Smart City Database Design Report

1. Introduction

The Smart City Database serves as the central information system for a modern city environment. This database supports multiple city services while ensuring data integrity and efficiency. Citizens can access various services like transportation, education, banking, healthcare, and entertainment through different service providers. The system connects citizens with these services while maintaining consistent data across all interactions.

Each service has dedicated functionality but shares common elements such as citizen information, payment processing, and location data. This integrated approach allows for a seamless citizen experience when using multiple city services while giving service providers the specific tools they need to operate efficiently.

2. Database Design

2.1 Design Pattern

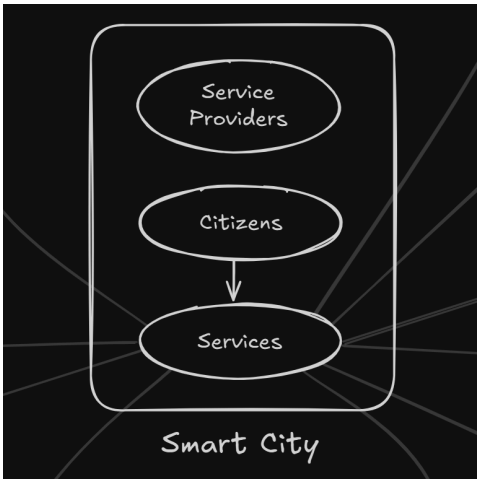
The database follows a service-oriented design pattern that enhances scalability and maintenance. This structure organizes tables around services while sharing common entities. This approach offers several advantages:

- Clear separation between different service domains
- Consistent handling of shared information
- Easy addition of new services without disrupting existing ones
- Simplified relationships between entities

The modular nature of this design allows the city to add or modify services independently. For example, if the transportation system needs updates, those changes can occur without affecting the education or healthcare systems. This flexibility is essential for a growing smart city environment.

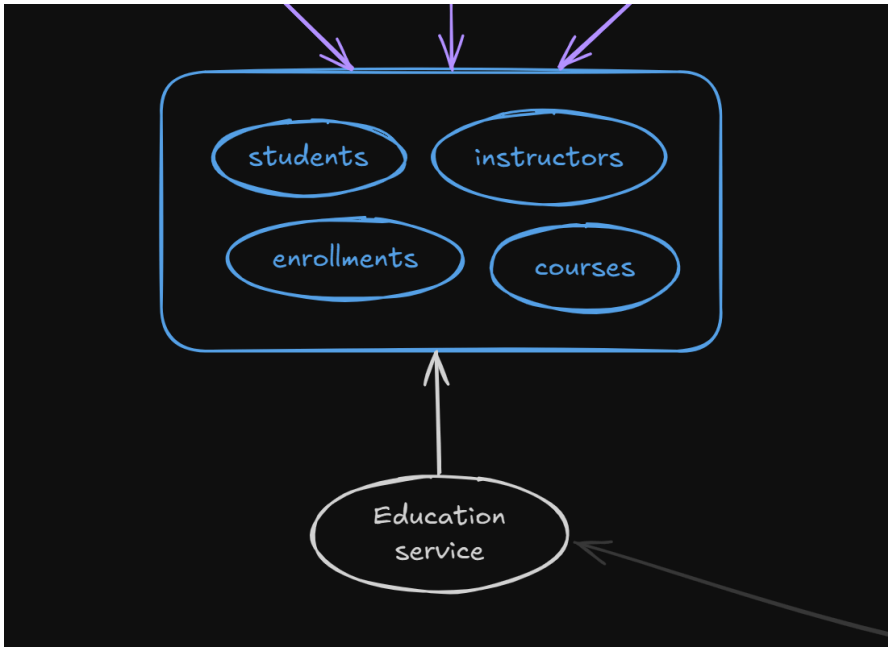
2.2 Conceptual Framework

Core Smart City Concept:



- Smart city providers various services including education, banking, healthcare, transportation, entertainment, etc.
- Citizens can access these services as needed, while these services are provided by different service providers

