Business Report for Fantastic DVD Rental Store

1. Summarize one real-world written business report that can be created from the DVD Dataset:

A picture containing indoor, shelf, retail, convenience store

Description automatically generated

A real-world business inquiry for stores such as Blockbuster video would have been to “Identify the top 10 most rented DVDs in each genre”. Inventory must be maintained for each physical copy of a DVD and as demand for some DVDs begins to lag, we can use this report to replace titles with those that are currently in demand in each genre, display them in-store—as per regular business practice pictured above—and charge a premium for them as these are coveted titles.

1. Identify the specific fields that will be included in the detailed table and the summary table of the report.
   1. The detail table shall include the following fields.

1. inventory\_id: integer,

2. rental\_id: integer,

3. film\_id: integer,

4. title: text,

5. rental\_rate: numeric,

6. premium\_rate: numeric,

7. month\_year: timestamp,

8. category\_id: integer,

9. category\_name: text,

b. The summary table shall include the following fields.

1. rank\_id: integer,

2. category\_name: text,

3. movie\_title: text,

4. new\_rental\_rate: numeric

c. A Data dictionary for the previous fields can be found below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Field type** | **From table** | **To Table** | **Description** |
| Inventory\_id | Integer | Inventory | detail\_table | key identifying a films inventory number |
| Rental\_id | Integer | Rental | Detail\_table | Key identifying a rental by a customer |
| Film\_id | Integer | Film | Detail\_table | Key identifying a film’s numeric code tied to a movie title. |
| Title | Text | Film | Detail\_table | Title of film |
| Rental\_rate | Numeric | Rental | Detail\_table | Standard rental amount for the film |
| Premium\_rate | Numeric | Transform from rental\_rate field on rental table | Detail\_table | Premium rate for movies if they rank as a top 10 candidate. |
| Month\_year | Text | Transform from rental\_date on rental table | Detail\_table | Human readable time—month and year of the time the movie was rented. |
| Category\_id | Integer | Category | Detail\_table | Key identifying a category for a genre. |
| Category\_name | Text | Category | Detail\_table / Summary\_table | The name of the genre |
| Rank\_id | Integer | Calculated on the fly | Summary\_table | Key identifying the placement for the movie’s rank. This can be a integer value from 1-10 |
| Movie\_title | Text | Detail – renamed from title | Summary\_table | The title of the movie. |
| New\_rental\_rate | Numeric | Detail – renamed from premium\_rate | Summary\_table | The designated premium rate to charge at checkout for a movie classified as “top 10” |

3.  Identify at least **two** specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report.

The following tables will be used to extract information to create the detail table:

1. inventory -- movies on hand

2. film -- film information including title and rating

3. category -- category data such as the name of the category

4. rental -- the rental dates for each movie

5. film\_category -- the category for each movie.

4.  Identify at least **one** field in the detailed table section that will require a custom transformation with a user-defined function and explain why it should be transformed (e.g., you might translate a field with a value of N to No and Y to Yes).

the following fields in the detail table will require a custom transformation

1. rental\_date: timestamp – The month and year will need to be extracted to be more a human-readable timestamp.
2. Rental\_rate: numeric. – The rate will be adjusted to add a premium charge.
3. Explain the different business uses of the detailed table section and the summary table section of the report.

The detail table will provide an overview of all movies that have been rented across all stores, the rates being charged, and their rental dates and count number. Business users will be able to use this data to create summary reports that tabulate the top movies that are being rented by category as well as rank and apply a premium to those movies that are designated as top 10 to increase revenue. This detailed report will also allow business users to look at movie inventory items that are not being rented often and begin a process for reducing the price of the item or liquidating the item.

1. Explain how frequently your report should be refreshed to remain relevant to stakeholders.

This report should be re-run at least once a week after the working hours of all stores or once a night. This way, we create a historical track per week/day of what the top 10 movies per genre are. A business case that would exemplify its use is the following one:

*Fantastic DVD Rental Store sets up new product placement and top 10 movies to display on Friday morning. For the store to prepare for this, they run a report that compiles a list of the top 10 movies per genre a day or two days prior. With the information in hand, they assess inventory levels, update standard rates to premium rates for the top 10 and create product placement assets to be printed and displayed at the store. This report is also fed into their P.O.S., which adjusts the normal rental rate to a premium rate when a top-10 movie is scanned.*

1. Provide original code for function(s) in text format that performs the transformation(s) you identified in part A4.
2. Transform function to extract month and year from rental\_date timestamp:

1. CREATE OR REPLACE FUNCTION extract\_month\_year(rental\_date timestamp)

2. RETURNS text AS $$

3. BEGIN

4. RETURN to\_char(rental\_date, 'MM/YYYY');

5. END;

6. $$ LANGUAGE plpgsql;

1. Transform function to add a premium to rate for all movies:

1. CREATE OR REPLACE FUNCTION add\_premium\_to\_rental\_rate(rental\_rate numeric)

2. RETURNS numeric AS $$

3. DECLARE

4. new\_rental\_rate numeric;

5. premium numeric;

6. BEGIN

7. premium := 0.5;

8. new\_rental\_rate := rental\_rate + premium;

9. RETURN new\_rental\_rate;

10. END;

11. $$ LANGUAGE plpgsql;

C.  Provide original SQL code in a text format that creates the detailed and summary tables to hold your report table sections.

1. Create script for detail\_table:

1. CREATE TABLE IF NOT EXISTS detail\_table (

2. detail\_id integer GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

3. inventory\_id integer NOT NULL,

4. film\_id integer NOT NULL,

5. rental\_id integer NOT NULL,

6. title text NOT NULL,

7. rental\_rate numeric NOT NULL,

8. premium\_rate numeric NOT NULL,

9. month\_year text NOT NULL,

10. category\_id integer NOT NULL,

11. category\_name text NOT NULL

12. );

2. Create for summary\_table:

1. CREATE TABLE IF NOT EXISTS summary\_table (

2. id integer GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

3. rank\_id integer NOT NULL,

4. category\_name text NOT NULL,

5. movie\_title text NOT NULL,

6. new\_rental\_rate numeric NOT NULL

7. );

D.  Provide an original SQL query in a text format that will extract the raw data needed for the detailed section of your report from the source database.

1. Populate script for detail\_table:

-- This script assumes that the detai\_table has already been created

1. INSERT INTO detail\_table (

2. inventory\_id,

3. film\_id,

4. rental\_id,

5. title,

6. rental\_rate,

7. premium\_rate,

8. month\_year,

9. category\_id,

10. category\_name

11. )

12. SELECT

13. INV.inventory\_id,

14. FI.film\_id,

15. RE.rental\_id,

16. FI.title,

17. FI.rental\_rate,

18. add\_premium\_to\_rental\_rate(FI.rental\_rate) as premium\_rate,

19. extract\_month\_year(RE.rental\_date) as month\_year,

20. CA.category\_id,

21. CA.name AS category\_name

22. FROM Film AS FI

23. INNER JOIN Inventory AS INV

24. ON INV.film\_id = FI.film\_id

25. INNER JOIN rental AS RE

26. ON RE.inventory\_id = INV.inventory\_id

27. INNER JOIN film\_category AS FI\_CA

28. ON FI\_CA.film\_id = FI.film\_id

29. INNER JOIN category AS CA

30. ON CA.category\_id = FI\_CA.category\_id

31.

1. Populate script for summary\_table:

1. – This script assumes that summary\_table has already been created.

2. INSERT INTO summary\_table (

3. rank\_id,

4. category\_name,

5. movie\_title,

6. new\_rental\_rate

7. )

8. SELECT

9. row\_num AS rank\_id,

10. category\_name,

11. title AS movie\_title,

12. premium\_rate AS new\_rental\_rate

13. FROM (

14. SELECT

15. COUNT(rental\_id) AS number\_of\_rentals,

16. title,

17. premium\_rate,

18. month\_year,

19. category\_name,

20. ROW\_NUMBER()

21. OVER (

22. PARTITION BY category\_name ORDER BY COUNT(rental\_id) DESC

23. ) AS row\_num

24. FROM detail\_table

25. GROUP BY title, premium\_rate, month\_year, category\_name

26. ) AS subquery

27. WHERE row\_num <= 10;

28.

E.  Provide original SQL code in a text format that creates a trigger on the detailed table of the report that will continually update the summary table as data is added to the detailed table.

1. -- Create the trigger function

2. CREATE OR REPLACE FUNCTION update\_summary\_table()

3. RETURNS TRIGGER AS $$

4. BEGIN

5. -- Delete existing records for the affected category

6. DELETE FROM summary\_table

7. WHERE category\_name = NEW.category\_name;

8.

9. -- Insert new records for the affected category

10. INSERT INTO summary\_table (rank\_id, category\_name, movie\_title, new\_rental\_rate)

11. SELECT row\_num AS rank\_id, category\_name, title AS movie\_title, premium\_rate AS new\_rental\_rate

12. FROM (

13. SELECT

14. COUNT(\*) AS number\_of\_rentals,

15. title,

16. premium\_rate,

17. category\_name,

18. ROW\_NUMBER()

19. OVER (

20. PARTITION BY category\_name ORDER BY COUNT(\*) DESC

21. ) AS row\_num

22. FROM detail\_table

23. WHERE category\_name = NEW.category\_name

24. GROUP BY title, premium\_rate, category\_name

25. ) AS subquery

26. WHERE row\_num <= 10;

27.

28. RETURN NEW;

29. END;

30. $$ LANGUAGE plpgsql;

31.

32. -- Create the trigger

33. DROP TRIGGER update\_summary\_trigger ON detail\_table;

34. CREATE TRIGGER update\_summary\_trigger

35. AFTER INSERT ON detail\_table

36. FOR EACH ROW

37. EXECUTE FUNCTION update\_summary\_table();

38.

F.  Provide an original stored procedure in a text format that can be used to refresh the data in *both* the detailed table and summary table. The procedure should clear the contents of the detailed table and summary table and perform the raw data extraction from part D.

1. -- Create the stored procedure

2. CREATE OR REPLACE PROCEDURE clear\_and\_repopulate\_detail\_and\_summary\_table()

3. LANGUAGE plpgsql

4. AS $$

5. BEGIN

6. -- Clear the detail\_table

7. DELETE FROM detail\_table;

8. -- Clear the summary\_table

9. DELETE FROM summary\_table;

10.

11. -- Repopulate the detail\_table

12. INSERT INTO detail\_table (

13. inventory\_id,

14. film\_id,

15. rental\_id,

16. title,

17. rental\_rate,

18. premium\_rate,

19. month\_year,

20. category\_id,

21. category\_name

22. )

23. SELECT

24. INV.inventory\_id,

25. FI.film\_id,

26. RE.rental\_id,

27. FI.title,

28. FI.rental\_rate,

29. add\_premium\_to\_rental\_rate(FI.rental\_rate) as premium\_rate,

30. extract\_month\_year(RE.rental\_date) as month\_year,

31. CA.category\_id,

32. CA.name AS category\_name

33. FROM Film AS FI

34. INNER JOIN Inventory AS INV

35. ON INV.film\_id = FI.film\_id

36. INNER JOIN rental AS RE

37. ON RE.inventory\_id = INV.inventory\_id

38. INNER JOIN film\_category AS FI\_CA

39. ON FI\_CA.film\_id = FI.film\_id

40. INNER JOIN category AS CA

41. ON CA.category\_id = FI\_CA.category\_id;

42.

43. -- We have a trigger on the detail\_table that will repopulate the summary\_table

44. END;

45. $$;

46.

1.  Identify a relevant job scheduling tool that can be used to automate the stored procedure.

Different scheduling software can be used to automate processes / stored procedures.

Depending on the size of the company or budget for the organization, you should adapt accordingly. If a single individual uses a stored procedure for a personal project/venture and most code runs locally, you could use the local cron job scheduler with your operating system. You’ll simply apply the cron schedule to execute a local file at a particular cadence.

As the organization scales, you could use Jenkins or Apache Airflow. These two pieces of software provide features for managing and orchestrating complex data pipelines and tasks. They support various types of scheduling, including time-based scheduling, interval-based scheduling, and dependency-based scheduling but come at a cost for usage.

H.  Acknowledge all utilized sources, including any sources of third-party code, using in-text citations and references. If no sources are used, clearly declare that no sources were used to support your submission.

No sources were utilized to complete this project as this was completed with general knowledge and experience gained in my current area of expertise at work.