Project Report: Database Design for Ride-Hailing Company

1. Introduction

This report outlines the logical database design for a ride-hailing company, similar to Uber. The primary objective is to create a robust, scalable database management system to efficiently manage the large volumes of data generated by the company's operations. This database will be integral to handling trip records, user information, and service dispatches, supporting both day-to-day operations and strategic decision-making.

2. Logical Database Design

2.1 Overview

The logical database design phase focuses on developing a data model that accurately represents the ride-hailing system's entities, relationships, and constraints. This model abstracts physical storage concerns, ensuring the database structure is optimized for performance, scalability, and ease of maintenance.

2.2 Key Entity Types and Relationships

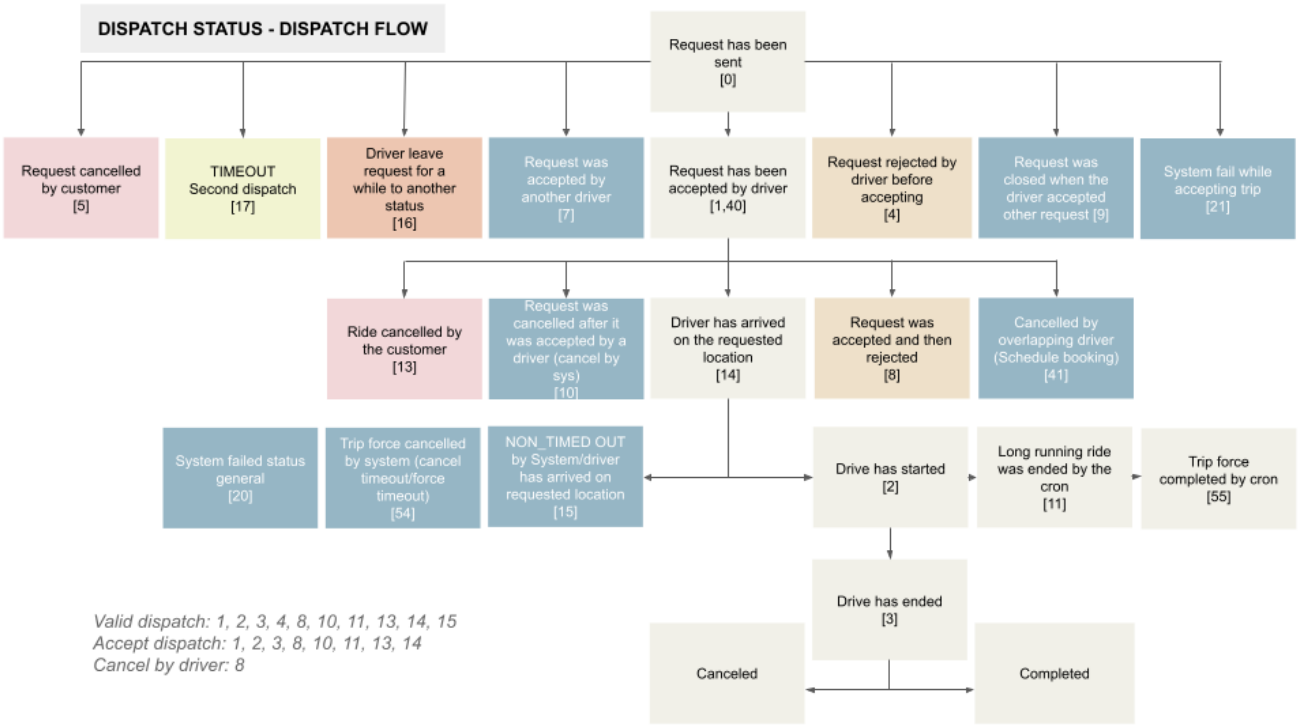
The logical data model for the ride-hailing company comprises several key entities:

- **Requests**: Captures the details of ride requests.
  - **Primary Key**: `request_id`
  - **Attributes**: `request_created_at`, `service_type`, `rider_id`, `city_id`, `priority`
- **Users**: Represent the drivers and riders.
  - **Primary Key**: `user_id`
  - **Attributes**: `name`, `email`, `role`, `password`
- **Dispatches**: Links ride requests to drivers, managed by dispatchers.
  - **Primary Key**: `dispatch_id`
  - **Attributes**: `request_id`, `driver_id`, `dispatcher_id`, `dispatch_time`

2.3 Relationships and Integrity Constraints

- **One-to-Many (1:M) Relationships**:
  - **Requests** ↔ **Dispatches**: Each ride request can result in multiple dispatches, but each dispatch is associated with only one request.