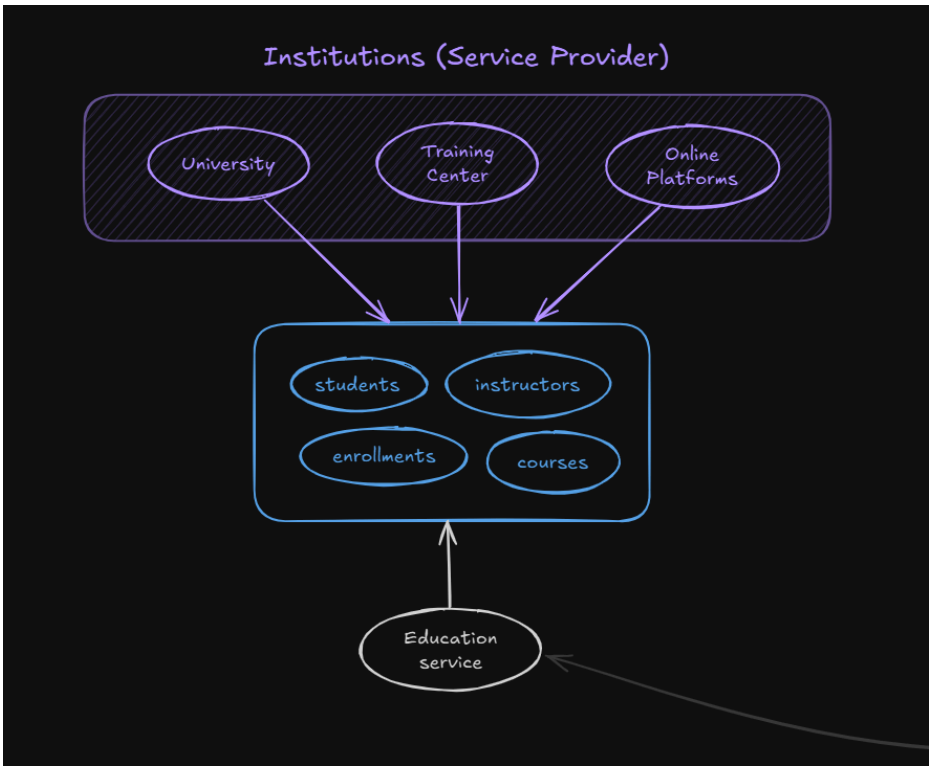
Service Example: Education Service



- The structure of each service is defined by the smart city

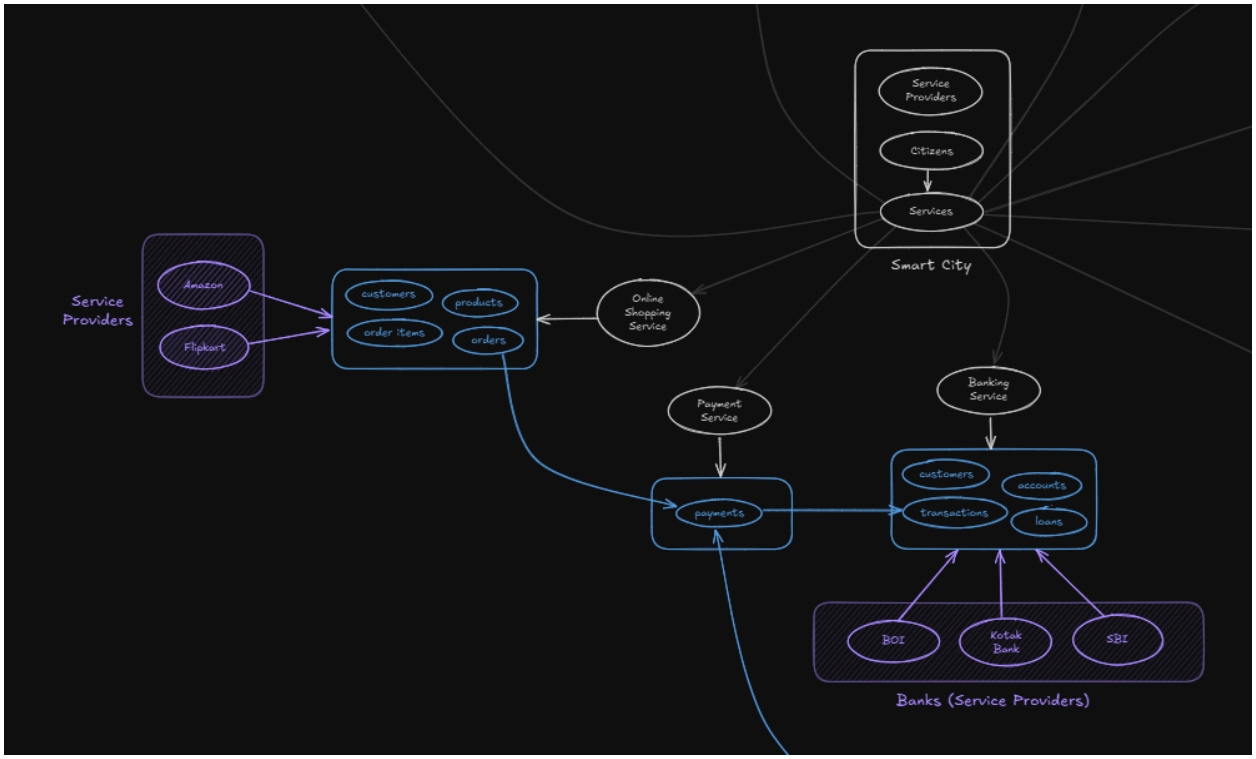
- For the education service, service providers (institutions) work with key entities like students, instructors, enrollments, and courses.
- This standardized structure ensures consistent data handling across different educational institutions.

