Student Name: Blake Higdon

Student ID: 011015090

Class: D326

Task A)

Problem: It is hard to forecast required inventory.

Solution: In order to assist with inventory forecasting we shall create a new tables to let the business owners know the rental amount and time associated with each actor so that way when a new movie comes out they will know how much inventory to order based on the actors in that movie. This can save the business money as they would be avoiding overstocking with this table.

A1)

Detailed table: actor\_rental\_details:

rental\_id INT,

actor\_id INT,

actor\_first\_name TEXT,

actor\_last\_name TEXT,

rental\_date DATE,

return\_date DATE,

rental\_duration INTERVAL,

film\_title TEXT

Summary table: actor\_rental\_summary:

actor\_id INT

first\_name TEXT

last\_name TEXT

total\_rentals INT

avg\_rental\_duration INTERVAL

* ("PostgreSQL Sample Database," n.d.)

A2)

actor\_id: represents a unique identifier for each actor (INT value) (apart of actor\_rental\_summary, actor\_rental\_details and actor tables)

first\_name: Stores the first name of each actor (TEXT value) (in actor\_rental\_summary, actor\_rental\_details, and actor tables)

last\_name:  Stores the last name of each actore (TEXT value) (in actor\_rental\_summary, actor\_rental\_details and actor tables)

total\_rentals: This field represents the total rental count for each actor (INT value) (in actor\_rental\_summary table)

avg\_rental\_duration: This field stores the average duration of rentals for each actor. (INTERVAL value) (in actor\_rental\_summary and actor\_rental\_details)

rental\_id: represents a unique identifier for each rental (INT value) (in rental table and actor\_rental\_details)

return\_date and rental\_date: Represents when films were returned or rented out, respectively. (DATE value) (in rental table)

film\_id: A unique identifier for each film (INT value)  (in film, film\_actor, and inventory tables, and actor\_rental\_details)

inventory\_id: Represents a unique identifier for each inventory item (INT value) (in inventory and rental tables)

A3)

Tables Required:

actor

film\_actor

film

inventory

rental

A4)

Transformation:

CREATE OR REPLACE FUNCTION calculate\_rental\_duration(rental\_date TIMESTAMP WITHOUT TIME ZONE, return\_date TIMESTAMP WITHOUT TIME ZONE)

RETURNS INTERVAL

LANGUAGE SQL

AS $$

SELECT (return\_date - rental\_date);

$$;

This transformation will allow us to find get the rental duration which could be useful in forecasting our data. If there is a popular actor with a long rental duration we would be more likely to stock up on more of those movies than if a popular actor had a low average rental duration.

A5)

Detailed Table: The detailed table provides a comprehensive record of individual rental transactions. This gives us the benefit of being able to see the film titles for each transaction which would allow us to see any trends in films as well as actors.

Summary Table: This table is going to provide a high level view of rental performance by actor. This allows the business to identify the most popular actors. This information can be used for inventory forecasting and marketing.

A6)

Ultimately this would depend on the specific needs of the business/stakeholds, which I do not have. However, I believe providing real-time reporting is the ideal solution for the summary table. In the code block provided below, the trigger to update the table occurs with each update to our detailed table so we are essentially having real time data built in because of the trigger. Since our overhead is relatively low I believe providing the stakeholders with the most current information to be best for their decision making processes. As stated in the video, a business such as this is probably financially sensitive in today’s market so providing them with the most current data at any given time is best to ultimately help their bottom line.

F1) A relevant job scheduling tool that could be used is AWS Lambda. AWS is absolutely massive in today’s market so I believe many small businesses such as this one should go ahead and get into the AWS ecosystem to make further business needs down the line that much more easier to integrate into their environment.

B - F: Code provided below.

