Xi'an Jiaotong-Liverpool University
西交利物浦大學

XJTLU Entrepreneur College (Taicang) Cover Sheet

Module code and Title	DTS106TC: Introduction to Database	
School Title	School of AI and Advanced Computing	
Assignment Title	Assessment Task 002 (CW)): Individual Coursework	
Submission Deadline	**May 21st ,2025 at 17:00 PM (GMT +8)**	
Final Word Count	NA	
If you agree to let the university use your work anonymously for teaching and learning purposes, please type **"yes"** here.		**Yes**

I certify that I have read and understood the University's Policy for dealing with Plagiarism, Collusion and the Fabrication of Data (available on Learning Mall Online). With reference to this policy I certify that:

- My work does not contain any instances of plagiarism and/or collusion.
  My work does not contain any fabricated data.

By uploading my assignment onto Learning Mall Online, I formally declare that all of the above information is true to the best of my knowledge and belief.

Scoring – For Tutor Use	
Student ID	

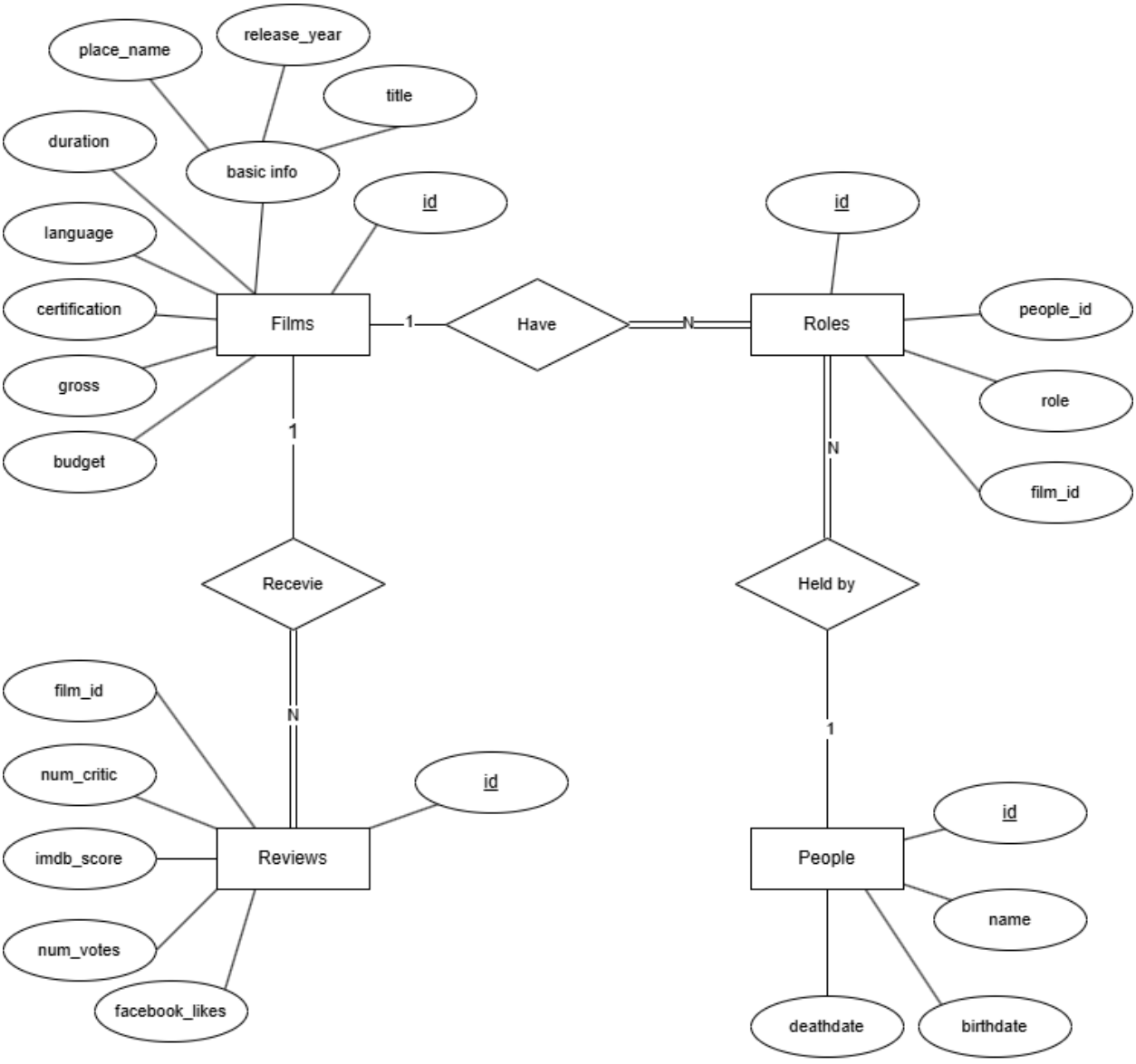
Stage of Marking	**Marker Code**	**Learning Outcomes Achieved (F/P/M/D) (please modify as appropriate)**					**Final Score**
		A	**B**	**C**	**D**	**E**	
1st Marker – red pen							
Moderation – green pen	**IM Initials**	The original mark has been accepted by the moderator (please circle as appropriate):					Y / N
		Data entry and score calculation have been checked by another tutor (please circle):					Y
2nd Marker if needed – green pen							

