input type="text"/>				
Export:  Wrap Cell Content:  Fetch rows:				
tconst	titleType	primaryTitle	category	characters
tt0076851	movie	The Purple Taxi	actor	["Dr. Seamus Scully"]
tt0077536	tvMovie	A Family Upside Down	actor	["Ted Long"]
tt0077898	tvMovie	The Man in the Santa Claus Suit	actor	["Costume Shop Proprietor", "Chauffeur", "Police..."]
tt0047680	movie	Woman's World	actress	["Elizabeth Burns"]
tt0047944	movie	The Cobweb	actress	["Meg Faversen Rinehart"]
tt0049966	movie	Written on the Wind	actress	["Lucy Moore Hadley"]
tt0099846	movie	Innocent Victim	actress	["Marsha Archdale"]
tt0107859	tvMovie	The Portrait	actress	["Fanny Church"]
tt0119418	movie	Le jour et la nuit	actress	["Sonia"]
tt0211719	movie	The Venice Project	actress	["Countess Camilla Volta"]
tt1368858	movie	The Forger	actress	["Anne-Marie"]
tt0047607	movie	Concert of Intrigue	actress	["Anna"]
tt0048321	movie	The Light Across the Street	actress	["Olivia Marceau"]

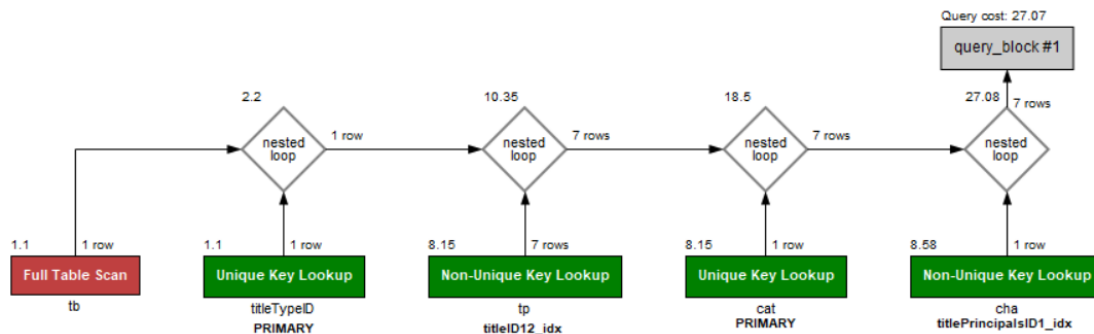
### New query:

```

1 • SELECT tb.titleID, titleTypeID.type, tb.primaryTitle, cat.category, cha.characters
2 FROM titleBasics AS tb
3 LEFT JOIN titleTypeID
4 ON titleTypeID.typeID = tb.typeID
5 LEFT JOIN titlePrincipals AS tp
6 ON tp.titleID = tb.titleID
7 LEFT JOIN titlePrincipalCategoryID AS cat
8 ON cat.ID = tp.categoryID
9 LEFT JOIN titlePrincipalCharacters AS cha
10 ON cha.titlePrincipalsID = tp.titlePrincipalsID
11 WHERE tb.primaryTitle LIKE 'The S%';

```

Result Grid				
Filter Rows: <input type="text"/>				
Export:  Wrap Cell Content:  Fetch rows:				
titleID	type	primaryTitle	category	characters
4570	movie	The Secret of the Mountain	NULL	NULL
6031	movie	The Shame of a Nation	NULL	NULL
6053	movie	The Sioux City Round-Up	NULL	NULL
574	movie	The Story of the Kelly Gang	actor	School Master
574	movie	The Story of the Kelly Gang	actor	Steve Hart
2483	movie	The Sergeant's Boy	actor	NULL
2483	movie	The Sergeant's Boy	actor	NULL
2483	movie	The Sergeant's Boy	actor	NULL
2483	movie	The Sergeant's Boy	actor	The Sergeant...
2483	movie	The Sergeant's Boy	actor	NULL
2483	movie	The Sergeant's Boy	actor	NULL