Service Providers Example:



- Different types of service providers can operate within the same service domain
- For education, providers include universities, training centers, and online platforms. While their business models may differ, they all interact with the same core entities defined by the smart city structure.

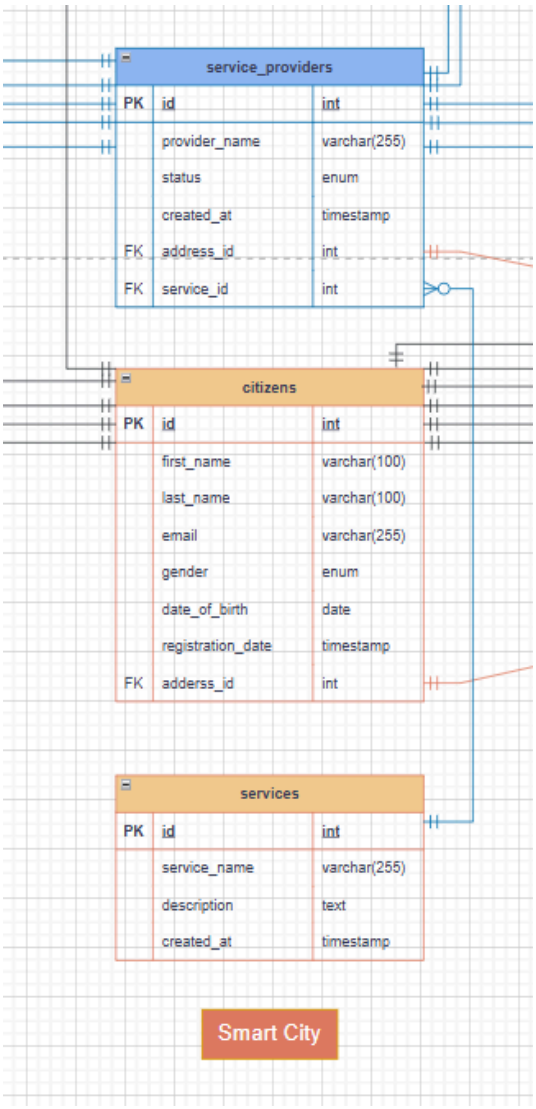
Cross-Service Interactions:



- A key advantage of this approach is that services can interact with each other to implement specific functionality.
- For example, when a citizen makes an online purchase, the shopping service connects with the payment service, which then interacts with the banking service to complete the transaction.
- This integration happens seamlessly while maintaining appropriate separation between domains.

