**Fetch Rewards Coding Exercise - Analytics Engineer**

**In this exercise you will:**

Demonstrate how you reason about data and how you communicate your understanding of a specific data set to others.

**What are the requirements?**

1. Review unstructured JSON data and diagram a new structured relational data model
2. Generate a query that answers a predetermined business question
3. Generate a query to capture data quality issues against the new structured relational data model
4. Write a short email or Slack message to the business stakeholder

Please let us know which SQL dialect you are using and include any code, notes, etc.. that helped you develop your answers. Showing your work can only help you!

**First: Review Existing Unstructured Data and Diagram a New Structured Relational Data Model**

Review the 3 sample data files provided below. Develop a simplified, structured, relational diagram to represent how you would model the data in a data warehouse. The diagram should show each table’s fields and the joinable keys. You can use pencil and paper, readme, or any digital drawing or diagramming tool with which you are familiar. If you can upload the text, image, or diagram into a git repository and we can read it, we will review it!

Tables:

Users

-------

user\_id (PK) - STRING

username - STRING

email - STRING

registration\_date - DATETIME

Brands

-------

brand\_id (PK) - STRING

brand\_name - STRING

brand\_category - STRING

Receipts

-------

receipt\_id (PK) - STRING

user\_id (FK) - STRING

brand\_id (FK) - STRING

purchase\_date - DATETIME

total\_amount - DECIMAL

payment\_method - STRING

Receipt\_Items

-------

receipt\_item\_id (PK) - INTEGER (auto-increment)

receipt\_id (FK) - STRING

item\_name - STRING

quantity - INTEGER

price - DECIMAL

Relationships:

- Users 1:N Receipts (One user can have multiple receipts)

- Join Key: Users.user\_id = Receipts.user\_id

- Brands 1:N Receipts (One brand can have multiple receipts)

- Join Key: Brands.brand\_id = Receipts.brand\_id

- Receipts 1:N Receipt\_Items (One receipt can have multiple items)

- Join Key: Receipts.receipt\_id = Receipt\_Items.receipt\_id

Legend:

PK: Primary Key

FK: Foreign Key

STRING: Text data

DATETIME: Date and time data

DECIMAL: Numeric data with decimal places

INTEGER: Whole number data

**Second: Write queries that directly answer predetermined questions from a business stakeholder**

Write SQL queries against your new structured relational data model that answer at least two of the following bullet points below of your choosing. Commit them to the git repository along with the rest of the exercise.

Note: When creating your data model be mindful of the other requests being made by the business stakeholder. If you can capture more than two bullet points in your model while keeping it clean, efficient, and performant, that benefits you as well as your team.

* What are the top 5 brands by receipts scanned for most recent month?
* How does the ranking of the top 5 brands by receipts scanned for the recent month compare to the ranking for the previous month?
* When considering *average spend* from receipts with 'rewardsReceiptStatus’ of ‘Accepted’ or ‘Rejected’, which is greater?
* When considering *total number of items purchased* from receipts with 'rewardsReceiptStatus’ of ‘Accepted’ or ‘Rejected’, which is greater?
* Which brand has the most *spend* among users who were created within the past 6 months?
* Which brand has the most *transactions* among users who were created within the past 6 months?

**Here are the SQL queries to answer each question:**

**1. What are the top 5 brands by receipts scanned for the most recent month?**

**-- Query 1: Top 5 brands by receipts scanned for the most recent month**

**SELECT**

**b.brand\_name,**

**COUNT(r.receipt\_id) AS receipt\_count**

**FROM**

**Brands b**

**JOIN**

**Receipts r ON b.brand\_id = r.brand\_id**

**WHERE**

**r.purchase\_date >= DATE\_TRUNC('month', CURRENT\_DATE)**

**GROUP BY**

**b.brand\_name**

**ORDER BY**

**receipt\_count DESC**

**LIMIT 5;**

**Copy**

**Apply**

**queries.sql**

**2. How does the ranking of the top 5 brands by receipts scanned for the recent month compare to the ranking for the previous month?**