For Academic Office Use			**Possible Academic Infringement (please tick as appropriate)**	
Date Received	**Days late**	**Late Penalty**	☐ **Category A**	Total Academic Infringement Penalty (A,B, C, D, E, Please modify where necessary) _____
			☐ **Category B**	
			☐ **Category C**	
			☐ **Category D**	
			☐ **Category E**	

Xi'an Jiaotong-Liverpool University
西交利物浦大学

Q1:

Figure1: ERD



Explanation:

As shown in the figure, the four entities have their attributes, relationships. Each review must correspond to exactly one film, and same for the role. One role must be held by exactly one person, while a person can act as different roles. As for the keys, each entity's Primary Key is its id, which should be unique and Not Null. And Review's **film_id,** Role's **film_id** and **people_id** are foreign keys. The **Receive, Held by, Have** relationships are all one-side mandatory. Because I choose the id for 'Roles' as its PK, there is no weak entity in my ERD.

Q2:

Initial analysis:

Films: There are no repeating groups or arrays in attributes (1NF), there is only one PK 'id' in Films, so there is no partial dependency (2NF). I initially suspect that 'language' of films may be influenced by 'place_name', but after searching, I found that multiple languages can be seen for some place (such as Australia), and there are some empty values exist in 'language'. Then I suspect that 'title' of films can determine other attributes if they are unique, but there are different films with same title (such as Out of the Blue). Furthermore, I find that if the combination of 'title', 'release_year', 'place_name' is not unique in each line, they **may** act as a non-primary key to determine other attributes, disobeying 3NF.

Reviews: There are no repeating groups or arrays in attributes (1NF), there is only one PK 'id' in Films, so there is no partial dependency (2NF). I initially suspect that 'film_id', a foreign key, may dominate other non-primary keys. However, I didn't find any situations where the candidate key or combination of candidate keys can determine other non-primary keys. So, there are no transitive dependencies (3NF).

Roles: There are no repeating groups or arrays in attributes (1NF), there is only one PK 'id' in Films, so there is no partial dependency (2NF). I found that there exist different roles of the same 'film_id' and 'people_id' combination (e.x. When film_id == 159, and people_ id == 8297, the role can be both actor and director). So, combination cannot determine other non-primary keys (3NF).

People: There are no repeating groups or arrays in attributes (1NF), there is only one PK 'id' in Films, so there is no partial dependency (2NF). Considering that people's names often repeat in real life, I don't think we should regard 'name' as an attribute that can determine other attributes. Therefore, the PK 'id' determines all non-key attributes, so there are no transitive dependencies (3NF).

Normalization:

Films:

1. The same lines (**all** attributes are the **same**, except for id) are **meaningless** when store the film entry information.
2. In reality, it is almost **impossible** to have a different film with the same title, same release year and same country. But if one of these three elements is missing, the possibility **will increase significantly**.
3. If I split the Films table, I need to create a table with every attribute except for id in the current table, when keeping one table with 'id', 'title', 'release_year', 'place_name', forcing every film lookup to perform **extra joins.**

According to the factors mentioned above, I choose to

1. Edit **meaningless duplicate entries** in the current data (as well as the foreign key in other tables).

2. Add a **UNIQUE** constraint on **{title, release_year, place_name}** to make it become a candidate key when building a database to prevent true duplicates, thus supporting 3NF.