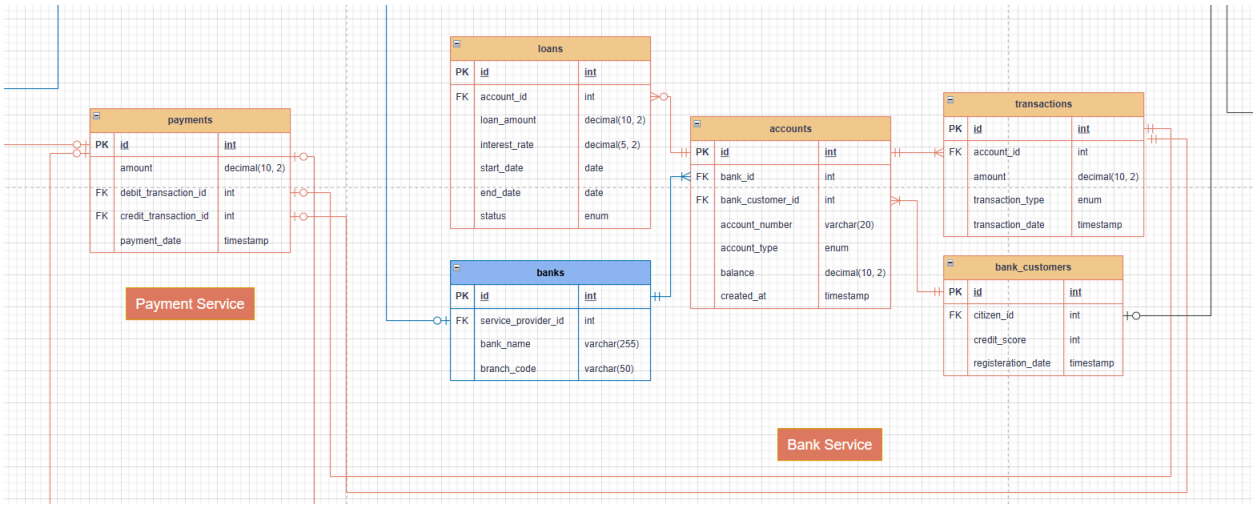
3. Services and Entity Relationships

3.1 Core Entities



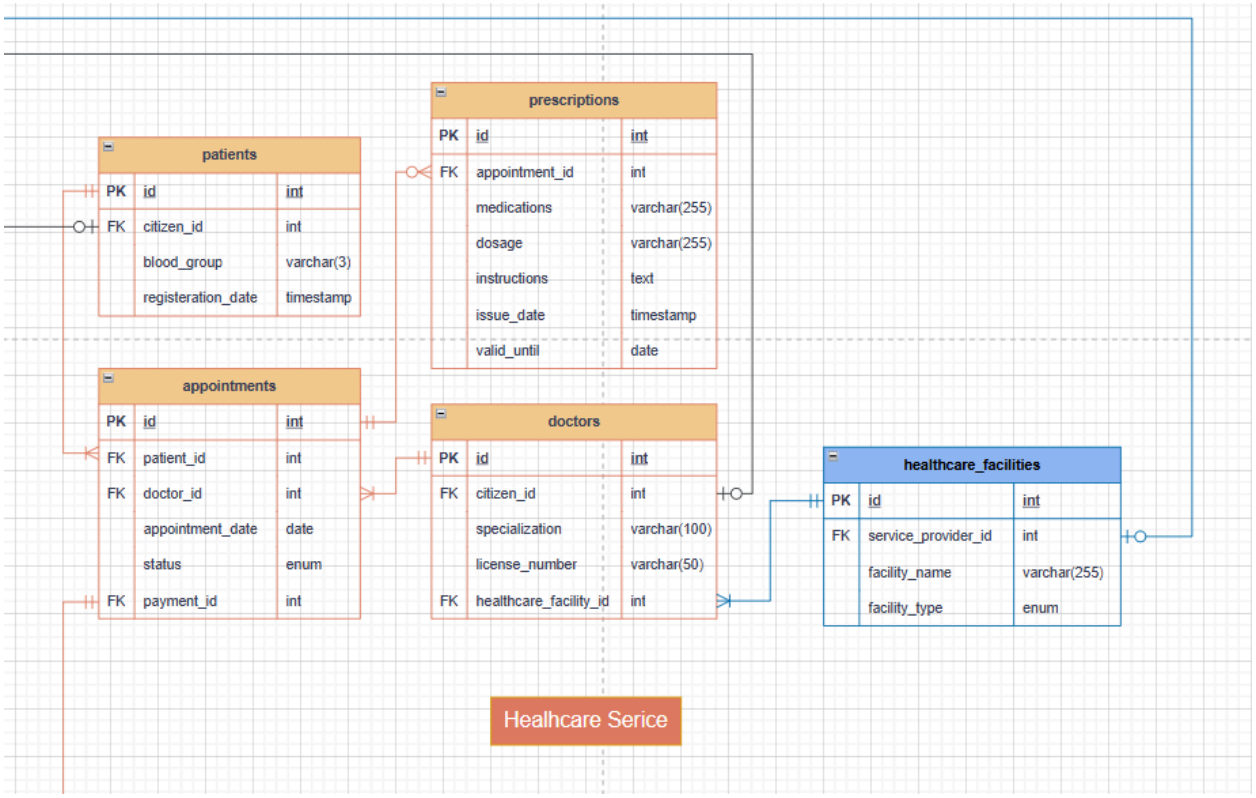
- **`services` ↔ `service_providers`** (One-to-Many): One service can have multiple service_providers; each service_provider belongs to one service.