|  |  |
| --- | --- |
|  |  |
|  |

|  |
| --- |
| DROP TABLE IF EXISTS actor\_rental\_summary;  CREATE TABLE actor\_rental\_summary (  actor\_id INT PRIMARY KEY,  first\_name TEXT,  last\_name TEXT,  total\_rentals INT DEFAULT 0,  avg\_rental\_duration INTERVAL  );  CREATE OR REPLACE PROCEDURE update\_actor\_rental\_summary(  IN rental\_id INT  )  LANGUAGE PLPGSQL  AS $$  DECLARE  actor\_id1 INT;  avg\_duration INTERVAL;  BEGIN  -- Get the actor ID and average rental duration for the given rental  SELECT a.actor\_id, AVG(r.return\_date - r.rental\_date) INTO actor\_id1, avg\_duration  FROM actor a  JOIN film\_actor fa ON a.actor\_id = fa.actor\_id  JOIN film f ON fa.film\_id = f.film\_id  JOIN inventory i ON f.film\_id = i.film\_id  JOIN rental r ON i.inventory\_id = r.inventory\_id  WHERE r.rental\_id = update\_actor\_rental\_summary.rental\_id  GROUP BY a.actor\_id;    UPDATE actor\_rental\_summary ars  SET total\_rentals = ars.total\_rentals + 1,  avg\_rental\_duration = avg\_duration  FROM actor\_rental\_summary ars2  WHERE ars.actor\_id = actor\_id1;  IF NOT FOUND THEN  INSERT INTO actor\_rental\_summary (actor\_id, first\_name, last\_name, total\_rentals, avg\_rental\_duration)  SELECT a.actor\_id, a.first\_name, a.last\_name, 1, avg\_duration  FROM actor a  WHERE a.actor\_id = actor\_id1;  END IF;  END;  $$;  DO $$  DECLARE  rec RECORD;  rental\_cursor CURSOR FOR SELECT rental\_id FROM rental;  BEGIN  OPEN rental\_cursor;  LOOP  FETCH rental\_cursor INTO rec;  EXIT WHEN NOT FOUND;  CALL update\_actor\_rental\_summary(rec.rental\_id);  END LOOP;  CLOSE rental\_cursor;  END $$;  CREATE OR REPLACE FUNCTION update\_rental\_summary\_trigger\_function()  RETURNS TRIGGER AS $$  BEGIN  PERFORM update\_actor\_rental\_summary(NEW.rental\_id);  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  DROP TRIGGER IF EXISTS update\_rental\_summary ON rental;  CREATE OR REPLACE FUNCTION update\_actor\_rental\_summary\_trigger\_function()  RETURNS TRIGGER AS $$  BEGIN    UPDATE actor\_rental\_summary  SET total\_rentals = total\_rentals + 1  WHERE actor\_id = NEW.actor\_id;    UPDATE actor\_rental\_summary  SET avg\_rental\_duration = (  SELECT AVG(rental\_duration)  FROM actor\_rental\_details  WHERE actor\_id = NEW.actor\_id  )  WHERE actor\_id = NEW.actor\_id;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  DROP TRIGGER IF EXISTS update\_actor\_rental\_summary\_trigger ON actor\_rental\_details;  CREATE TRIGGER update\_actor\_rental\_summary\_trigger  AFTER INSERT ON actor\_rental\_details  FOR EACH ROW  EXECUTE FUNCTION update\_actor\_rental\_summary\_trigger\_function();  --("PostgreSQL Triggers," n.d.)  CREATE OR REPLACE FUNCTION format\_duration(duration INTERVAL)  RETURNS TEXT  LANGUAGE PLPGSQL  AS $$  DECLARE  days INT;  hours INT;  minutes INT;  result TEXT;  BEGIN  days := EXTRACT(DAY FROM duration);  hours := EXTRACT(HOUR FROM duration);  minutes := EXTRACT(MINUTE FROM duration);  result := '';  IF days > 0 THEN  result := result || days || ' days ';  END IF;  IF hours > 0 THEN  result := result || hours || ' hours ';  END IF;  IF minutes > 0 THEN  result := result || minutes || ' minutes';  END IF;  RETURN TRIM(result);  END;  $$;  DROP TABLE IF EXISTS actor\_rental\_details;  CREATE TABLE actor\_rental\_details (  rental\_id INT,  actor\_id INT,  actor\_first\_name TEXT,  actor\_last\_name TEXT,  rental\_date DATE,  return\_date DATE,  rental\_duration INTERVAL,  film\_title TEXT,  PRIMARY KEY (rental\_id, actor\_id)  );  CREATE OR REPLACE FUNCTION calculate\_rental\_duration(rental\_date TIMESTAMP WITHOUT TIME ZONE, return\_date TIMESTAMP WITHOUT TIME ZONE)  RETURNS INTERVAL  LANGUAGE SQL  AS $$  SELECT (return\_date - rental\_date);  $$;  --("PostgreSQL Date Functions," n.d.)  rental\_duration field  CREATE OR REPLACE PROCEDURE update\_actor\_rental\_details()  LANGUAGE PLPGSQL  AS $$  BEGIN  INSERT INTO actor\_rental\_details (  rental\_id,  actor\_id,  actor\_first\_name,  actor\_last\_name,  rental\_date,  return\_date,  rental\_duration,  film\_title  )  SELECT  r.rental\_id,  a.actor\_id,  a.first\_name,  a.last\_name,  r.rental\_date,  r.return\_date,  calculate\_rental\_duration(CAST(r.rental\_date AS TIMESTAMP WITHOUT TIME ZONE), CAST(r.return\_date AS TIMESTAMP WITHOUT TIME ZONE)) AS rental\_duration,  f.title AS film\_title  FROM  actor a  JOIN  film\_actor fa ON a.actor\_id = fa.actor\_id  JOIN  film f ON fa.film\_id = f.film\_id  JOIN  inventory i ON f.film\_id = i.film\_id  JOIN  rental r ON i.inventory\_id = r.inventory\_id;  END;  $$;  CALL update\_actor\_rental\_details();  CREATE OR REPLACE FUNCTION get\_actor\_rental\_details()  RETURNS TABLE (  rental\_id INT,  actor\_first\_name TEXT,  actor\_last\_name TEXT,  rental\_date DATE,  return\_date DATE,  rental\_duration\_formatted TEXT,  film\_title TEXT  )  LANGUAGE PLPGSQL  AS $$  BEGIN  RETURN QUERY  SELECT  actor\_rental\_details.rental\_id,  actor\_rental\_details.actor\_first\_name,  actor\_rental\_details.actor\_last\_name,  actor\_rental\_details.rental\_date,  actor\_rental\_details.return\_date,  format\_duration(actor\_rental\_details.rental\_duration) AS rental\_duration\_formatted,  actor\_rental\_details.film\_title  FROM  actor\_rental\_details;  END;  $$;  -- can use AWS Lambda to schedule execution of this stored procedure  CREATE OR REPLACE PROCEDURE refresh\_rental\_data()  LANGUAGE plpgsql  AS $$  BEGIN  DELETE FROM actor\_rental\_details;  DELETE FROM actor\_rental\_summary;  INSERT INTO actor\_rental\_details (  rental\_id,  actor\_id,  actor\_first\_name,  actor\_last\_name,  rental\_date,  return\_date,  rental\_duration,  film\_title  )  SELECT  r.rental\_id,  a.actor\_id,  a.first\_name,  a.last\_name,  r.rental\_date,  r.return\_date,  calculate\_rental\_duration(r.rental\_date, r.return\_date) AS rental\_duration,  f.title AS film\_title  FROM  actor a  JOIN  film\_actor fa ON a.actor\_id = fa.actor\_id  JOIN  film f ON fa.film\_id = f.film\_id  JOIN  inventory i ON f.film\_id = i.film\_id  JOIN  rental r ON i.inventory\_id = r.inventory\_id;  INSERT INTO actor\_rental\_summary (  actor\_id,  first\_name,  last\_name,  total\_rentals,  avg\_rental\_duration  )  SELECT  actor\_id,  first\_name,  last\_name,  COUNT(\*),  AVG(rental\_duration)  FROM  actor\_rental\_details  GROUP BY  actor\_id, first\_name, last\_name;  END;  $$;  BEGIN;  SELECT \* FROM get\_actor\_rental\_details();  ROLLBACK;  --SELECT \*  --FROM actor\_rental\_details;  SELECT first\_name, last\_name, total\_rentals, format\_duration(avg\_rental\_duration) AS avg\_duration\_formatted  FROM actor\_rental\_summary  ORDER BY total\_rentals DESC;  --("PostgreSQL Sample Database," n.d.)  Source(s) used to prepare me for the assignment:  "PostgreSQL Triggers." PostgreSQL Tutorial, PostgreSQL Tutorial, <https://www.postgresqltutorial.com/postgresql-triggers/>.  "PostgreSQL Sample Database." PostgreSQL Tutorial, PostgreSQL Tutorial, <https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/>.  PostgreSQL Date Functions. (n.d.). PostgreSQL Tutorial. Retrieved from <https://www.postgresqltutorial.com/postgresql-date-functions/>.  Advanced Data Management. WGU, 2020, https://apps.cgp-oex.wgu.edu/wgulearning/course/course-v1:WGUx+OEX0034+v01/block-v1:WGUx+OEX0034+v01+type@sequential+block@496695e8664d4a57a86c6762fc4b640e/block-v1:WGUx+OEX0034+v01+type@vertical+block@55f3f46cbc224a3db2bc8a48e5afc2a4 |