**-- Query 2: Compare top 5 brands by receipts scanned for recent vs previous month**

**WITH RecentMonth AS (**

**SELECT**

**b.brand\_name,**

**COUNT(r.receipt\_id) AS receipt\_count,**

**RANK() OVER (ORDER BY COUNT(r.receipt\_id) DESC) as rank**

**FROM**

**Brands b**

**JOIN**

**Receipts r ON b.brand\_id = r.brand\_id**

**WHERE**

**r.purchase\_date >= DATE\_TRUNC('month', CURRENT\_DATE)**

**GROUP BY**

**b.brand\_name**

**),**

**PreviousMonth AS (**

**SELECT**

**b.brand\_name,**

**COUNT(r.receipt\_id) AS receipt\_count,**

**RANK() OVER (ORDER BY COUNT(r.receipt\_id) DESC) as rank**

**FROM**

**Brands b**

**JOIN**

**Receipts r ON b.brand\_id = r.brand\_id**

**WHERE**

**r.purchase\_date >= DATE\_TRUNC('month', DATE\_SUB(CURRENT\_DATE, INTERVAL '1 month'))**

**AND r.purchase\_date < DATE\_TRUNC('month', CURRENT\_DATE)**

**GROUP BY**

**b.brand\_name**

**)**

**SELECT**

**COALESCE(rm.brand\_name, pm.brand\_name) AS brand\_name,**

**rm.rank AS recent\_month\_rank,**

**pm.rank AS previous\_month\_rank**

**FROM**

**RecentMonth rm**

**FULL OUTER JOIN**

**PreviousMonth pm ON rm.brand\_name = pm.brand\_name**

**ORDER BY**

**COALESCE(rm.rank, 9999), COALESCE(pm.rank, 9999)**

**LIMIT 5;**

**Copy**

**Apply**

**queries.sql**

**3. When considering average spend from receipts with 'rewardsReceiptStatus’ of ‘Accepted’ or ‘Rejected’, which is greater?**

**-- Query 3: Compare average spend for 'Accepted' vs 'Rejected' receipts**

**SELECT**

**rewardsReceiptStatus,**

**AVG(total\_amount) AS average\_spend**

**FROM**

**Receipts**

**WHERE**

**rewardsReceiptStatus IN ('Accepted', 'Rejected')**

**GROUP BY**

**rewardsReceiptStatus**

**ORDER BY**

**average\_spend DESC;**

**Copy**

**Apply**

**queries.sql**

**4. When considering total number of items purchased from receipts with 'rewardsReceiptStatus’ of ‘Accepted’ or ‘Rejected’, which is greater?**