### Query3: 3.5% lower query cost

SELECT primaryName FROM nameBasics  
WHERE birthyear > 1920 AND primaryName LIKE 'J%';

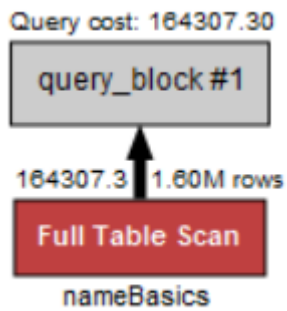
Query result was identical to the old one.

```

1 • SELECT primaryName FROM nameBasics
2   WHERE birthyear > 1920 AND primaryName LIKE 'J%';
3

```

Result Grid		Filter Rows:	Export:	Wrap Cell Co
	primaryName			
▶	John Belushi			
	James Dean			
	Judy Garland			
	Jerry Goldsmith			
	James Horner			
	Jane Russell			
	John Cleese			
	Jennifer Aniston			
	James Cameron			
	John Carpenter			
	Jim Carrey			

**Query4: 3.5% lower query cost**

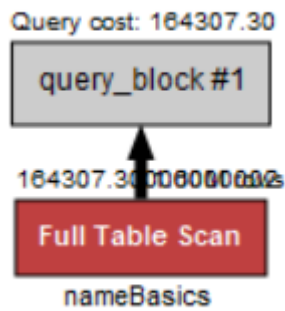
```
SELECT birthYear FROM nameBasics  
WHERE primaryName LIKE 'J%';
```

Query result was identical to the old one.

```
1 • SELECT birthYear FROM nameBasics  
2   WHERE primaryName LIKE 'J%';  
3
```

The screenshot shows a database interface with a 'Result Grid' tab. It includes a 'Filter Rows' input field and an 'Export' button. The table has one column, 'birthYear', and displays a list of years. The first row is highlighted with a blue background.

birthYear
1949
1899
1931
1917
1922
1904
1929
1953
1909
1921
1908



### **Query5: Results in higher query cost by 200%**

```
SELECT tp.titleID, tp.personID, tp.ordering, cat.category, cha.characters, job.job
FROM titlePrincipals AS tp
LEFT JOIN titlePrincipalCategoryID AS cat
ON cat.ID = tp.categoryID
LEFT JOIN titlePrincipalCharacters AS cha
ON cha.titlePrincipalsID = tp.titlePrincipalsID
LEFT JOIN titlePrincipalJobs AS job
ON job.titlePrincipalsID = tp.titlePrincipalsID
WHERE cat.category = 'actor';
```

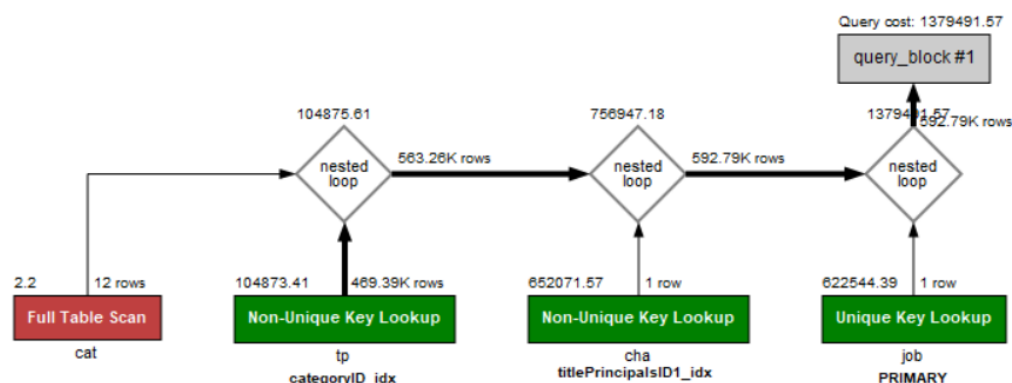
Query result was identical to the old one. The cost however increased by 200% and we couldn't figure out how to solve it.

```

1 • SELECT tp.titleID, tp.personID, tp.ordering, cat.category, cha.characters, job.job
2 FROM titlePrincipals AS tp
3 LEFT JOIN titlePrincipalCategoryID AS cat
4 ON cat.ID = tp.categoryID
5 LEFT JOIN titlePrincipalCharacters AS cha
6 ON cha.titlePrincipalsID = tp.titlePrincipalsID
7 LEFT JOIN titlePrincipalJobs AS job
8 ON job.titlePrincipalsID = tp.titlePrincipalsID
9 WHERE cat.category = 'actor';
10
11