3.2 Payment and Banking Services



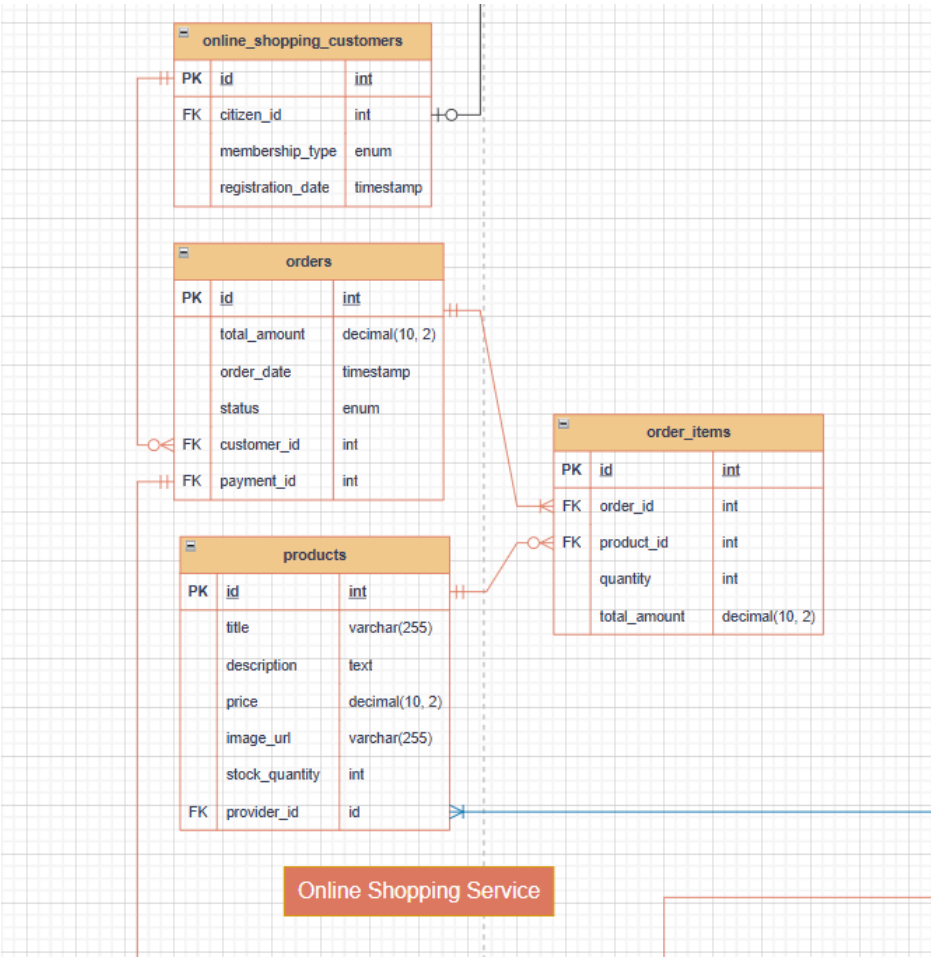
- `bank_accounts` ↔ `citizens` (One-to-One): Each bank account is linked to a unique citizen.
- `bank_accounts` ↔ `transactions` (One-to-Many): One bank account can have multiple transactions; each transaction belongs to one account.
- `transactions` ↔ `payments` (One-to-One): Each transaction corresponds to one payment record.
- `banks` ↔ `service_providers` (Many-to-One): Multiple banks reference one service provider.