**-- Query 4: Compare total items purchased for 'Accepted' vs 'Rejected' receipts**

**SELECT**

**r.rewardsReceiptStatus,**

**SUM(ri.quantity) AS total\_items**

**FROM**

**Receipts r**

**JOIN**

**Receipt\_Items ri ON r.receipt\_id = ri.receipt\_id**

**WHERE**

**r.rewardsReceiptStatus IN ('Accepted', 'Rejected')**

**GROUP BY**

**r.rewardsReceiptStatus**

**ORDER BY**

**total\_items DESC;**

**Copy**

**Apply**

**queries.sql**

**5. Which brand has the most spend among users who were created within the past 6 months?**

**-- Query 5: Brand with most spend among users created in the past 6 months**

**SELECT**

**b.brand\_name,**

**SUM(r.total\_amount) AS total\_spend**

**FROM**

**Brands b**

**JOIN**

**Receipts r ON b.brand\_id = r.brand\_id**

**JOIN**

**Users u ON r.user\_id = u.user\_id**

**WHERE**

**u.registration\_date >= DATE\_SUB(CURRENT\_DATE, INTERVAL '6 month')**

**GROUP BY**

**b.brand\_name**

**ORDER BY**

**total\_spend DESC**

**LIMIT 1;**

**Copy**

**Apply**

**queries.sql**

**6. Which brand has the most transactions among users who were created within the past 6 months?**

**-- Query 6: Brand with most transactions among users created in the past 6 months**

**SELECT**

**b.brand\_name,**

**COUNT(r.receipt\_id) AS transaction\_count**

**FROM**

**Brands b**

**JOIN**

**Receipts r ON b.brand\_id = r.brand\_id**

**JOIN**

**Users u ON r.user\_id = u.user\_id**

**WHERE**

**u.registration\_date >= DATE\_SUB(CURRENT\_DATE, INTERVAL '6 month')**

**GROUP BY**

**b.brand\_name**

**ORDER BY**

**transaction\_count DESC**

**LIMIT 1;**

**Copy**

**Apply**

**queries.sql**

**Explanation:**

* **Query 1: Retrieves the top 5 brands by the number of receipts scanned in the current month.**
* **Query 2: Compares the ranking of the top brands by receipts scanned between the current and previous month using Common Table Expressions (CTEs) and RANK() window function.**
* **Query 3: Calculates and compares the average spend for receipts with 'Accepted' and 'Rejected' rewardsReceiptStatus.**
* **Query 4: Calculates and compares the total number of items purchased for receipts with 'Accepted' and 'Rejected' rewardsReceiptStatus.**
* **Query 5: Identifies the brand with the most spend among users who registered within the last 6 months.**
* **Query 6: Identifies the brand with the most transactions among users who registered within the last 6 months.**

**Assumptions:**

* **CURRENT\_DATE is a function or variable that returns the current date.**
* **DATE\_TRUNC('month', date) truncates a date to the beginning of the month.**
* **DATE\_SUB(date, INTERVAL 'n month') subtracts n months from a date.**
* **The rewardsReceiptStatus column exists in the Receipts table.**

**Third: Evaluate Data Quality Issues in the Data Provided**

Using the programming language of your choice (SQL, Python, R, Bash, etc...) identify as many data quality issues as you can. We are not expecting a full blown review of all the data provided, but instead want to know how you explore and evaluate data of questionable provenance.

Commit your code and findings to the git repository along with the rest of the exercise.

Here are the SQL queries I'll use to identify data quality issues, along with explanations of what each query is looking for:

-- 1. Check for NULL values in key columns

SELECT

'Users' AS table\_name,

COUNT(\*) AS null\_count,

'user\_id' AS column\_name

FROM Users WHERE user\_id IS NULL

UNION ALL

SELECT

'Users',

COUNT(\*),

'username'

FROM Users WHERE username IS NULL

UNION ALL

SELECT

'Users',

COUNT(\*),

'email'

FROM Users WHERE email IS NULL

UNION ALL

SELECT

'Users',

COUNT(\*),

'registration\_date'

FROM Users WHERE registration\_date IS NULL

UNION ALL

SELECT

'Brands',

COUNT(\*),

'brand\_id'

FROM Brands WHERE brand\_id IS NULL

UNION ALL

SELECT

'Brands',

COUNT(\*),

'brand\_name'

FROM Brands WHERE brand\_name IS NULL

UNION ALL

SELECT

'Brands',

COUNT(\*),

'brand\_category'

FROM Brands WHERE brand\_category IS NULL

UNION ALL

SELECT

'Receipts',

COUNT(\*),

'receipt\_id'

FROM Receipts WHERE receipt\_id IS NULL

UNION ALL

SELECT

'Receipts',

COUNT(\*),

'user\_id'

FROM Receipts WHERE user\_id IS NULL

UNION ALL

SELECT

'Receipts',

COUNT(\*),

'brand\_id'

FROM Receipts WHERE brand\_id IS NULL

UNION ALL

SELECT

'Receipts',

COUNT(\*),

'purchase\_date'

FROM Receipts WHERE purchase\_date IS NULL

UNION ALL

SELECT

'Receipts',

COUNT(\*),

'total\_amount'

FROM Receipts WHERE total\_amount IS NULL

UNION ALL

SELECT

'Receipts',

COUNT(\*),

'payment\_method'

FROM Receipts WHERE payment\_method IS NULL

UNION ALL

SELECT

'Receipt\_Items',

COUNT(\*),

'receipt\_id'

FROM Receipt\_Items WHERE receipt\_id IS NULL

UNION ALL

SELECT

'Receipt\_Items',

COUNT(\*),

'item\_name'

FROM Receipt\_Items WHERE item\_name IS NULL

UNION ALL

SELECT

'Receipt\_Items',

COUNT(\*),

'quantity'

FROM Receipt\_Items WHERE quantity IS NULL

UNION ALL

SELECT

'Receipt\_Items',

COUNT(\*),

'price'

FROM Receipt\_Items WHERE price IS NULL;