    - **Dispatches ↔ Users**: A dispatch involves one driver, but a driver can handle multiple dispatches.
- **Many-to-One (M:1) Relationship**:
    - **Requests ↔ Users**: Each request is made by a single user (rider), but each rider can make multiple requests.
- **Integrity Constraints**:
    - **Entity Integrity**: Ensures every table has a unique primary key.
    - **Referential Integrity**: Maintains valid foreign key relationships between tables, ensuring data consistency across the database.



2.4 Normalization Process

The normalization process is applied to the database architecture to enhance data integrity and minimize redundancy, adhering to the following principles:

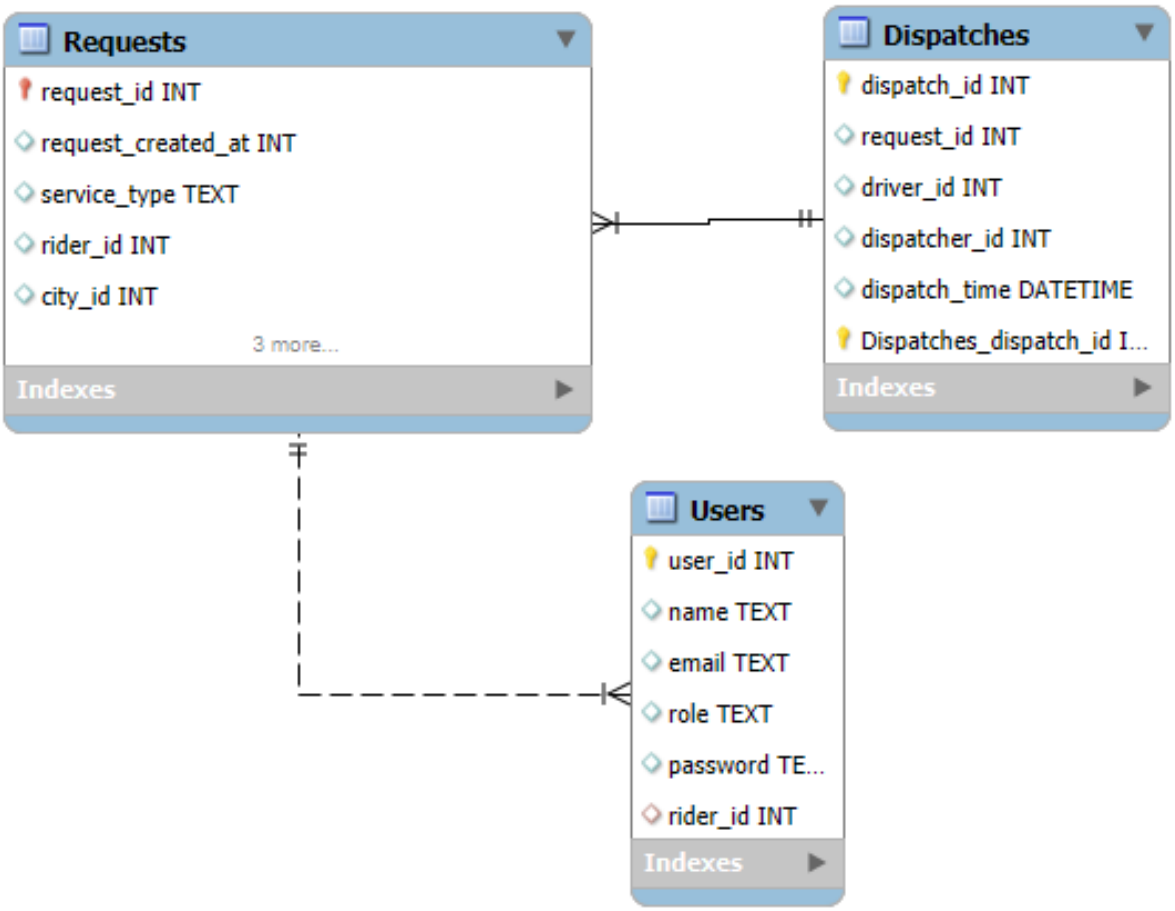
- **1NF - First Normal Form:** Affirms that every column in the table consists of singular, indivisible values, ensuring the uniqueness of each record.

- **2NF - Second Normal Form:** Eliminates dependencies on only part of the main identifier, ensuring that all columns not part of this key depend entirely on the complete identifier.

- **3NF - Third Normal Form:** Removes indirect dependencies, ensuring that attributes outside the primary key depend only on the primary key itself, without relying on other non-primary key columns.

2.5 Entity-Relationship Diagram (ERD)

An ERD was developed to visually represent the logical data model, encompassing entities, primary keys, foreign keys, attributes, and their relationships. The ERD serves as a blueprint for the physical database implementation, ensuring a clear and well-structured database design. MySQL Workbench was used to create the diagram, and forward engineering was employed to generate the script (see Appendix A).



3. Database Build Proposal

3.1 DBMS Selection

PostgreSQL is recommended as the DBMS for this project due to its open-source nature, advanced features, and strong support for complex queries and large-scale applications. PostgreSQL's robustness, scalability, and support for JSON data types make it ideal for managing the diverse data requirements of a ride-hailing platform.

3.2 Database Model Design

The proposed database structure includes the following key tables:

- **Users**: Stores details of both drivers and riders.

    - **Key Attributes**: `user_id`, `name`, `email`, `role`, `password`
- **Requests**: Records information related to ride requests.
    - **Key Attributes**: `request_id`, `request_created_at`, `service_type`, `rider_id`, `city_id`, `priority`
- **Dispatches**: Logs dispatch actions linking requests to drivers.
    - **Key Attributes**: `dispatch_id`, `request_id`, `driver_id`, `dispatcher_id`, `dispatch_time`

3.3 Cloud Hosting and Scalability

To support the system's scalability and availability needs, the database will be hosted on AWS RDS. This cloud-based platform offers automatic backups, failover support, and easy scaling, which are crucial for maintaining service continuity and handling fluctuating workloads typical in a ride-hailing business.

3.4 Security and Access Control

Given the sensitivity of user data, stringent security measures will be implemented, including:

- **Encryption**: Both data at rest and data in transit will be encrypted using industry-standard protocols.
- **Role-Based Access Control (RBAC)**: Access to the database will be controlled through RBAC, ensuring that only authorized personnel can access or modify sensitive information.

4. Data Management Pipeline

4.1 Data Capture

The data management pipeline begins with the capture of data from multiple sources, including user applications, GPS devices, and transactional systems. Real-time data ingestion ensures that the database remains updated with minimal delay, facilitating real-time analytics and informed decision-making.

4.2 Data Cleaning and Preparation

Data cleaning is a critical step in ensuring data accuracy and consistency. Key processes include:

- **Deduplication**: Eliminating duplicate records to ensure data uniqueness.
- **Normalization**: Standardizing data formats to maintain consistency across the database.
- **Error Correction**: Identifying and correcting inaccuracies in the data.

These processes are automated through SQL scripts and data processing tools, ensuring efficient and reliable data preparation.

4.3 Data Validation and Transformation

Post-cleaning, data is validated against business rules to ensure it meets the necessary standards. Data transformation processes are then applied to convert the cleaned data into formats suitable for downstream analytics, reporting, and machine learning tasks.

4.4 Continuous Improvement

The data management pipeline is designed to evolve with the business. Continuous monitoring and iterative improvements ensure the pipeline remains efficient, accommodating growing data volumes and changing business requirements.

5. Critical Evaluation of the Data Management Pipeline

5.1 Strengths

- **Data Quality**: The rigorous data cleaning and validation processes significantly enhance data reliability.
- **Scalability**: The use of PostgreSQL and cloud hosting ensures the system can scale efficiently as the business grows.
- **Security**: The implementation of robust security measures protects sensitive user data, maintaining user trust and regulatory compliance.

5.2 Challenges

- **Data Integration**: Initial challenges in integrating data from diverse sources were overcome through the development of custom integration scripts.
- **Real-Time Processing**: Ensuring efficient real-time processing requires ongoing optimization of the data pipeline, particularly as data volumes increase.

5.3 Recommendations

- **Automated Monitoring**: Implementing automated monitoring tools can provide real-time insights into the performance of the data pipeline, allowing for proactive issue resolution.
- **Regular Audits**: Conduct regular audits of the data and pipeline processes to ensure ongoing compliance with data governance standards and to identify opportunities for further improvement.

6. Conclusion

The logical database design and build proposal outlined in this report provide a solid foundation for the ride-hailing company’s data management needs. With PostgreSQL as the DBMS, cloud hosting for scalability, and a carefully designed data management pipeline, the database system is well-equipped to handle current operations while supporting future growth.

Continuous monitoring and iterative improvements will ensure the system remains robust, secure, and adaptable in a dynamic business environment.

References

Quvvatov, B. (2024). *SQL Databases and Big Data Analytics: Navigating the Data Management Landscape*. Available at: http://www.econferences.ru/index.php/dptms/article/view/11708 [Accessed 21 August 2024].

Silberschatz, A., Korth, H.F., & Sudarshan, S. (2019). *Database System Concepts*. 7th edn. McGraw-Hill Education.

Connolly, T., & Begg, C. (2014). *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson Education Limited.

Ponniah, P. (2010). *Data Modeling Fundamentals: A Practical Guide for IT Professionals*. Wiley.

Elmasri, R., & Navathe, S.B. (2016). *Fundamentals of Database Systems*. 7th edn. Pearson.

IBM (2010). *What is a Database Management System?* [online]. Available at: https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-database-management-system [Accessed 22 August 2024].

Vassiliadis, P.A., Simitsis, A., & Skiadopoulos, S. (2002). *Conceptual Modeling for ETL Processes*. In *21st International Conference on Conceptual Modeling*. Springer, pp. 14-21.

Woodie, A. (2019). *Data Pipeline Automation: The Next Step Forward in DataOps*. Available at: https://www.datanami.com/2019/04/24/data-pipeline-automation-the-next-step-forward-in-dataops/ [Accessed 21 August 2024].

Zhang, Y., & Pan, F. (2022). *Design and Implementation of a New Intelligent Warehouse Management System Based on MySQL Database Technology*. Available at: https://www.informatica.si/index.php/informatica/article/view/3348 [Accessed 21 August 2024].

Appendix

```sql
-- MySQL Script generated by MySQL Workbench
-- Sat Aug 31 11:02:49 2024
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS = @@UNIQUE_CHECKS, UNIQUE_CHECKS = 0;
SET @OLD_FOREIGN_KEY_CHECKS = @@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS = 0;
SET @OLD_SQL_MODE = @@SQL_MODE,
    SQL_MODE = 'ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `mydb`;

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8;
USE `mydb`;

-- -----------------------------------------------------
-- Table `mydb`.`Dispatches`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Dispatches`;

CREATE TABLE IF NOT EXISTS `mydb`.`Dispatches` (
  `dispatch_id` INT NOT NULL,
  `request_id` INT NULL,
  `driver_id` INT NULL,
  `dispatcher_id` INT NULL,
  `dispatch_time` DATETIME NULL,
  `Dispatches_dispatch_id` INT NOT NULL,
  PRIMARY KEY (`dispatch_id`, `Dispatches_dispatch_id`)
) ENGINE = InnoDB;

CREATE UNIQUE INDEX `dispatch_id_UNIQUE` ON `mydb`.`Dispatches` (`dispatch_id` ASC) VISIBLE;
CREATE UNIQUE INDEX `request_id_UNIQUE` ON `mydb`.`Dispatches` (`request_id` ASC) VISIBLE;
CREATE UNIQUE INDEX `driver_id_UNIQUE` ON `mydb`.`Dispatches` (`driver_id` ASC) VISIBLE;

-- -----------------------------------------------------
-- Table `mydb`.`Requests`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Requests`;

CREATE TABLE IF NOT EXISTS `mydb`.`Requests` (
  `request_id` INT NOT NULL,
  `request_created_at` INT NULL,
  `service_type` TEXT NULL,
  `rider_id` INT NULL,
  `city_id` INT NULL,
  `priority` INT NULL,
  `Requests_request_id` INT NOT NULL,
  `Dispatches_dispatch_id` INT NOT NULL,
  PRIMARY KEY (`request_id`),
  CONSTRAINT `request_id`
    FOREIGN KEY (`request_id`)
    REFERENCES `mydb`.`Dispatches` (`request_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

CREATE UNIQUE INDEX `request_created_at_UNIQUE` ON `mydb`.`Requests` (`request_created_at` ASC) VISIBLE;

-- -----------------------------------------------------
-- Table `mydb`.`Users`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Users`;

CREATE TABLE IF NOT EXISTS `mydb`.`Users` (
  `user_id` INT NOT NULL,
  `name` TEXT NULL,
  `email` TEXT NULL,
  `role` TEXT NULL,
  `password` TEXT NULL,
  `rider_id` INT NULL,
  PRIMARY KEY (`user_id`),
  CONSTRAINT `rider_id`
    FOREIGN KEY (`rider_id`)
    REFERENCES `mydb`.`Requests` (`request_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

CREATE UNIQUE INDEX `password_UNIQUE` ON `mydb`.`Users` (`password` ASC) VISIBLE;
CREATE UNIQUE INDEX `email_UNIQUE` ON `mydb`.`Users` (`email` ASC) VISIBLE;
CREATE UNIQUE INDEX `user_id_UNIQUE` ON `mydb`.`Users` (`user_id` ASC) VISIBLE;
CREATE UNIQUE INDEX `rider_id_UNIQUE` ON `mydb`.`Users` (`rider_id` ASC) VISIBLE;

SET SQL_MODE = @OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS = @OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS = @OLD_UNIQUE_CHECKS;
```

Conducted by

1. *Dinh Khoi Dang* - dinhkhoi.dang.work@gmail.com
2. *Matthew Bowyer* - matthewmbowyer@gmail.com
3. *Adrienn Pajtok* - apajtok@gmail.com