# D-191 Business Case Document

### Data Used in Report

The data used in this report will be used to highlight top rental revenue for a DVD company. In particular, the report will include data points for a DVD’s title, inventory\_id, release\_year, rental\_duration, rental\_rate, length, replacement\_cost, rating, times\_rented, and total. To aggregate this data, the report will pull in data from the payment, rental, inventory, and film tables from the public schema. Once this report is generated, then users will be able to visualize which DVDs bring in the most revenue and how often they were rented. This is helpful information that can be used to make business decisions like ordering more copies of DVDs that are known to drive revenue.

***Note:*** *the SQL examples in this document are excerpts from files within the same folder as this document divide into sub-folders named functions, queries, stored\_procedures, and triggers accordingly. Please see files named with each example for more in-depth SQL and accompanying comments.*

### Tables Used in Report

To populate the data within the report’s Summary and Detailed sections the following tables are used:

* public.payment
* public.rental
* public.inventory
* public.film

### Data Fields Included in Report

* idis of type PRIMARY KEY which means that the field is unique to each row of the table to avoid having any duplicate entries.
* Title which is of type TEXT which means this field houses text data.
* inventory\_id is of type INT which means this field is an integer.
* release\_year is of type YEAR which means this field is a datetime value.
* rental\_duration is of type INT which means this field is an integer.
* rental\_rate is of type INT which means this field is an integer.
* Length is of type DECIMAL which means this field is a decimal.
* replacement\_cost is of INT which means this field is an integer.
* Rating is of type MPAA\_RATING which means this field is a custom enum type used to hold a rating value specified by the MPAA standards.
* times\_rented is of type INT which means this field is an integer.
* Total is of type DECIMAL which means this field is a decimal.

### Data Transformation in Report

Within this report the length field is initially displayed as total minutes of the DVD. Most users will be used to seeing the length in hours, not minutes, so this field will have a conversion performed on it to convert the value for length from minutes into hours with a precision of 1.

### Report Uses

This report has the primary objective of aggregating the top ten revenue generating DVD rentals and detailed information about those rentals. This data is valuable and can be used for discerning things such as what DVDs should have backup copies on hand in case something bad were to happen to a copy that’s being rented frequently, which DVDs potentially bring in the most profit (which requires additional to successfully prove of course), how often higher revenue DVDs are rented, and plenty of other points of interest. A data team could easily take this report and highlight points that are most critical to the business and current endeavors.

In addition to the previously mentioned, the ancillary data can also be used for making inferences upon why these specific DVDs are the top revenue providers. These inferences can then be used to make decisions such as seeking and purchasing more DVDs that are likely to become revenue generators. For example, the generated report shows all the DVDs were released in 2006, the majority are rated as PG-13, and the length is either one or two hours long. Based on this report, the company knows to look for DVDs that came out around 2006, are rated as PG or PG-13, and are no longer than two hours. Depending on the priorities, this data can influence decisions

### Report Run Frequency

This report will be most effective for users when its data is refreshed on a weekly basis. The reason for this is that the data will be frequently updated in a reasonable amount of time to be accurate with the rapidly changing nature of rentals. Most rentals will be completed from rental date to return within a few days, so having a weekly update will allow for noticing trends and behaviors more easily over the course of the year.

### Table Creation SQL

File name: create\_report\_tables.sql

-- Create the report schema

CREATE SCHEMA IF NOT EXISTS report;

-- Create the summary table to house our report data

CREATE TABLE IF NOT EXISTS report.top\_ten\_rentals\_summary (

   id                SERIAL PRIMARY KEY  NOT NULL,

   title             TEXT NOT NULL,

   inventory\_id      INT  NOT NULL,

   times\_rented      INT  NOT NULL,

   total             DECIMAL  NOT NULL

);

-- Create the details table to house our report data

CREATE TABLE IF NOT EXISTS report.top\_ten\_rentals\_details (

   id               SERIAL PRIMARY KEY NOT NULL,

   title            TEXT NOT NULL,

   inventory\_id     INT NOT NULL,

   release\_year     YEAR NOT NULL,

   rental\_duration  INT,

   rental\_rate      INT,

   length           DECIMAL,

   replacement\_cost INT,

   rating           MPAA\_RATING NOT NULL,

   times\_rented     INT NOT NULL,

   total            DECIMAL NOT NULL

)

### Report Details Data Extraction Query

File name: get\_details.sql

SELECT f.title,

        r.inventory\_id,

        f.release\_year,

        f.rental\_duration,

        f.rental\_rate,

        fn\_minutes\_to\_hours(f.length) AS hours\_long,

        f.replacement\_cost,

        f.rating,

        COUNT(\*) AS times\_rented,

        SUM(p.amount) AS total

FROM public.payment AS p

    JOIN public.rental AS r ON p.rental\_id = r.rental\_id

    JOIN public.inventory AS i ON r.inventory\_id = i.inventory\_id

    JOIN public.film AS f ON i.film\_id = f.film\_id

GROUP BY f.title,

            r.inventory\_id,

            f.release\_year,

            f.rental\_duration,

            f.rental\_rate,

            f.length,

            f.replacement\_cost,

            f.rating

ORDER BY total DESC

LIMIT 10;

### Function for Data Transformation

File name: fn\_minutes\_to\_hours.sql

CREATE OR REPLACE FUNCTION fn\_minutes\_to\_hours(lengthInMinutes INTEGER) RETURNS DECIMAL AS

$$

BEGIN

    RETURN ROUND(lengthInMinutes::DECIMAL / 60.0, 1);

END;

$$

LANGUAGE plpgsql;

### Trigger SQL

File name: tr\_populate\_summary.sql

-- function to run when the trigger is hit

CREATE OR REPLACE FUNCTION fn\_summary\_trigger() RETURNS TRIGGER AS $$

   BEGIN

      CALL sp\_get\_details();

      RETURN NULL;

   END;$$

LANGUAGE plpgsql;

-- we want to drop the trigger if it exists already

DROP TRIGGER tr\_update\_summary ON report.top\_ten\_rentals\_details;

CREATE TRIGGER tr\_update\_summary

   AFTER INSERT ON report.top\_ten\_rentals\_details

   FOR EACH ROW

      EXECUTE FUNCTION public.fn\_summary\_trigger();

### Report Population Stored Procedures

The below stored procedure (first one) can be run weekly via a scheduled task with the Windows Task Scheduler tool. This is to refresh the data weekly, as that is the suggested frequency to run at. When this procedure is run, it triggers the summary population trigger that runs the second stored procedure from below. Thus, both procedures clear out the existing report data to repopulate with more accurate data.

### Populate Detail Report Table Procedure

File name: sp\_get\_details.sql

CREATE OR REPLACE PROCEDURE sp\_get\_details()

AS

$$

-- Start a tansaction to get the data

BEGIN

    -- clear out existing data to refresh list

    DELETE FROM report.top\_ten\_rentals\_details;

    INSERT INTO report.top\_ten\_rentals\_details (title,

                                                inventory\_id,

                                                release\_year,

                                                rental\_duration,

                                                rental\_rate,

                                                length,

                                                replacement\_cost,

                                                rating,

                                                times\_rented,

                                                total)

        SELECT f.title,

                r.inventory\_id,

                f.release\_year,

                f.rental\_duration,

                f.rental\_rate,

                fn\_minutes\_to\_hours(f.length) AS hours\_long,

                f.replacement\_cost,

                f.rating,

                COUNT(\*) AS times\_rented,

                SUM(p.amount) AS total

        FROM public.payment AS p

            JOIN public.rental AS r ON p.rental\_id = r.rental\_id

            JOIN public.inventory AS i ON r.inventory\_id = i.inventory\_id

            JOIN public.film AS f ON i.film\_id = f.film\_id

        GROUP BY f.title,

                    r.inventory\_id,

                    f.release\_year,

                    f.rental\_duration,

                    f.rental\_rate,

                    f.length,

                    f.replacement\_cost,

                    f.rating

        ORDER BY total DESC

        LIMIT 10;

    -- Rollback when there is an exception to perserve data integrity

    EXCEPTION

        WHEN OTHERS THEN

            ROLLBACK;

-- Commit the data

END;

$$

LANGUAGE plpgsql;

### Populate Summary Table Procedure

File name: sp\_get\_summary.sql

CREATE OR REPLACE PROCEDURE sp\_get\_summary()

AS

$$

-- Start a tansaction to get the data

BEGIN

    -- clear out existing data to refresh list

    DELETE FROM report.top\_ten\_rentals\_summary;

    INSERT INTO report.top\_ten\_rentals\_summary (title, inventory\_id, times\_rented, total)

    SELECT f.title AS title,

           r.inventory\_id AS inventory\_id,

           COUNT(\*) AS times\_rented,

           SUM(p.amount) AS total

    FROM public.payment AS p

        JOIN public.rental AS r ON p.rental\_id = r.rental\_id

        JOIN public.inventory AS i ON r.inventory\_id = i.inventory\_id

        JOIN public.film AS f ON i.film\_id = f.film\_id

    GROUP BY r.inventory\_id, f.title, f.rating, f.length

    ORDER BY total DESC

    LIMIT 10;

    -- Rollback when there is an exception to perserve data integrity

    EXCEPTION

        WHEN OTHERS THEN

            ROLLBACK;

-- Commit the data

END;

$$

LANGUAGE plpgsql;