```

titleID	personID	ordering	category	characters	job
25164	1	1	actor	Guy Holden	NULL
26942	1	2	actor	Huck Haines	NULL
27125	1	1	actor	Jerry Travers	NULL
27630	1	1	actor	Bake Baker	NULL
28333	1	1	actor	Lucky Garnett	NULL
28757	1	1	actor	Jerry Halliday	NULL
29546	1	1	actor	Peter P. Peters aka Petrov	NULL
29971	1	1	actor	Tony Flagg	NULL
31983	1	1	actor	Vernon Castle	NULL
32284	1	1	actor	Johnny Brett	NULL
33029	1	1	actor	Danny O'Neill	NULL
34409	1	1	actor	Robert Curtis	NULL
34862	1	2	actor	Ted Hanover	NULL



### Extra tasks:

- View

This view shows all the english movies with their genre and rating.

CREATE

ALGORITHM = UNDEFINED

DEFINER = `a23dasc605`@`%`

SQL SECURITY DEFINER

VIEW `English Movies` AS

SELECT

```

`originalTitles`.`primaryTitle` AS `Title`,
`titleRatings`.`averageRating` AS `Rating`,
`titleGenreID`.`genre` AS `Genre`,
`akaLanguages`.`language` AS `language`
FROM
    (((`akaLanguages`
    LEFT JOIN `originalTitles` ON ((`akaLanguages`.`titleAkasID` =
`originalTitles`.`titleID`)))
    LEFT JOIN `titleRatings` ON ((`akaLanguages`.`titleAkasID` =
`titleRatings`.`titleID`)))
    LEFT JOIN `titleGenres` ON ((`akaLanguages`.`titleAkasID` =
`titleGenres`.`titleID`)))
    LEFT JOIN `titleGenreID` ON ((`titleGenres`.`ID` = `titleGenreID`.`ID`)))
WHERE
    ((`akaLanguages`.`language` = 'EN')
    AND (`originalTitles`.`primaryTitle` IS NOT NULL))

```

	Title	Rating	Genre	language
▶	Hamlet	3	History	EN
	Today and Tomorrow	5	NULL	EN
	Old Brandis' Eyes	NULL	NULL	EN
	The Sparrow	NULL	NULL	EN
	0-18 or A Message from the Sky	7	NULL	EN
	0-18 or A Message from the Sky	7	NULL	EN
	The Naked Truth	4	NULL	EN
	Saints and Sorrows	NULL	NULL	EN
	One Who Is Loved by Two	5	NULL	EN
	Sugar and Spice	6	NULL	EN
	The Strength of the Fatherland	6	NULL	EN
	The Strength of the Fatherland	6	NULL	EN
	Shoe Palace Pinkus	6	NULL	EN
	Victoria	NULL	NULL	EN
	Masked Ball	NULL	NULL	EN
	The Hell Ship	7	NULL	EN
	The Hell Ship	7	NULL	EN
	The Sun of St. Moritz	NULL	NULL	EN
	The Moon of Israel	6	NULL	EN
	Don Quintin the Bitter	NULL	NULL	EN
	Mother	8	NULL	EN
	Women of Ryazan	7	NULL	EN
	The Beggar Student	NULL	NULL	EN
	Art of Love	6	NULL	EN