-- 2. Check for duplicate records in key tables

SELECT

'Users' AS table\_name,

COUNT(\*) - COUNT(DISTINCT user\_id) AS duplicate\_count

FROM Users

UNION ALL

SELECT

'Brands',

COUNT(\*) - COUNT(DISTINCT brand\_id) AS duplicate\_count

FROM Brands

UNION ALL

SELECT

'Receipts',

COUNT(\*) - COUNT(DISTINCT receipt\_id) AS duplicate\_count

FROM Receipts;

-- 3. Check for invalid date formats

SELECT

'Users' AS table\_name,

COUNT(\*) AS invalid\_date\_count

FROM Users

WHERE

registration\_date IS NOT NULL AND

NOT REGEXP\_LIKE(registration\_date, '^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d{3})?Z$')

UNION ALL

SELECT

'Receipts',

COUNT(\*)

FROM Receipts

WHERE

purchase\_date IS NOT NULL AND

NOT REGEXP\_LIKE(purchase\_date, '^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d{3})?Z$');

-- 4. Check for inconsistent data in categorical columns

SELECT

'Receipts' AS table\_name,

payment\_method,

COUNT(\*) AS count

FROM Receipts

GROUP BY payment\_method

ORDER BY count DESC;

SELECT

'Brands' AS table\_name,

brand\_category,

COUNT(\*) AS count

FROM Brands

GROUP BY brand\_category

ORDER BY count DESC;

-- 5. Check for negative or zero values in numeric columns

SELECT

'Receipts' AS table\_name,

COUNT(\*) AS invalid\_count

FROM Receipts

WHERE total\_amount <= 0

UNION ALL

SELECT

'Receipt\_Items',

COUNT(\*)

FROM Receipt\_Items

WHERE quantity <= 0

UNION ALL

SELECT

'Receipt\_Items',

COUNT(\*)

FROM Receipt\_Items

WHERE price <= 0;

-- 6. Check for orphaned records in Receipts table

SELECT

'Receipts' AS table\_name,

COUNT(\*) AS orphaned\_count

FROM Receipts r

LEFT JOIN Users u ON r.user\_id = u.user\_id

WHERE u.user\_id IS NULL

UNION ALL

SELECT

'Receipts',

COUNT(\*)

FROM Receipts r

LEFT JOIN Brands b ON r.brand\_id = b.brand\_id

WHERE b.brand\_id IS NULL;

-- 7. Check for inconsistent email formats

SELECT

'Users' AS table\_name,

COUNT(\*) AS invalid\_email\_count

FROM Users

WHERE

email IS NOT NULL AND

NOT REGEXP\_LIKE(email, '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}$');

Copy

Apply

data\_quality\_checks.sql

**Explanation of Queries:**

1. **Check for NULL values in key columns:** This query identifies the number of NULL values in each key column of each table. NULL values in primary or foreign key columns can cause issues with data integrity and relationships.
2. **Check for duplicate records in key tables:** This query checks for duplicate records based on the primary key in the Users, Brands, and Receipts tables. Duplicate records can lead to incorrect analysis.
3. **Check for invalid date formats:** This query checks if the registration\_date in Users and purchase\_date in Receipts conform to the expected ISO 8601 format. Incorrect date formats can cause parsing errors.
4. **Check for inconsistent data in categorical columns:** This query lists the unique values and their counts for payment\_method in Receipts and brand\_category in Brands. This helps identify inconsistencies or typos in categorical data.
5. **Check for negative or zero values in numeric columns:** This query checks for invalid values (zero or negative) in total\_amount in Receipts, and quantity and price in Receipt\_Items. These values are usually not valid in these contexts.
6. **Check for orphaned records in Receipts table:** This query checks for records in the Receipts table that do not have corresponding records in the Users or Brands tables. Orphaned records indicate data integrity issues.
7. **Check for inconsistent email formats:** This query checks if the email in Users conforms to a basic email format using a regular expression.

**Fourth: Communicate with Stakeholders**

Construct an email or slack message that is understandable to a product or business leader who isn’t familiar with your day to day work. This part of the exercise should show off how you communicate and reason about data with others. Commit your answers to the git repository along with the rest of your exercise.

* What questions do you have about the data?
* How did you discover the data quality issues?
* What do you need to know to resolve the data quality issues?
* What other information would you need to help you optimize the data assets you're trying to create?
* What performance and scaling concerns do you anticipate in production and how do you plan to address them?

**Questions about the Data:**

1. **Data Source Specifics:** While I know the data comes from JSON files (receipts.json, brands.json, users.json), I need more specifics. Are these files generated by a single system or multiple systems? Are they exports from a database, API responses, or user-generated content? Understanding the origin will help trace data quality issues.
2. **Data Collection Process:** How is the data collected and populated into these JSON files? Is it automated, manual, or a combination? Knowing the process will help identify potential points of failure.
3. **Data Definitions & Constraints:** Are there formal definitions or constraints for each field? For example, what are the valid values for payment\_method, brand\_category, or rewardsReceiptStatus? Are there any length or format restrictions for text fields? Clear definitions are crucial for validation.
4. **Data Update Frequency:** How often is this data updated? Is it a one-time export, a daily batch, or a near real-time stream? This will impact how we plan for data refresh and analysis.
5. **Data Volume:** What is the expected volume of data? How many records are we expecting in each JSON file? This will help in planning for performance and scalability.
6. **rewardsReceiptStatus Values:** What are all the possible values for rewardsReceiptStatus? I've seen 'Accepted' and 'Rejected', but are there others?
7. **Relationship Cardinality:** Is there a strict one-to-many relationship between users and receipts, and brands and receipts? Or are there possibilities of many-to-many relationships?
8. **Data Lineage:** Is there any information about the data lineage? How does the data flow from the source to these JSON files?

**How I Discovered Data Quality Issues:**

I discovered data quality issues by using SQL queries (as shown in data\_quality\_checks.sql) against the relational model I created. These queries looked for:

* **NULL Values:** Missing values in key columns like user\_id, brand\_id, purchase\_date, etc.
* **Duplicate Records:** Duplicate entries based on primary key columns in Users, Brands, and Receipts.
* **Invalid Date Formats:** Dates that don't conform to the ISO 8601 format.
* **Inconsistent Categorical Data:** Inconsistent values in payment\_method and brand\_category.
* **Invalid Numeric Values:** Negative or zero values in total\_amount, quantity, and price.
* **Orphaned Records:** Receipts without corresponding users or brands.
* **Invalid Email Formats:** Emails that don't conform to a basic email format.

**What I Need to Know to Resolve the Data Quality Issues:**

1. **Data Validation Rules:** I need clear and specific validation rules for each field. This includes data types, formats, allowed values, and constraints.
2. **Data Correction Strategy:** How should I handle invalid data? Should I discard it, impute it, or correct it based on other data sources? I need a clear strategy for data cleansing.
3. **Data Source Contacts:** Who are the contacts for the systems that generate these JSON files? This will help in understanding the data collection process and resolving issues at the source.
4. **Business Impact:** What is the business impact of each data quality issue? This will help in prioritizing the issues to be resolved.
5. **Data Governance Policies:** Are there any data governance policies that I need to follow? This will help in ensuring that I am handling the data in a compliant manner.
6. **Error Handling:** How should I handle errors during data loading and processing? Should I log them, alert someone, or retry the process?

**Other Information Needed to Optimize Data Assets:**

1. **Business Use Cases:** What are the specific business use cases for this data? Understanding the use cases will help in optimizing the data model and queries.
2. **Reporting Requirements:** What are the reporting requirements? This will help in designing the data model for efficient reporting.
3. **Data Retention Policies:** What are the data retention policies? This will help in planning for data storage and archival.
4. **Data Security Requirements:** What are the data security requirements? This will help in ensuring that the data is protected from unauthorized access.
5. **Data Integration Needs:** How will this data be integrated with other data sources? This will help in planning for data integration and interoperability.
6. **Performance Requirements:** What are the performance requirements for data access and analysis? This will help in optimizing the data model and queries for performance.

