HW3 - Product Catalogs with Relational and NoSQL

Ryan Liang, Vivian Chen, Hiren Patel

Link to **Github repository**: https://github.com/ryanl35/DS4300_homework.git

---

Option #1 - One big table / Relational Model (SQL)

```
SELECT *
FROM products
WHERE
      (diameter = 44
      AND brand = "Tommy Hilfiger"
      AND dial_color = "beige");
```

Option #2 - Category tables / Relational Model (SQL)

```
SELECT *
FROM products p
      JOIN watches w ON p.product_id = w.product_id
WHERE
      (w.diameter = 44
      AND w.brand = "Tommy Hilfiger"
      AND w.dial_color = "beige");
```

Option #3 - Property table / Relational Model (SQL)

```
SELECT *
FROM
      (SELECT t2.property_id, t2.product_id, t2.category_id, t2.key, t2.value
      FROM
            (SELECT t1.property_id, t1.product_id, t1.category_id, t1.key,
            t1.value
            FROM
                  (SELECT property_id, product_id, p.category_id, key, value
                  FROM category c
                  JOIN property p ON p.category_id = c.category_id
                  WHERE
                        c.category_name = "watches") t1
            WHERE
                  t1.key = "Diameter" AND t1.value = "44") t2
      JOIN property p2 ON t2.product_id = p2.product_id
      WHERE
            p2.key = "Brand" AND p2.value = "Tommy Hilfiger") t3
JOIN property p3 ON t3.product_id = p3.product_id
WHERE
      p3.key = "Dial color" AND p3.value = "Beige";
```

Option #4 - Key-Value Store (Redis)

We will have a main database in Redis that will use **zadd** with the "*category*" as the **set** name, the "*product name*" as the **key**, and the location of the hashmap of attributes as the value. The hashmap of attributes would be created with **hset** where the hash is the value which was in the main database. Then, each field would have an attribute name as the key and the description as the value (example: Hashtable 15 (which is a specific watch) has "color" : "blue).

Option #5 - Document Store (MongoDB)

```
db.productCatalog.find({diameter: "44mm", brand: "Tommy Hilfiger",
color: "beige"})
```

Analysis:

Write a couple of paragraphs comparing the different approaches with respect to the complexity of the data model, the query or application complexity, and inherent model limitations if any.

In Option 1, we use 1 big table to store our large product catalog. We typically normalize databases, as we did in Option 2, to remove redundancy and inconsistencies between data, but this would not be necessary to maintain if we are dealing with a historical, unchanging product catalog. Having one big table reduces the need to perform complex joins that perform full table scans to join the data, using less CPU, memory, and I/O and thus making the query faster. However, having a large table can also increase the size of the tables past storage limits (?). Having one big table speeds up retrieval but can slow down updates to the database. Having category tables increases the complexity of the code but allows for better database organization and data consistency. Normalizing the data also increases database security because a database administrator can appropriately segment users' access to certain data. Option 3 represents a hybrid of the first 2 options, where we create a "property" table that keeps track of each attribute of a product in a separate entry. The entries work very similar to Redis, where there is a key and value. In order to be able to find multiple different keys that equal multiple different values, we had to create a lot of nested queries where each query helped filter out one of the key-value pairs which was one of the specified product properties we were searching for.

In Option 4, we used Redis to store products in a key-value store. The data model is complex but the queries are simple. Unlike MongoDB, Redis is memory-based and thus super fast. However, it requires a client that is language specific that is a language library that can translate against the Redis API.

In Option 5, we used MongoDB to render the item as a single document. The idea of querying data using a document store NoSQL database such as MongoDB is efficient and fast. The data doesn't need to join any tables, it instead searches the documents for any attribute that has been specified. The data model is complex but the queries are simple. However, the maximum document size is 16MB.