- Stored procedure

This procedure returns the name and the number of professions of all the persons that have at least 3 professions.

```
CREATE DEFINER=`a23dasc605`@`%` PROCEDURE `3 professions`()
BEGIN
select nameBasics.primaryName as Name, count(nameProfessions.personID) as
nr_of_professions
from nameProfessions
join nameBasics on nameProfessions.personID = nameBasics.personID
group by nameBasics.primaryName
having count(nameProfessions.personID) >2;
END
```

	Name	nr_of_professions
▶	Fred Astaire	3
	Brigitte Bardot	3
	John Belushi	3
	Ingmar Bergman	3
	Ingrid Bergman	3
	Humphrey Bogart	3
	Marlon Brando	3
	Richard Burton	4
	James Cagney	3
	Gary Cooper	10
	Bette Davis	3
	Doris Day	5
	James Dean	13
	Georges Delerue	3
	Marlene Dietrich	3
	Kirk Douglas	9
	Federico Fellini	3
	Henry Fonda	3
	Joan Fontaine	3
	Clark Gable	3
	John Gielgud	3
	Jerry Goldsmith	3
	Cary Grant	6
	Alec Guinness	3

#### - Trigger

This trigger checks the deathYear when it gets updated. If it's not the current or a past year, it will be changed to the current year.



delimiter //

```
CREATE TRIGGER check_death_year BEFORE UPDATE ON nameBasics
FOR EACH ROW
BEGIN
    IF NEW.deathYear > YEAR(CURDATE()) THEN
        SET NEW.deathYear = YEAR(CURDATE());
    END IF;
END; //
```

**Checklist:**

Check	Requirement
✓	All tables have a primary key. See course notes.
✓	Data is correctly imported from .tsv file. See below for instructions.
✓	Database adheres to the first normal form. See below for instructions.
✓	Database adheres to the second normal form. See below for instructions.  Note: not all tables adhere to 2NF.
✓	Database adheres to the third normal form. See below for instructions.
✓	All columns have fitting data types and constraints. Check numbers & dates!
✓	Tables are correctly linked with foreign key constraints.

✓	<b>Referential integrity is guaranteed throughout the database.</b>
✓	<b>Database is not sparse. Null values in itself are not a problem but may be a sign of bad design. Analyse the business meaning of a NULL value. Eg: a person that is still alive has an empty value in the column 'deathYear'. That is no problem.</b>
✓	<b>Database does not contain unnecessary duplicate data.</b>
✓	<b>Correct character sets have been configured.</b>
✓	<b>Create at least one view that is sufficiently complex and insightful.</b>
✓	<b>Create at least one stored procedure that is sufficiently complex and insightful.</b>
✓	<b>Create at least one trigger that is sufficiently complex and insightful.</b>
✓	<b>Optimize the five queries.</b> Not all queries returned identical results to the old ones and one query had a higher cost.
✓	<b>Dive into the data and find some interesting insights.</b>

### Conclusion:

Optimizing such an immense database with so many issues was rather challenging, it was very confusing in the beginning trying to understand what some of the columns and tables were for.

The most challenging part however was trying to make the database as time saving as possible by ultimately using integers in the big central four tables.

Furthermore, towards the end of the project, the server would often get overloaded, it even crashed multiple times just a few days before the deadline.

We think we have made the right decision by working on the database early on during the semester and finishing it beforehand. The extra tasks and the five queries were however left till the end which is why we didn't have enough time to test them correctly and get a perfect optimization.

Making dumps of the database regularly throughout the process helped us easily retrieve old versions of tables whenever something went wrong.

Working with such a database was an interesting experience, especially while creating the view, procedure and trigger. Besides it now being less of a chaos and adhering to different normalization and SQL principals, we think data-wise it still needs a lot of work from the user, by providing correct and sufficient data.

Overall, the project was more time consuming than expected.