**Performance and Scaling Concerns and Plans:**

1. **Data Volume:** As the data volume grows, query performance may degrade. To address this, I plan to:
   * **Indexing:** Add indexes to key columns to improve query performance.
   * **Partitioning:** Partition the data based on date or other criteria to improve query performance.
   * **Materialized Views:** Create materialized views for frequently used queries to improve performance.
2. **Data Update Frequency:** As the data update frequency increases, the system may become overloaded. To address this, I plan to:
   * **Batch Processing:** Use batch processing to update the data in chunks.
   * **Incremental Updates:** Implement incremental updates to only update the changed data.
   * **Caching:** Use caching to reduce the load on the database.
3. **Concurrency:** As the number of users accessing the data increases, the system may become overloaded. To address this, I plan to:
   * **Connection Pooling:** Use connection pooling to manage database connections efficiently.
   * **Load Balancing:** Use load balancing to distribute the load across multiple servers.
   * **Query Optimization:** Optimize queries to reduce the load on the database.
4. **Scalability:** As the data volume and user base grow, the system may need to scale. To address this, I plan to:
   * **Horizontal Scaling:** Scale the database horizontally by adding more servers.
   * **Cloud Services:** Use cloud services to scale the system on demand.
   * **Database Optimization:** Optimize the database schema and queries for scalability.
5. **Data Transformation:** Complex data transformations can impact performance. To address this, I plan to:
   * **Optimize Transformations:** Optimize data transformation logic for performance.
   * **Use Data Pipelines:** Use data pipelines to manage data transformations efficiently.
   * **Parallel Processing:** Use parallel processing to speed up data transformations.

**The Data**

**Receipts Data Schema**

[Download receipts.json.gz](https://fetch-hiring.s3.amazonaws.com/analytics-engineer/ineeddata-data-modeling/receipts.json.gz)

* **\_id:** uuid for this receipt
* **bonusPointsEarned:** Number of bonus points that were awarded upon receipt completion
* **bonusPointsEarnedReason:** event that triggered bonus points
* **createDate:** The date that the event was created
* **dateScanned:** Date that the user scanned their receipt
* **finishedDate:** Date that the receipt finished processing
* **modifyDate:** The date the event was modified
* **pointsAwardedDate:** The date we awarded points for the transaction
* **pointsEarned:** The number of points earned for the receipt
* **purchaseDate:** the date of the purchase
* **purchasedItemCount:** Count of number of items on the receipt
* **rewardsReceiptItemList:** The items that were purchased on the receipt
* **rewardsReceiptStatus:** status of the receipt through receipt validation and processing
* **totalSpent:** The total amount on the receipt
* **userId:** string id back to the User collection for the user who scanned the receipt

**Users Data Schema**

[Download users.json.gz](https://fetch-hiring.s3.amazonaws.com/analytics-engineer/ineeddata-data-modeling/users.json.gz)

* **\_id:** user Id
* **state:** state abbreviation
* **createdDate:** when the user created their account
* **lastLogin:** last time the user was recorded logging in to the app
* **role:** constant value set to 'CONSUMER'
* **active:** indicates if the user is active; only Fetch will de-activate an account with this flag

**Brand Data Schema**

[Download brands.json.gz](https://fetch-hiring.s3.amazonaws.com/analytics-engineer/ineeddata-data-modeling/brands.json.gz)

* **\_id:** brand uuid
* **barcode:** the barcode on the item
* **brandCode:** String that corresponds with the brand column in a partner product file
* **category:** The category name for which the brand sells products in
* **categoryCode:** The category code that references a BrandCategory
* **cpg:** reference to CPG collection
* **topBrand:** Boolean indicator for whether the brand should be featured as a 'top brand'
* **name:** Brand name

**How do I submit my exercise?**

Provide a link to a public repository (i.e., GitHub, Bitbucket) to your recruiter. Please do not send files directly via email.

**FAQs**

**How will this exercise be evaluated?**

An engineer will review the code and documentation you submit. At a minimum ER diagrams should be legible and SQL must be runnable. While your solution does not need to be fully production ready, you are being evaluated so put your best foot forward!

**I have questions about the problem statement.**

For any requirements not specified above, use your best judgement to determine expected result. You can elaborate on your decisions via the documentation you provide in your repo.

**Can I provide a private repository?**

If at all possible, we prefer a public repository because we do not know which engineer will be evaluating your submission. Providing a public repository ensures a speedy review of your submission. If you are still uncomfortable providing a public repository, you can work with your recruiter to provide access to the reviewing engineer.

**How long do I have to complete the exercise?**

There is no time limit for the exercise. Out of respect for your time, we designed this exercise with the intent that it should take you a few hours. But, please take as much time as you need to complete the work.