Improvements:

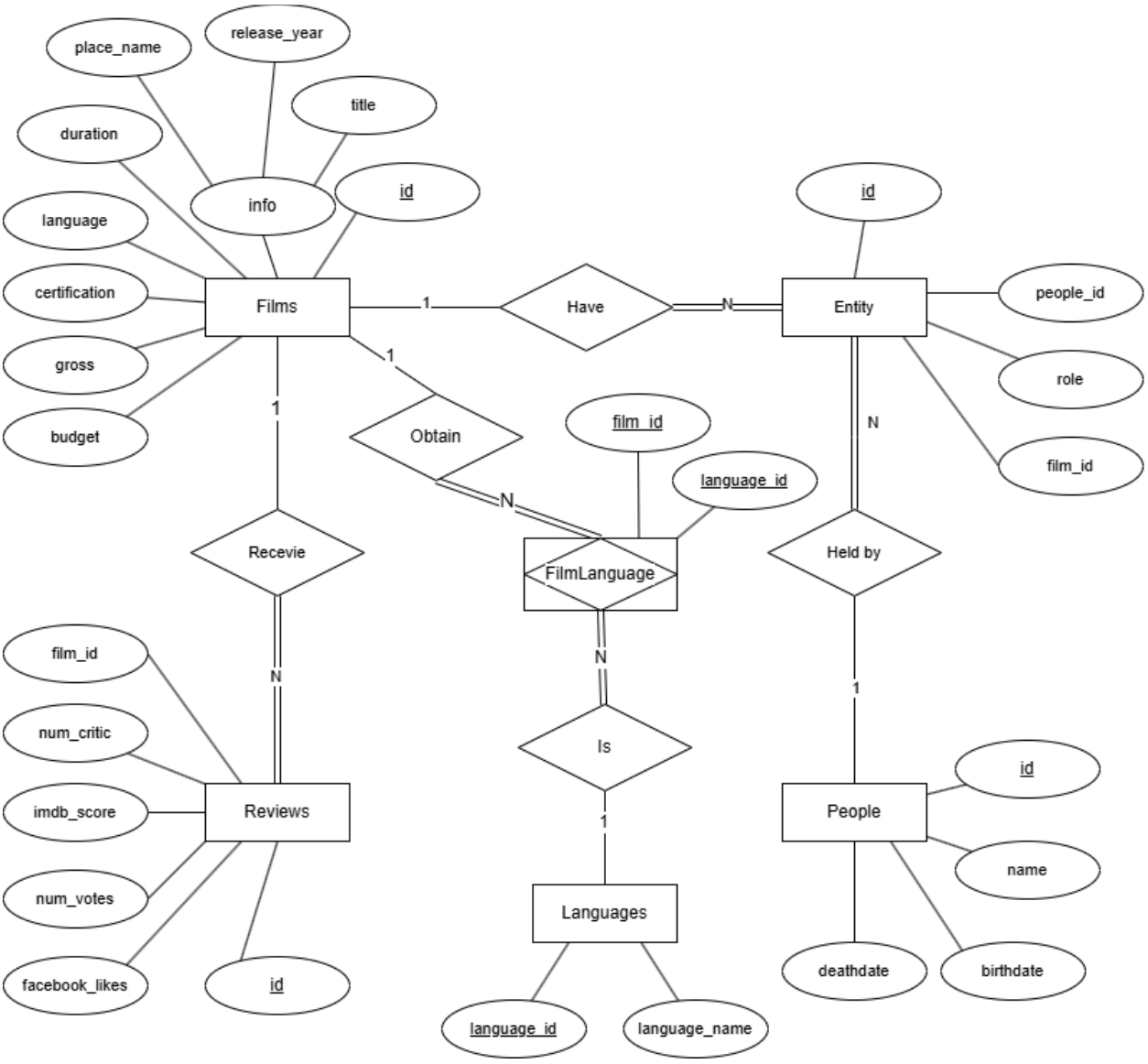
Given that many movies can have different languages when published in different countries, which is not considered in current structure. And when we try to add new language to the current 'film' entity, it will

disobey 1NF. Therefore, I create two new tables: FilmLanguage and Languages. These will handle with the 'many-to-many' relationship between 'Films' and 'Languages'. The detailed description are as follows:

Updated Structures:

Besides the current tables, I will create two new tables: FilmLanguage (intersection entity) and Languages. FilmLanguage have two keys which are both PK and FK: '**film_id**' and '**language_id**'. Languages will have PK '**language_id**' and non-primary attribute '**language_name**'. The new relationships are Films **Obtain** FilmLanguage **(1: N)** and FilmLanguage **Is** Languages **(N:1).** And the **FilmLanguage** → **Films** and **FilmLanguage** → **Languages** are both **one-side mandatory,** while in opposite direction are both optional. I will draw a new ERD to better demonstrate my updated structure.

Figure2: ERD (Updated)



Xi'an Jiaotong-Liverpool University
西交利物浦大學

Q3:

Preface: Although I did create two new tables to maintain normalization regarding the situation that o**ne movie may have several languages.** But given that the dataset only contains **films with only 0/1 language**, and I have made sure that **all four tables are normalized** with the given dataset**.** In the following parts, I will only create four tables in the database.

Table1: FILMS

Column Name	ID	TITLE	RELEASE_ YEAR	PLACE _NAME	DURATI ON	LANGUAGE	CERTIFIC ATION	GROSS	BUDGET
Key Type	PK								
Not Null = NN Unique = U	NN, U	NN	NN	NN					
DATA Type	INTEG AR	VARCHAR	INTEGER	VARCH AR	NUMERI C	VARCHAR	VARCHA R	NUMERIC	NUMERI C
Length	/	200	/	50	(5,1)	30	20	(12,2)	(14,2)

Other conditions: I will enforce a **UNIQUE** constraint on **{title, release_year, place_name}**, declaring a candidate key.

Table2: PEOPLE

Column Name	ID	NAME	BIRTHDATE	DEATHDATE
Key Type	PK			
Not Null = NN Unique = U	NN, U	NN		
DATA Type	INTEG AR	VARCHAR	TEXT	TEXT
Length	/	100	/	/

Explanation: PostgreSQL **doesn't receive the datestyle in the dataset**. When I tried to import the data, it returned WRONG again and again. Therefore, I changed the **DATE** to **TEXT**.

Table3: REVIEWS

Column Name	ID	FILM_ID	NUM_CRITIC	IMDB_SCORE	NUM_VOTES	FACEBOOK_LIKES
Key Type	PK	FK				
Not Null = NN Unique = U	NN, U	NN		NN		
DATA Type	INTEGAR	INTEGER	INTEGER	NUMERIC	INTEGER	INTEGER
Length	/	200	/	(3,1)	/	/

Table4: ROLES

Column Name	ID	FILM_ID	PEOPLE_ID	ROLE
Key Type	PK			
Not Null = NN Unique = U	NN, U	NN	NN	NN
DATA Type	INTEGAR	INTEGER	INTEGER	VARCHAR
Length	/	/	/	20

Image Sharing and Explanation:

1. Films:

Xi'an Jiaotong-Liverpool University
西交利物浦大学

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1 of 5

	id [PK] integer	title character varying (200)	release_year integer	place_name character varying (50)	duration numeric (5,1)	language character varying (30)	certification character varying (20)	gross numeric (12,2)	budget numeric (14,2)
1	1	Intolerance: Love's Struggle Throughout the Ages	1916	USA	123.0	[null]	Not Rated	[null]	385907.00
2	2	Over the Hill to the Poorhouse	1920	USA	110.0	[null]	[null]	3000000.00	100000.00
3	3	The Big Parade	1925	USA	151.0	[null]	Not Rated	[null]	245000.00
4	4	Metropolis	1927	Germany	145.0	German	Not Rated	26435.00	6000000.00
5	5	Pandora's Box	1929	Germany	110.0	German	Not Rated	9950.00	[null]
6	6	The Broadway Melody	1929	USA	100.0	English	Passed	2808000.00	379000.00
7	7	Hell's Angels	1930	USA	96.0	English	Passed	[null]	3950000.00
8	8	A Farewell to Arms	1932	USA	79.0	English	Unrated	[null]	800000.00
9	9	42nd Street	1933	USA	89.0	English	Unrated	2300000.00	439000.00
10	10	She Done Him Wrong	1933	USA	66.0	English	Approved	[null]	200000.00
11	11	It Happened One Night	1934	USA	65.0	English	Unrated	[null]	325000.00
12	12	Top Hat	1935	USA	81.0	English	Approved	3000000.00	609000.00
13	13	Modern Times	1936	USA	87.0	English	G	163245.00	1500000.00
14	14	The Charge of the Light Brigade	1936	USA	100.0	English	Approved	[null]	1200000.00
15	15	Snow White and the Seven Dwarfs	1937	USA	83.0	English	Approved	184925485.00	2000000.00

	A	B	C	D	E	F	G	H	I
1	id	title	release_ye	place_nam	duration	langauge	certificatio	gross	budget
2	1	Intolerance	1916	USA	123		Not Rated		385907
3	2	Over the H	1920	USA	110			3000000	100000
4	3	The Big Pa	1925	USA	151		Not Rated		245000
5	4	Metropolis	1927	Germany	145	German	Not Rated	26435	6000000
6	5	Pandora's	1929	Germany	110	German	Not Rated	9950	
7	6	The Broadv	1929	USA	100	English	Passed	2808000	379000
8	7	Hell's Ange	1930	USA	96	English	Passed		3950000
9	8	A Farewell	1932	USA	79	English	Unrated		800000
10	9	42nd Stree	1933	USA	89	English	Unrated	2300000	439000
11	10	She Done I	1933	USA	66	English	Approved		200000
12	11	It Happene	1934	USA	65	English	Unrated		325000
13	12	Top Hat	1935	USA	81	English	Approved	3000000	609000
14	13	Modern Ti	1936	USA	87	English	G	163245	1500000
15	14	The Charg	1936	USA	100	English	Approved		1200000
16	15	Snow Whit	1937	USA	83	English	Approved	1.85E+08	2000000

Explanation: The result well presents the dataset. Some small differences exist in the **decimal points** under duration, gross, and budget, because I use NUMERIC datatypes to improve scalability. I also converted the release_year values in the dataset from one-decimal-place numbers to **pure integers**, since it's unreasonable for year to be a decimal.

2. People:

Data Output Messages Notifications

	id [PK] integer	name character varying (100)	birthdate text	deathdate text
1	1	50 Cent	7/6/1975	[null]
2	2	A. Michael Baldwin	4/4/1963	[null]
3	3	A. Raven Cruz	[null]	[null]
4	4	A.J. Buckley	2/9/1978	[null]
5	5	A.J. DeLucia	[null]	[null]
6	6	A.J. Langer	5/22/1974	[null]
7	7	Aaliyah	1/16/1979	8/25/2001
8	8	Aaron Ashmore	10/7/1979	[null]
9	9	Aaron Hann	[null]	[null]
10	10	Aaron Hill	4/23/1983	[null]
11	11	Aaron Hughes	[null]	[null]
12	12	Aaron Kwok	10/26/1965	[null]
13	13	Aaron Schneider	[null]	[null]
14	14	Aaron Seltzer	1/12/1974	[null]
15	15	Aaron Stanford	12/27/1976	[null]
16	16	Aaron Staton	[null]	[null]
17	17	Aaron Yoo	5/12/1979	[null]

	A	B	C	D
1	id	name	birthdate	deathdate
2	1	50 Cent	7/6/1975	
3	2	A. Michael Baldwin	4/4/1963	
4	3	A. Raven Cruz		
5	4	A.J. Buckley	2/9/1978	
6	5	A.J. DeLucia		
7	6	A.J. Langer	5/22/1974	
8	7	Aaliyah	1/16/1979	8/25/2001
9	8	Aaron Ashmore	10/7/1979	
10	9	Aaron Hann		
11	10	Aaron Hill	4/23/1983	
12	11	Aaron Hughes		
13	12	Aaron Kwok	10/26/1965	
14	13	Aaron Schneider		
15	14	Aaron Seltzer	1/12/1974	
16	15	Aaron Stanford	12/27/1976	
17	16	Aaron Staton		
18	17	Aaron Yoo	5/12/1979	
19	18	Aasheekaa Bathija		
20	19	Aasif Mandvi	3/5/1966	
21	20	Abbie Cornish	8/7/1982	
22	21	Abby Elliott	6/16/1987	
23	22	Abby Mukiibi Nkaaga		
24	23	Abel Ferrara	7/19/1951	
25	24	Abhishek Bachchan	2/5/1976	
26	25	Abigail Evans		
27	26	Abigail Spencer	8/4/1981	
28	27	Abraham Benrubi	10/4/1969	

Explanation: If I need to filter actors by birthdate/deathdate in the future, I will convert the text format **back to DATE format.**

3. Reviews:

	id [PK] integer	film_id integer	num_critic integer	imdb_score numeric (3,1)	num_votes integer	facebook_likes integer
1	3934	588	432	7.1	203461	46000
2	3405	285	267	6.4	149998	0
3	478	65	29	3.2	8465	491
4	74	83	25	7.6	7071	930
5	1254	1437	224	8.0	241030	13000
6	740	111	64	6.4	64742	0
7	4841	1058	579	8.1	479047	117000
8	2869	59	104	6.8	18442	689
9	3252	117	160	7.2	49855	10000
10	1181	163	99	7.3	16995	0
11	2020	402	145	6.7	91092	0
12	4152	371	359	5.9	108242	39000
13	3220	62	64	6.2	15780	2000
14	2312	251	161	5.8	63912	912
15	1820	113	41	7.0	8535	872
16	718	85	43	6.4	76850	1000
17	831	107	65	6.6	40126	975
18	1231	226	113	6.5	26034	0
19	1746	125	71	6.6	17261	455
20	3508	43	10	5.3	8598	250

	A	B	C	D	E	F
1	id	film_id	num_critic	imdb_scor	num_votes	facebook_likes
2	3934	588	432	7.1	203461	46000
3	3405	285	267	6.4	149998	0
4	478	65	29	3.2	8465	491
5	74	83	25	7.6	7071	930
6	1254	1437	224	8	241030	13000
7	740	111	64	6.4	64742	0
8	4841	1058	579	8.1	479047	117000
9	2869	59	104	6.8	18442	689
10	3252	117	160	7.2	49855	10000
11	1181	163	99	7.3	16995	0
12	2020	402	145	6.7	91092	0
13	4152	371	359	5.9	108242	39000
14	3220	62	64	6.2	15780	2000
15	2312	251	161	5.8	63912	912
16	1820	113	41	7	8535	872
17	718	85	43	6.4	76850	1000
18	831	107	65	6.6	40126	975
19	1231	226	113	6.5	26034	0
20	1746	125	71	6.6	17261	455

Explanation: If I expand the imdb_score column in the CSV, the values show up as seven-decimal-place numbers that are **very close to the original one-decimal values**, so I standardized them to **one decimal place** in the database. I also converted the num_critic values in the dataset from one-decimal-place numbers to **pure integers**, since it's unreasonable for a count of people to be a decimal.

4. Roles:

	id [PK] integer	film_id integer	people_id integer	role character varying (20)
1	1	1	1630	director
2	2	1	4843	actor
3	3	1	5050	actor
4	4	1	8175	actor
5	5	2	3000	director
6	6	2	4019	actor
7	7	2	5274	actor
8	8	2	7449	actor
9	9	3	1459	actor
10	10	3	3929	actor
11	11	3	4581	director
12	12	3	6580	actor
13	13	4	1008	actor
14	14	4	2649	director
15	15	4	2941	actor
16	16	4	6950	actor
17	17	5	2580	actor
18	18	5	2648	actor
19	19	5	2747	director
20	20	5	4958	actor

	A	B	C	D
1	id	film_id	people_id	role
2	1	1	1630	director
3	2	1	4843	actor
4	3	1	5050	actor
5	4	1	8175	actor
6	5	2	3000	director
7	6	2	4019	actor
8	7	2	5274	actor
9	8	2	7449	actor
10	9	3	1459	actor
11	10	3	3929	actor
12	11	3	4581	director
13	12	3	6580	actor
14	13	4	1008	actor
15	14	4	2649	director
16	15	4	2941	actor
17	16	4	6950	actor
18	17	5	2580	actor
19	18	5	2648	actor
20	19	5	2747	director

Explanation: The result well presents the dataset.

Q4:

Query1:

This combines the films and reviews tables using a **JOIN** on f.id = r.film_id; calculates the average IMDb score for each film and rounds it to two decimal places; groups the scores per film using **GROUP BY**; sorts the movies by average_score in descending order via **ORDER BY**; and limits the output to the top five records using **LIMIT 5**.

Data Output:

	movie_title character varying (200)	average_score numeric
1	Bending Steel	9.30
2	The Butterfly Effect	9.20
3	Cliffhanger	9.00
4	Freeheld	9.00
5	Leaving Las Vegas	8.90

Query2:

This combines the films and reviews tables using a **JOIN** on f.id = r.film_id; groups reviews per film using **GROUP BY**; and applies a **HAVING** clause to include only films with more than five reviews.

Data Output (The first several rows):

	movie_title character varying (200)	review_count bigint
1	Raging Bull	8
2	Silent Running	7
3	Everything You Always Wanted to Know About Sex * But Were Afraid to Ask	10
4	Escape from the Planet of the Apes	11
5	The Party's Over	16
6	The Beast from 20,000 Fathoms	16
7	Gone with the Wind	17
8	55 Days at Peking	18
9	She Done Him Wrong	29
10	Show Boat	26
11	Modern Times	19
12	Over the Hill to the Poorhouse	32
13	Live and Let Die	7
14	The Pirate	15
15	A Guy Named Joe	14
16	Escape from Alcatraz	6

Total rows: 284 Query complete 00:00:00.154

Query3:

This combines the roles, films, and people tables using **JOIN**s on r.film_id = f.id and r.people_id = p.id; groups the roles by p.id and p.name using **GROUP BY**; and computes the number of distinct films each actor appears in via **COUNT(DISTINCT r.film_id)**

Data Output (The first several rows):

	actor_name character varying (100)	film_count bigint
1	50 Cent	5
2	A. Michael Baldwin	1
3	A. Raven Cruz	1
4	A.J. Buckley	5
5	A.J. DeLucia	1
6	A.J. Langer	1
7	Aaliyah	2
8	Aaron Ashmore	2
9	Aaron Hann	1
10	Aaron Hill	1
11	Aaron Hughes	1
12	Aaron Kwok	1
13	Aaron Schneider	1
14	Aaron Seltzer	1
15	Aaron Stanford	4
16	Aaron Staton	1

Total rows: 8397 Query complete 00:00:00.174

Xi'an Jiaotong-Liverpool University
西交利物浦大學

Query 1:

SQL Algebras:

Film_Reviews ← Films ⋈ Films.id=Reviews.film_id Reviews

Avg_Scores ← γ film_id,title, avg_score ←γ AVG (imdb_score) (Film_Reviews)

Ordered_Scores ←τ avg_score (Avg_Scores)

Result ← π⁵ title, avg_score (Ordered_Scores)

Explanation:

This combines the Films and Reviews tables with a ⋈ on Films.id = Reviews.film_id; groups the joined rows by film_id and title with **γ**, also using **γ** to calculate each movie's AVG(imdb_score) as avg_score; orders the aggregated result by avg_score in descending order with **τ**; and finally projects just the title and avg_score columns with **π⁵ (restricted projection)** for output.

Query 2:

SQL Algebras:

Film_Reviews ← Films ⋈ Films.id=Reviews.film_id Reviews

Review_Counts ← γ film_id, title, review_cnt ←γ COUNT (review_id) (Film_Reviews)

Filtered ← σ review_cnt > 5 (Review_Counts)

Result ← π title, review_cnt (Filtered)

Explanation:

This combines the Films and Reviews tables via a ⋈ on Films.id = Reviews.film_id; groups the result by film_id and title with **γ**, using **γ** to compute COUNT (review_id) as review_cnt; applies a selection to retain only those groups where review_cnt > 5 with **σ**; and projects the title and review_cnt columns with **π** to produce the final list.

Reflection:

When answering question4, the most difficult part I met was writing the relational algebra. I'm ashamed that I didn't understand it during class. However, I **revised the corresponding content** on week4 slide. Then I found that notation for **sorting** and **"top-k"** are not mentioned in lectures, which are used in my query2. I tried hard to look up the reference e-books on LMO and ask module leader. In the end, I followed module leader's instructions and **applied 'τ A(R)' and restricted projection.**

Q5:

The **conceptual model design** part is my most memorable. Not only because it is the funniest part for me, but also because it **establishes a solid foundation for both the database and further study.** In this part, I acquired the **basic components of a database**: entity, attribute (PK, FK) and relationship. I also mastered **a magic tool for building conceptual models – ERD.** I think a good conceptual model can help me **keep my mind clear in any subsequent conversion and construction.** And I think that a person I met in class helped me a lot in understanding these concepts and their structural relationships.

