Database Final Report

Group:

05

Member:

112550134 賴僑樞、112550106 林瀚璿、112550174 林紹安、

112550121 江宸安、112550010 朱自中

Main Idea

The application aims to analyze Formula 1 race performance and present detailed data on a user-friendly website. It serves F1 enthusiasts by providing race, driver, and circuit information, along with interactive features for data analysis and user engagement.

Data

The dataset contains comprehensive Formula 1 data from 1950 to 2024, organized into 17 tables, including details of circuits, drivers, constructors, races, and standings. All attributes include:

1. **Circuits Table:**

   Contains data about F1 circuits, including location details like latitude, longitude, and altitude.

   - circuits_id: Unique ID for each circuit.
   - circuit_ref: Circuit reference name.
   - circuit_name: Circuit name.
   - city: City or location of the circuit.
   - country: Country where the circuit is located.

- lat, lng: Latitude and longitude of the circuit.
- alt: Altitude of the circuit (in meters).
- wiki_url: Official URL of the circuit.

2. **Constructor Results Table:**

   Performance data of constructors per race, including points and status.

- constructor_result_id: Unique ID for constructor results.
- race_id: Refers to the specific race.
- constructor_id: Constructor's unique identifier.
- points: Points scored by the constructor in that race.
- status_of_result: Performance status (e.g., finished, retired).

3. **Constructor Standings Table:**

   Constructor rankings by points and wins.

- constructor_standings_id: Unique ID for standings.
- race_id: Refers to a particular race.
- Constructor_id: Constructor's unique identifier.
- points: Total points the constructor earned.
- position: Constructor’s position in the standings.
- position_text: Text representation of position.
- wins: Number of race wins by the constructor.

4. **Pit Stops Table:**

   Information about pit stops during races, including duration.

- race_id: Refers to a specific race.
- driver_id: Driver’s unique identifier.
- stop_num: Number of the pit stop.
- lap_num: Lap during which the pit stop occurred.
- time_of_pit_stop: Pit stop time.
- duration: Duration of the pit stop.
- milliseconds: Duration in milliseconds.

5. **Constructors Table:**

Basic details of constructors like nationality and names.

- constructor_id: Unique ID for the constructor.
- constructor_ref: Reference name for constructor.
- constructor_name: Constructor name.
- nationality: Constructor's nationality.
- wiki_url: Official URL of the constructor.

6. **Driver Standings Table:**

Drivers' ranking data across races.

- driver_standings_id: Unique ID for the driver standings.
- race_id: Refers to a specific race.
- driver_id: Driver's unique identifier.
- points: Total points the driver earned.
- position: Driver's position in the standings.
- position_text: Text representation of position.
- wins: Number of race wins by the driver.

7. **Drivers Table:**

Personal information about drivers including nationality, date of birth, and code.

- driver_id: Unique ID for the driver.
- driver_ref: Reference name for the driver.
- driver_number: Driver's car number.
- code: Driver's code (short form).
- f_name: Driver's first names.
- l_name: Driver's last names.
- date_of_birth: Date of birth.
- nationality: Driver's nationality.
- wiki_url: Official URL for more details on the driver.

8. **Lap Times Table:**

Details of lap times for drivers in each race.

- race_id: Refers to a specific race.
- driver_id: Driver’s unique identifier.
- lap_num: Lap number.
- position: Driver's position for the lap.
- finish_time: Lap time.
- finish_time_in_milliseconds: Lap time in milliseconds.

9. **Qualifying Table:**

Qualifying results for each driver per race.

- qualify_id: Unique ID for qualifying data.
- race_id: Refers to a specific race.
- driver_id: Driver’s unique identifier.
- constructor_id: Constructor’s unique identifier.
- car_num: Driver’s car number.
- position: Qualifying position.
- q1, q2, q3: Times for qualifying sessions 1, 2, and 3.

10. **Races Table:**

Information about races, including dates, circuits, and race names.

- race_id: Unique ID for the race.
- year_of_race: Year the race was held.
- round: Round number in the season.
- circuits_id: Refers to the circuit.
- circuit_name: Name of the race.
- race_date: Date of the race.
- race_time: Time the race started.
- wiki_url: Official race URL.
- fp1_date, fp1_time: Date and time of Free Practice 1.

- fp2_date, fp2_time: Date and time of Free Practice 2.
- fp3_date, fp3_time: Date and time of Free Practice 3.
- quali_date, quali_time: Date and time of qualifying.
- sprint_date, sprint_time: Date and time of sprint race.

11. Sprint Results Table:

Results of sprint races, including fastest laps and positions.

