Code Smells and Refactoring Techniques

1. Long Method

- **Code Smell:**
  - The `query_time` method is relatively long and involves conditional logic for constructing a query string.
- **Refactoring Technique:**
  - **Extract Method:**
    - Extracted the logic for constructing the query into a separate method (`construct_query`).

```
def construct_query(self, start_date, end_date, task, tag):
    # ... (query construction logic)
```

2. Large Class

- **Code Smell:**
  - The `TimeTracker` class handles multiple responsibilities.
- **Refactoring Technique:**
  - **Extract Class:**
    - Created a separate class (`DatabaseManager`) for database-related functionality.

```
class DatabaseManager:
    # ... (database-related methods)
```

    - Updated the `TimeTracker` class to use an instance of `DatabaseManager`.

3. Comments

- **Code Smell:**
  - Comments in the code could be more informative.
- **Refactoring Technique:**
  - **Self-Documenting Code:**
    - Improved method names and code structure to make the comments less necessary.

4. Dead Code

- **Code Smell:**

    - Using `print` for error messages in the `record_time` method.

- **Refactoring Technique:**

    - **Exception Handling:**
        - Raised a `ValueError` for error cases instead of using `print`.

```
raise ValueError("Error: Please provide values for all fields.")
```

5. Alternate Class with Different Interfaces

- **Code Smell:**

    - Inconsistent usage of the `datetime` module for date manipulation.

- **Refactoring Technique:**

    - **Consistent Datetime Handling:**
        - Used `parser.parse` consistently for date parsing.

```
date = parser.parse("today").strftime("%Y/%m/%d")
```

Other Code Smells

6. Speculative Generality

- In early stages of project development, much of the app had been focused on preparing for the future features to be implemented.

7. Divergent Change

- Changes made to the database class will sometimes also need to be made to the main program class.

8. Inapropreate Intimacy

- The time tracker class and the database class rely heavily on eachother in order to function properly.

9. Primative Obsession

- Date and time handling might benefit from using a dedicated library or class instead of handling strings directly.

10. Message Chains

- The TimeTracker class directly accesses attributes of the record object in the generate_report method. This could lead to issues if the internal structure of the time_records table changes.