1) I have designed the relational diagram for the data warehouse using the principles of Kimball's dimensional modeling, by modeling the fact table and dimension tables in the form of a star schema.
   The grain of the fact table 'Purchased Item' represents a single item purchased which is present on a receipt scanned on the Fetch app. Every purchased item is connected using foreign keys to its dimensions of user, brand, receipt details and time which are modeled in their separate dimension tables. The user table is connected to the fact table using the key *user_id* and it contains details of the user who uploaded the receipt on the Fetch app (i.e. the user who purchased the single item), with details like *birth_date, gender, state* etc. The brand table is connected to the fact table using the key *barcode* and the table contains details of the brand of the purchased item, like *brand_id, cpg_id, category, name* etc. *brand_id* is an alternate key in the brand table, but has not been used as the key to connect to the fact table as the given data associates individual receipt items with *barcode* and not *brand_id* (in the dataset file receipt_items.xlsx). Also, the related_brand_ids associated with a brand_id has been modeled in a separate table to store the related_brand_ids as multiple records/rows instead of colon-separated values in a column, like in the given dataset. The receipt details table is connected to the fact table using the key *receipt_id*, and the table contains details of the uploaded receipt which the single purchased item belongs to – like *store_name, total_spent*, as well as the scanned receipt workflow details in the Fetch app like *rewards_receipt_status, created_date, pending_date, modify_date, flagged_date* etc. Finally, the time dimension table is a new addition which can help in performing time-based aggregations, and contains the year, month and day corresponding to a given timestamp. Whenever a new record in inserted into the fact table, the appropriate function can be applied to extract the year, month and day from the *purchased_timestamp* and the *receipt_scanned_timestamp* and store the same in the time table. This prevents the same operation from having to be performed during business intelligence aggregation queries applied on the data warehouse, thus reducing the latency and optimizing performance.

   The fact table is created using fields from the 'receipt_items' dataset (ex: receipt_item_id, item_index, description etc.), as well as from the 'receipts' dataset (ex: user_id, purchase_timestamp, receipt_scanned_timestamp) to combine keys of the necessary dimensions into a single table.

2) Queries in MySQL:
   a. Which brand saw the most dollars spent in the month of June?

```
WITH temp AS (
        SELECT barcode, SUM(total_final_price) AS price_sum
        FROM Purchased_Item
        WHERE purchase_timestamp IN
        (SELECT timestamp
        FROM Time
        WHERE month = 6)
        GROUP BY barcode
)
```

```
SELECT temp.barcode, b.brand_id, b.name, temp.price_sum
FROM
temp INNER JOIN Brands b
ON b.barcode = temp.barcode
WHERE temp.price_sum = (SELECT MAX(price_sum) from temp);
```

OR

```
WITH temp AS (
        SELECT barcode, SUM(total_final_price) AS price_sum
        FROM Purchased_Item
        WHERE MONTH(purchase_timestamp) = 6
        GROUP BY barcode
)
SELECT temp.barcode, b.brand_id, b.name, temp.price_sum
FROM
temp INNER JOIN Brands b
ON b.barcode = temp.barcode
WHERE temp.price_sum = (SELECT MAX(price_sum) from temp);
```

b. Which user spent the most money in the month of August?

```
WITH temp AS (
        SELECT user_id, SUM(total_final_price) AS price_sum
        FROM Purchased_Item
        WHERE purchase_timestamp IN
        (SELECT timestamp
        FROM Time
        WHERE month = 6)
        GROUP BY user_id
)
SELECT temp.user_id, temp.price_sum
FROM temp WHERE temp.price_sum = (SELECT MAX(price_sum) from
temp);
```

OR

```
WITH temp AS (
        SELECT user_id, SUM(total_final_price) AS price_sum
        FROM Purchased_Item
        WHERE MONTH(purchase_timestamp) = 6
        GROUP BY user_id
)
SELECT temp.user_id, temp.price_sum
FROM temp WHERE temp.price_sum = (SELECT MAX(price_sum) from
temp);
```

c. What user bought the most expensive item?

```
SELECT p.user_id, p.total_final_price/p.quantity
FROM Purchased_Item p WHERE p.total_final_price/p.quantity = (
        SELECT MAX(total_final_price/quantity)
        FROM Purchased_Item
);
```

d. What is the name of the most expensive item purchased?

```
SELECT p.description, p.barcode, b.name, p.total_final_price/p.quantity
FROM Purchased_Item p INNER JOIN Brands b
ON p.barcode = b.barcode
WHERE p.total_final_price/p.quantity = (
        SELECT MAX(total_final_price/quantity)
        FROM Purchased_Item

);
```

e. How many users scanned in each month?

```
SELECT t.month, COUNT(DISTINCT p.user_id)
FROM Purchased_Item p INNER JOIN Time t
ON p. receipt_scanned_timestamp = t.timestamp
GROUP BY t.month;
```