- result_id: Unique ID for the sprint result.
- race_id: Refers to a specific sprint race.
- driver_id: Driver's unique identifier.
- constructor_id: Constructor's unique identifier.
- car_num: Driver's car number.
- position_grid: Starting grid position for the sprint.
- position: Final position.
- position_text: Text for the position.
- position_order: Order of position.
- points: Points scored in the sprint.
- laps: Number of laps completed in the sprint.
- time: Finishing time.
- time_in_milliseconds: Time in milliseconds.
- fastest_lap: Number of the fastest lap.
- fastest_lap_time: Time for the fastest lap.
- status_id: Status of the driver.

12. Results Table:

Detailed race results for each driver, including position, points, and laps.

- result_id: Unique ID for the result.
- race_id: Refers to a specific race.
- driver_id: Driver's unique identifier.
- constructor_id: Constructor's unique identifier.
- car_num: Driver's car number.
- position_grid: Starting grid position.

- position: Finishing position.
- position_text: Text for the position.
- position_order: Order of position (numerical).
- points: Points scored in the race.
- laps: Number of laps completed.
- time: Finishing time.
- time_in_milliseconds: Time in milliseconds.
- fastest_lap: Number of the fastest lap.
- rank_of_fastest_lap: Rank for fastest lap.
- fastest_lap_time: Time for the fastest lap.
- fastest_lap_speed: Speed during the fastest lap.
- status_id: Status of the driver (finished, retired, etc.).

13. **Seasons Table:**

Seasons data, including year and corresponding URLs.

- year_of_race: Season year.
- wiki_url: Official URL for the season overview.

14. Status Table: Status codes for race outcomes such as "Finished" or "Retired."

- status_id: Unique ID for the status.
- status_name: Status description (e.g., "Finished," "Retired," etc.).

- **Users Table: Used to manage user accounts, facilitate login/logout operations, and associate user activity (like comments) with specific accounts.**
  - id: A unique identifier for each user.
  - username: The name chosen by the user for their account. It must be unique for each user.
  - password: A hashed and securely stored password to authenticate the user.

16. Comments Table: Enables users to share feedback, insights, or opinions about various aspects of Formula 1 data on the application.

- id: A unique identifier for each comment.
- username: Links the comment to the user who made it, referencing the username field in the Users table.
- text: The content of the comment provided by the user.
- created_at: A timestamp indicating when the comment was created.

17. countries Table: This table provides standardized information about countries, which could be used for attributes like race locations, driver nationalities, or constructor origins.

- num_code: A numeric code assigned to the country as per ISO standards.
- alpha_2_code: The two-letter country code (e.g., "US" for the United States).
- alpha_3_code: The three-letter country code (e.g., "USA" for the United States).
- en_short_name: The official English short name of the country.
- nationality: The term used to describe a person from that country (e.g., "American" for the United States).

ER Model

The ER model will include:

- **Entities**: Drivers, Races, Circuits, Constructors, Results, etc.

- **Relationships**: `Race` connects to `Driver`, `Constructor`, and `Circuit`. `Driver` and `Constructor` have a many-to-many relationship through `Results`.

Database

Database System Used

- Type: MySQL
    - Relational database management system (RDBMS) that is well-suited for structured data with relationships.
    - Supports ACID compliance, ensuring reliable transactions and data integrity.

Database Maintenance

1. Data Updates:
   - New data is inserted via CRUD operations.
   - Functions like 「insertDriver」 and 「insertResult」 allow real-time data addition.
2. Data Deletion:
   - Outdated or erroneous entries can be deleted via functions like 「deleteDriver」.
3. Schema Management:
   - Adjustments to table structures can be made using SQL commands such as 「ALTER TABLE」.
4. Backup Strategy:
   - Regular database backups are scheduled using MySQL's dump utility to prevent data loss.

Database Connection

1. Configuration:
   - 「db_config」 stores the connection details: host, user, password, and database name.
   - A reusable function 「get_db_connection()」 creates and returns a connection to the database using 「mysql.connector」.

```
def get_db_connection():
    return mysql.connector.connect(**db_config)
```

2. Connection Lifecycle:
   - Connections are opened when executing queries and closed promptly afterward to avoid resource leakage.

Backend Query Processing

1. Example Query: To fetch race details:

```
SELECT race_id, circuit_name, wiki_url
FROM races
WHERE year_of_race = %s;
```

- The parameterized query prevents SQL injection by using placeholders (`%s`) for user inputs.

2. Flow:

- Input validation occurs at the application level.
- Queries are executed using a cursor obtained from the database connection.
- Results are fetched and structured as Python dictionaries.

Exception Handling

1. Database Errors:

- Common issues like missing data or unique constraint violations are caught using 「try-except」blocks.

- Example:

```
try:
    cursor.execute("INSERT INTO users (username, password) VALUES (%s, %s)", (username, hash_password(password)))
    db_connection.commit()
except mysql.connector.Error as err:
    flash(f"Error: {err}", "danger")
```

2. Unexpected Inputs:

- Missing or invalid inputs are handled gracefully.
- Users are shown appropriate error messages (e.g., "All fields are required").