In the class activity, called 'Match the Data', he firmly believed that his card was exactly the movie my person wanted to find, because the number was the same. I trusted him at the very beginning and waited for the other card. But there was a same person claiming the same movie_id came. I was very confused. But later, I found that the first guy misread his card, and **he has a card with the same number but from a different section.**

I think that was my first time understanding **why the primary keys must be unique and candidate keys must identify a single row**, and **why foreign keys must point to exactly the right table.** In a real database, a single duplicated or misassigned key **can break joins or corrupt results.** It also taught me a lot about cardinality. By **normalizing relationships**, I can find errors that are harder to see.

In one lab class, when my partner and I were doing one estate firm ERD exercise, I **paid more attention to the relationship we construct.** I found that **a single office may list many properties, but each property must belong to exactly one office.** Ensuring that made me feel more relaxed, because it is like double insurance to **prevent misdirecting wrong elements** with the same id. Furthermore, when I am asked to write a query in future class exercise, I always want to **start with a well-structured ERD.** If there is no such diagram, I am happy to draw one myself.

This awareness also helps me when writing the coursework. Firstly, I **emphasize the correct identification** of entities, attributes, and cardinality when designing the conceptual model. Secondly, I clearly **mark the primary key of each entity and design the foreign key names in a way that is easy to identify their target entities.** Thirdly, I tried my best to **mark each mandatory relationship with their cardinality in the ERD**. As a result, the diagram **helps me a lot** in coming physical model design and database construction.

(word count:441)

Q6:

When writing coursework1, I faced some challenges. For example, when I wrote DML to import data into my created database, **one more decimal point could fail the process.** I tried many times to make sure **the data I imported fits perfectly into the pre-set data format.** And it raised my worry about future work on the database. **What if the datatype changes in the future? What if one attribute needs to be stored with various datatypes?** A well-designed database that stores data **is very difficult to modify its existing data structure** or **add new data with unknown datatypes** while ensuring that no data is lost. It is also necessary to **identify every dataset's structure** and **design the exact blank for only one kind of structure-clear data.** Moreover, I also found that when writing SQL to structure and search for data from different tables, it can be **very complicated to write query if the relationships between the tables are complicated itself.** I can imagine that programmers will have a hard time constructing query statements. when relationships are becoming **more and more complex.**

When **update/add/delete operation is frequently required**, **different and uncertain types of data are supposed to be stored together**, and **developers want to adjust the data model at any time** according to business changes **with minimal negative impact**, a different approach to database would be needed.

To deal with the above situations, there are several kinds of NoSQL database:

Firstly, key-value databases are designed to handle data with **simple data models, large scale, and extremely high state changes.** For example, during s**hopping seasons** such as China's Double 11, e-commerce websites have **huge numbers of** customers, and many of them visit the e-commerce platform at the **same time**. The platforms bind the shopping cart content to the user's **USERID key** to **search and locate data**, ensuring that the user's shopping cart content **remains consistent**, and **high-speed READ and WRITE.**

Secondly, document-based databases are designed to store, index and manage **document-oriented data**, and locate documents by key. Developers can **query complex nested relationships with just one statement.** Moreover, in nested relationships, each document can have its own **unique structure.** Take the example of **e-commerce platforms** again. On these platforms, there are different types of products, **whose attributes are also completely different.** For example, the attributes of a laptop include the CPU model, the type and size of the computer graphics card, the size of the computer hard disk, etc., which **can be easily stored in a document**.

Thirdly, graph databases are designed to process data with **large volume and complex data relationships**. For example, **social platforms**, which involve **relationships between users, personal information of each user, and users' use of the network system to conduct many queries.** The amount of data is large, and the relationships are complex. The graph database applies **graph theory to store relationship information** between entities and **uses graph structures for semantic queries**, including nodes, edges, and attributes to **represent and store data.** It can help users easily query the information they need. Since **each node** in the graph database has an **independent attribute list**, developers can also easily adjust the data model according to business changes.

In conclusion, Relational databases, key-value databases, document-based databases, and graph databases each have their own **advantages and disadvantages and cannot replace each other.**

(word count:552)