3.3 Healthcare Service



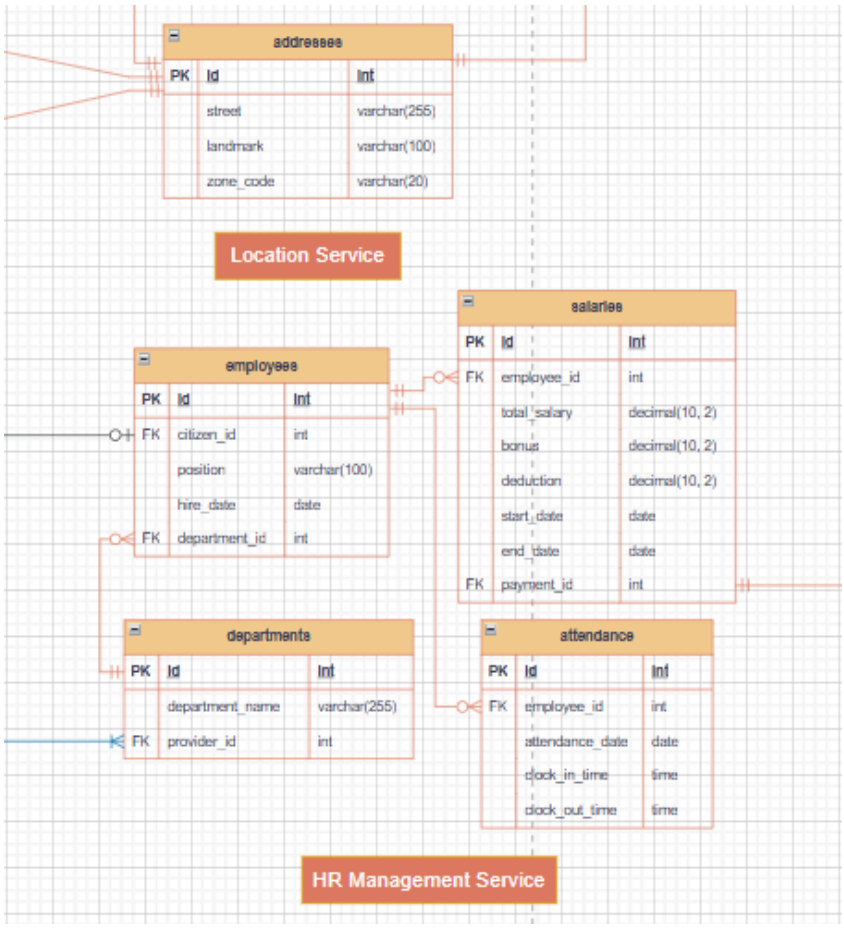
- `patients` ↔ `citizens` (One-to-One): Each patient is linked to a unique citizen
- `patients` ↔ `appointments` (One-to-Many): One user can have multiple appointments; each appointment belongs to one patient.
- `appointments` ↔ `doctors` (Many-to-One): Multiple appointments reference one doctor; each appointment is linked to one doctor.
- `appointments` ↔ `prescriptions` (One-to-One): Each appointment is linked to one prescription.

3.4 Online Shopping Service



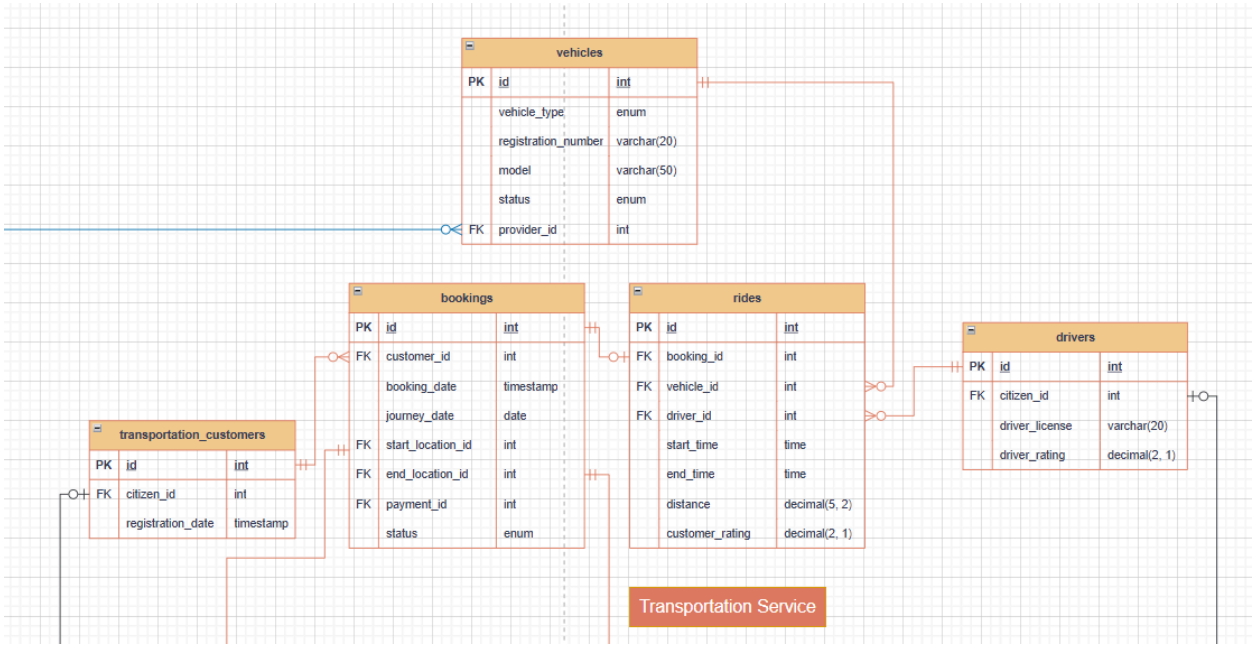
- online_shopping_customers ↔ citizens (One-to-One): Each customer is linked to a unique citizen.
- online_shopping_customers ↔ orders (One-to-Many): One customer can place multiple orders; each order belongs to one customer.
- orders ↔ order_items (One-to-Many): One order can include multiple items; each item belongs to one order.
- order_items ↔ products (Many-to-One): Multiple order items reference one product; each item is linked to one product.
- payments ↔ orders (One-to-One): Each order has one payment record.

3.5 HR Management and Location Services



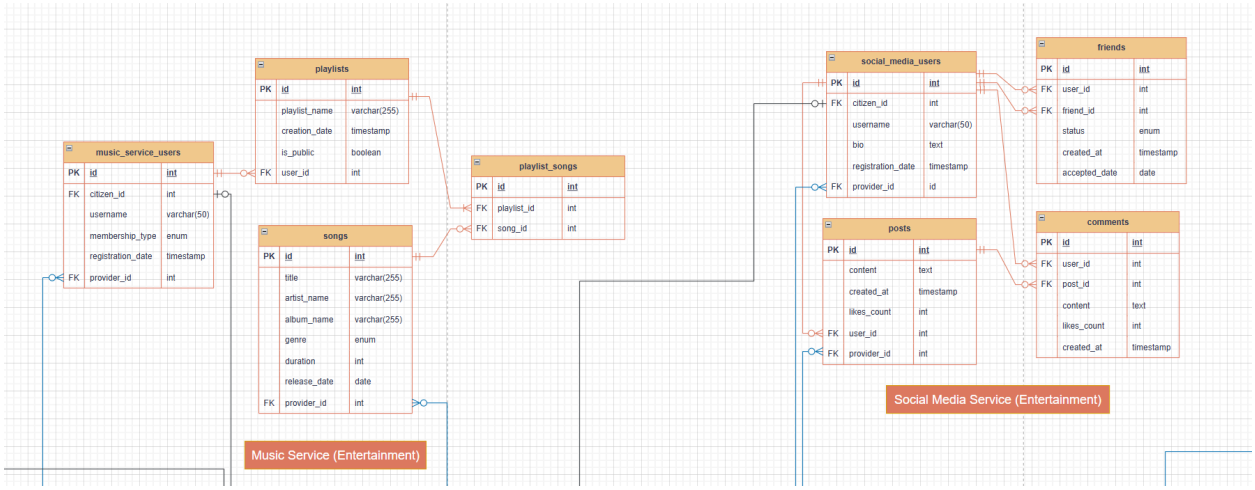
- `employees` ↔ `citizens` (One-to-One): Each employee is linked to a unique citizen.
- `employees` ↔ `departments` (Many-to-One): Multiple employees belong to one department; each department can have multiple employees.
- `employees` ↔ `salaries` (One-to-Many): One employee can have multiple salary records; each record is linked to one employee.
- `payments` ↔ `salaries` (One-to-One): Each salary payment has one payment record.

3.6 Transportation Service



- `transportation_customers` ↔ `citizens` (One-to-One): Each customer is linked to a unique citizen.
- `transportation_customers` ↔ `bookings` (One-to-Many): One customer can create multiple bookings; each booking belongs to one customer.
- `bookings` ↔ `rides` (One-to-One): Each booking is associated with one ride.
- `rides` ↔ `vehicles` (Many-to-One): Multiple rides can involve one vehicle; each ride uses one vehicle.
- `rides` ↔ `drivers` (Many-to-One): Multiple rides can be handled by one driver; each ride has one driver.
- `bookings` ↔ `payments` (Many-to-One): Multiple bookings reference one payment; each booking has a payment record.

3.7 Entertainment Services (Music, Social Media)



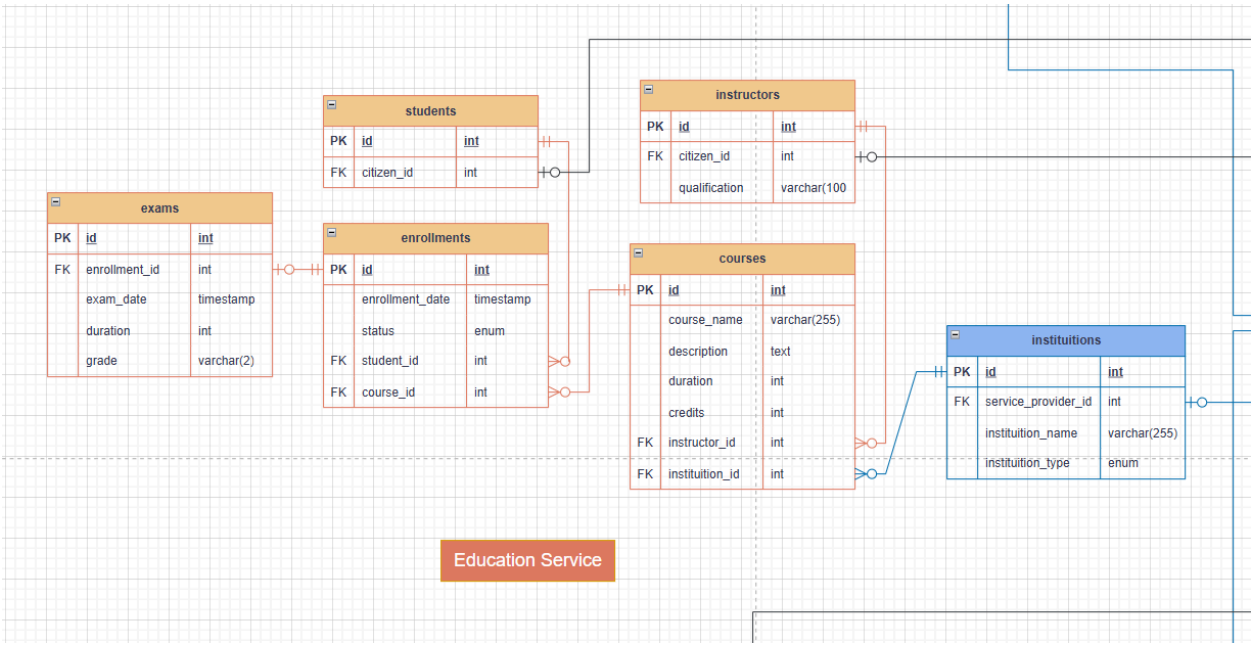
Music Service (Entertainment):

- `music_service_users` ↔ `citizens` (One-to-One): Each user is linked to a unique citizen
- `music_service_users` ↔ `playlists` (One-to-Many): One user can create multiple playlists; each playlist belongs to one user.
- `playlists` ↔ `playlist_songs` (One-to-Many): One playlist can have multiple songs; each record references a single song.
- `songs` ↔ `playlist_songs` (One-to-Many): One song can appear in multiple playlists; each playlist song references one song.
- `songs` ↔ `providers` (Many-to-One): Multiple songs are provided by one provider; each song is linked to one provider.

Social Media Service (Entertainment):

- `social_media_users` ↔ `citizens` (One-to-One): Each user is linked to a unique citizen.
- `social_media_users` ↔ `posts` (One-to-Many): One user can create multiple posts; each post belongs to one user.
- `posts` ↔ `comments` (One-to-Many): One post can have multiple comments; each comment belongs to one post.
- `social_media_users` ↔ `comments` (One-to-Many): One user can write multiple comments; each comment belongs to one user
- `social_media_users` ↔ `friends` (Many-to-Many): Users can have multiple friends; each friend relationship is bidirectional.

3.8 Education Service



- `students` ↔ `citizens` (One-to-One): Each student is linked to a unique citizen.
- `students` ↔ `enrollments` (One-to-Many): One student can have multiple enrollments; each enrollment belongs to one student.
- `enrollments` ↔ `courses` (Many-to-One): Multiple enrollments reference one course; each enrollment is linked to one course.
- `courses` ↔ `instructors` (Many-to-One): Multiple courses reference one instructor; each course is linked to one instructor.
- `instructors` ↔ `citizens` (One-to-One): Each instructor is linked to a unique citizen.
- `courses` ↔ `institutions` (Many-to-One): Multiple courses reference one institution; each course is linked to one institution.
- `institutions` ↔ `service_providers` (One-to-One): Each institution is associated with one service provider.
- `enrollments` ↔ `exams` (One-to-One): Each enrollment has one exam record.

4. Entity Relationship Diagram of Smart City Database