3. SQL Injection Prevention:

- All queries use parameterized placeholders to prevent malicious input from being executed.

Graphical Representation

The flow from the frontend to the database is as follows:

1. Frontend: User actions (e.g., form submission, button clicks).

2. Backend:

- Validates input.
- Constructs parameterized SQL queries.

- Executes queries using MySQL cursor.

3. Database:
    - Executes the query and returns results.
4. Frontend: Displays the results or error messages.

Application

Interface

- Homepage: Displays recent comments and a list of countries.
- Login/Register: Secure user authentication with a user-friendly interface.
- Search Bar: Allows users to search drivers or races dynamically.
- Filter Controls: Enables filtering and sorting by attributes like nationality or race year.
- CRUD Buttons: Provide options to create, update, and delete data interactively.

Application Functions

1. CRUD Implementation

1. Create:
   - Function: insertDriver/ insertResult/ insertUser
   - Adds a new driver to the database with required attributes.
   - Query:

```
INSERT INTO drivers (f_name, l_name, date_of_birth, nationality, wiki_url)
VALUES (%s, %s, %s, %s, %s);
```

     - The query uses placeholders for secure and dynamic data insertion.
   - Exception Handling:
     - Checks for missing fields.
     - Rolls back the transaction if an error occurs.
2. Read:
   - Function: getRace/ getDriver/ get_teams
   - Fetches details of races in a specific year.
   - Query:

```
SELECT race_id, circuit_name, wiki_url
FROM races
WHERE year_of_race = %s;
```

     - Filters results to improve performance by narrowing the scope using 『WHERE』.
   - Error Handling:
     - Returns an empty list if no races are found.
3. Update:
   - Function: modifyDriver
   - Updates details of a specific driver identified by `driver_id`.
   - Query:

```
UPDATE drivers
SET f_name = %s, l_name = %s, date_of_birth = %s, nationality = %s, wiki_url = %s
WHERE driver_id = %s;
```

     - Uses 「SET」 to modify specific columns.
     - The 「WHERE」 clause ensures only the targeted driver record is updated, preventing unintended modifications.
   - Error Handling:

- Rolls back the transaction if any database error occurs.
- Returns an error message if the provided `driver_id` does not exist or the query fails.

4. Delete:
   - Function: deleteDriver
   - Query:

```
DELETE FROM drivers
WHERE driver_id = %s;
```

   - Removes a driver by driver_id.
   - Error Handling:
     - Verifies the existence of the driver before deletion.
     - Prevents accidental deletions by using specific `WHERE` clauses.

2. Detailed Function Descriptions

1. Login/Authentication:
   - Validates user credentials using hashed passwords.
   - Maintains session state for logged-in users.
2. Driver Search:
   - Dynamically builds SQL queries based on input parameters.
   - Query:

```
SELECT driver_id, f_name, l_name, date_of_birth, nationality, wiki_url
FROM drivers
WHERE LOWER(f_name) LIKE %s AND LOWER(l_name) LIKE %s;
```

     - Rationale: Allows flexible, case-insensitive searching.
   - Exception Handling:
     - Ensures empty fields don't cause errors.
3. Race Details by Year:
   - Fetches race details for a given year.
   - Returns data in JSON format for easy frontend integration.
4. Comment System:
   - Users can add comments linked to their accounts.
   - Query: `INSERT INTO comments (username, text) VALUES (%s, %s);`
     - Includes feedback for invalid input or missing session.

5. Inspect Driver:
    - Fetches detailed information about a specific driver by 「driver_id」 and allows actions like modify or delete.
    - Query:

```
SELECT * FROM drivers WHERE driver_id = %s;
```

        - Rationale: Retrieves all driver details, ensuring precise identification using 「driver_id」.
        - Parameterized Query: Protects against SQL injection.
    - Additional Actions:
        1. Delete Driver:
            - Rationale: Safely removes the driver identified by 「driver_id」.
            - Error Handling:
                - Rolls back the transaction if an error occurs and flashes an error message.
        2. Modify Driver:
            - Rationale: Allows updating specific details while maintaining data integrity.
            - Error Handling:
                - Checks if the driver exists (`if not driver`) and provides user feedback if not found.
                - Validates user inputs for modification, ensuring all fields are completed.
                - Includes rollback mechanisms for transactional errors.

Others

1. Repository:

2. Project Video:

3. Progress:

- Expected vs. actual timeline detailed in milestones (e.g., schema design,

functionality implementation).

- Challenges faced: Delays in integrating filtering functionality, resolved through team collaboration and extended deadlines.

4. Team Contributions: