Question 1:

<u>Check for and clean dirty data</u>: Find out if the film table and the customer table contain any dirty data, specifically non-uniform or duplicate data, or missing values. Next to each query write 2 to 3 sentences explaining how you would clean the data (even if the data is not dirty).

<u>Table: film</u>

(A) Data Accuracy

- Check whether the data is correct or not using MAX, MIN, and AVG for each column (applicable to quantitative data: release_year, rental_duration, rental_rate, length, replacement_cost).

```
SELECT MAX(release_year) AS max_rel_year,
       MIN(release_year) AS min_rel_year
FROM film
```

max_rel_year integer	min_rel_year integer
2006	2006

```
SELECT MAX(rental_duration) AS max_rental_duration,
       MIN(rental_duration) AS min_rental_duration,
       AVG(rental_duration) AS avg_rental_duration
FROM film
```

max_rental_duration smallint	min_rental_duration smallint	avg_rental_duration numeric
7	3	4.9850000000000000

```
SELECT MAX(rental_rate) AS max_rental_rate,
       MIN(rental_rate) AS min_rental_rate,
       AVG(rental_rate) AS avg_rental_rate
FROM film
```

max_rental_rate numeric	min_rental_rate numeric	avg_rental_rate numeric
4.99	0.99	2.9800000000000000

```
SELECT MAX(length) AS max_length,
       MIN(length) AS min_length,
       AVG(length) AS avg_length
FROM film
```

max_length smallint	min_length smallint	avg_length numeric
185	46	115.2720000000000000

```
SELECT MAX(replacement_cost) AS max_replacement_cost,
       MIN(replacement_cost) AS min_replacement_cost,
       AVG(replacement_cost) AS avg_replacement_cost
FROM film
```

max_replacement_cost numeric	min_replacement_cost numeric	avg_replacement_cost numeric
29.99	9.99	19.9840000000000000

- ➢ No issue was found in data accuracy. All the quantitative data is within a reasonable range.
- ➢ If there are any incorrect data, we have to refer to the source owner or another source recording the same data to find out the actual value. Sometimes we might be able to know the correct value by checking the other values of that column. For instance, in a year column with values of 2010, 2011, 2012, 20133, 2014 and 2015, we know that the year 20133 is incorrect and it should be 2013. After knowing the correct value, we could correct the data using the UPDATE command.

(B) Data Consistency

- Use DISTINCT to check if there are any format/spelling/naming issues in the values of each column.

```
SELECT DISTINCT film_id
FROM film
ORDER BY film_id ASC
```

film_id [PK] integer
1
2
3
4
5
6

```
SELECT DISTINCT title
FROM film
ORDER BY title ASC
```

title character varying (255)
Academy Dinosaur
Ace Goldfinger
Adaptation Holes
Affair Prejudice
African Egg
Agent Truman

```
SELECT DISTINCT release_year
FROM film
ORDER BY release_year ASC
```

release_year integer
2006

```
SELECT DISTINCT language_id
FROM film
ORDER BY language_id ASC
```

language_id smallint
1

```
SELECT DISTINCT rental_duration
FROM film
ORDER BY rental_duration ASC
```

rental_duration smallint
3
4
5
6
7

```
SELECT DISTINCT rental_rate
FROM film
ORDER BY rental_rate ASC
```

rental_rate numeric (4,2)
0.99
2.99
4.99

```
SELECT DISTINCT length
FROM film
ORDER BY length ASC
```

length smallint
46
47
48
49
50
51

```
SELECT DISTINCT replacement_cost
FROM film
ORDER BY replacement_cost ASC
```

replacement_cost numeric (5,2)
9.99
10.99
11.99
12.99
13.99
14.99

```
SELECT DISTINCT rating
FROM film
ORDER BY rating ASC
```

rating mpaa_rating
G
PG
PG-13
R
NC-17

- No issue was found in data consistency. All the columns have a consistent format with their values.
- If there are any format inconsistencies in the values, we could use the UPDATE command to correct the wrong format.

(C) Data Completeness

- Use DISTINCT to check if there are any values with null, 'NA', 'missing', 'unknown', or dummy values in each column (The queries are the same as those in the data consistency section, so I'm not going to paste the queries here again).
- Use COUNT to compare the number of values in each column to the total number of rows in the table.

```
SELECT COUNT(*) AS count_table_rows,
       COUNT(film_id) AS count_film_id,
       COUNT(title) AS count_title,
       COUNT(description) AS count_description,
       COUNT(release_year) AS count_rel_year,
       COUNT(language_id) AS count_language_id,
       COUNT(rental_duration) AS count_rental_duration,
       COUNT(rental_rate) AS count_rental_rate,
       COUNT(length) AS count_length,
       COUNT(replacement_cost) AS count_replacement_cost,
       COUNT(rating) AS count_rating,
       COUNT(special_features) AS count_special_features
FROM film
```

count_table_rows bigint	count_film_id bigint	count_title bigint	count_description bigint	count_rel_year bigint	count_language_id bigint	count_rental_duration bigint	count_rental_rate bigint	count_length bigint
1000	1000	1000	1000	1000	1000	1000	1000	1000

count_replacement_cost bigint	count_rating bigint	count_special_features bigint
1000	1000	1000

- There is no null, dummy or missing value found in all the columns. The number of values of each column also tally with the total number of rows in the film table.
- If there is a column with a high percentage of missing values, we could simply omit that particular column in our SELECT command and leave a comment aside. If the missing values are less than 5% of the column, we could impute the missing values with estimates, say average, using the UPDATE command.

(D) Data Uniqueness

- Define the data grain of the table first. Then SELECT the data grain columns and COUNT all the rows. Show the rows that have more than one appearance.
- Data grain: title, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, rating

```
SELECT title,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       COUNT(*)
FROM film
GROUP BY title,
         release_year,
         language_id,
         rental_duration,
         rental_rate,
         length,
         replacement_cost,
         rating
HAVING COUNT(*) > 1     --no result set means we have no duplicates
```

title character varying (255)	release_year integer	language_id smallint	rental_duration smallint	rental_rate numeric (4,2)	length smallint	replacement_cost numeric (5,2)	rating mpaa_rating	count bigint

- From the result, there are no duplicates in the film table.
- If there are any duplicates, we could create a virtual table, known as a “view,” where we select only unique records using the DISTINCT or GROUP BY clause.
- We could also delete the duplicates from the table using the DELETE command only if we have permission to do so.

Table: customer

(A) Data Accuracy

- Check whether the data is correct or not using MAX, MIN, and AVG for each column (applicable to quantitative data: create_date).

```
SELECT MAX(create_date) AS max_create_date,
       MIN(create_date) AS min_create_date
FROM customer
```

max_create_date date	min_create_date date
2006-02-14	2006-02-14

➢ No issue was found in data accuracy. All the quantitative data is within a reasonable range.

(B) Data Consistency

- Use DISTINCT to check if there are any format/spelling/naming issues in the values of each column.

```
SELECT DISTINCT customer_id
FROM customer
ORDER BY customer_id ASC
```

customer_id [PK] integer
1
2
3
4
5

```
SELECT DISTINCT store_id
FROM customer
ORDER BY store_id ASC
```

store_id smallint
1
2

```
SELECT DISTINCT first_name
FROM customer
ORDER BY first_name ASC
```

first_name character varying (45)
Aaron
Adam
Adrian
Agnes
Alan
Albert

```
SELECT DISTINCT last_name
FROM customer
ORDER BY last_name ASC
```

last_name character varying (45)
Abney
Adam
Adams
Alexander
Allard

```
SELECT DISTINCT email
FROM customer
ORDER BY email ASC
```

email character varying (50)
aaron.selby@sakilacustomer.org
adam.gooch@sakilacustomer.org
adrian.clary@sakilacustomer.org
agnes.bishop@sakilacustomer.org
alan.kahn@sakilacustomer.org
albert.crouse@sakilacustomer.org

```
SELECT DISTINCT address_id
FROM customer
ORDER BY address_id ASC
```

address_id smallint
5
6
7
8
9
10

```
SELECT DISTINCT activebool
FROM customer
ORDER BY activebool ASC
```

activebool boolean
true

```
SELECT DISTINCT create_date
FROM customer
ORDER BY create_date ASC
```

create_date date
2006-02-14

```
SELECT DISTINCT active
FROM customer
ORDER BY active ASC
```

active integer
0
1

➢ No issue was found in data consistency. All the columns have a consistent format with their values.

(C) Data Completeness

- Use DISTINCT to check if there are any values with null, 'NA', 'missing', 'unknown', or dummy values in each column (The queries are the same as those in the data consistency section, so I'm not going to paste the queries here again).
- Use COUNT to compare the number of values in each column to the total number of rows in the table.

```
SELECT COUNT(*) AS count_table_rows,
       COUNT(customer_id) AS count_customer_id,
       COUNT(store_id) AS count_store_id,
       COUNT(first_name) AS count_first_name,
       COUNT(last_name) AS count_last_name,
       COUNT(email) AS count_email,
       COUNT(address_id) AS count_address_id,
       COUNT(activebool) AS count_activebool,
       COUNT(create_date) AS count_create_date,
       COUNT(active) AS count_active
FROM customer
```

count_table_rows bigint	count_customer_id bigint	count_store_id bigint	count_first_name bigint	count_last_name bigint	count_email bigint	count_address_id bigint	count_activebool bigint
599	599	599	599	599	599	599	599

count_create_date bigint	count_active bigint
599	599

➢ There is no null, dummy or missing value found in all the columns. The number of values of each column also tally with the total number of rows in the customer table.

(D) Data Uniqueness

- Define the data grain of the table first. Then SELECT the data grain columns and COUNT all the rows. Show the rows that have more than one appearance.
- Data grain: store_id, first_name, last_name, email, address_id, activebool, create_date, active

```
SELECT store_id,
       first_name,
       last_name,
       email,
       address_id,
       activebool,
       create_date,
       active,
       COUNT(*)
FROM customer
GROUP BY store_id,
         first_name,
         last_name,
         email,
         address_id,
         activebool,
         create_date,
         active
HAVING COUNT(*) > 1     --no result set means we have no duplicates
```

store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date	active integer	count bigint

➢ From the result, there are no duplicates in the customer table.