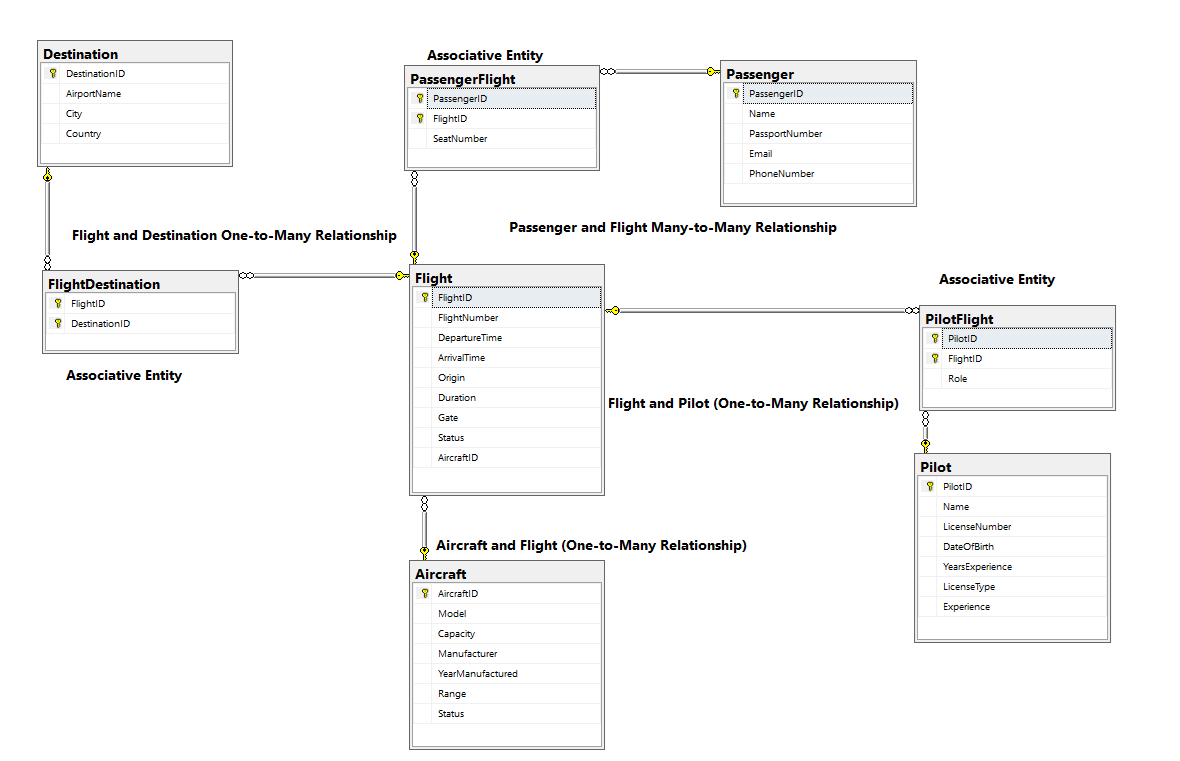
**Flight Database E-R Diagram**



The Entity-Relationship (ER) diagram is structured to fulfill the database requirements for an airline management system based on the following assumptions:

**Assumption A:** Each flight is assigned to only one aircraft, but an aircraft can be assigned to operate multiple flights.

* **Implementation:** This is depicted through a one-to-many relationship between the Aircraft and Flight entities. In the ER diagram, this relationship is indicated by a line that connects the Aircraft entity to the Flight entity, terminating with a crow's foot near the Flight, symbolizing that one aircraft can have multiple associated flights.

**Assumption B:** A pilot can operate several flights, and conversely, a flight may be operated by several pilots.

* **Implementation:** To handle the many-to-many relationship between pilots and flights, an associative entity named PilotFlight is used. This table connects Pilots and Flights through their IDs, allowing multiple pilots to be linked to multiple flights. Each association in the PilotFlight table includes a pilot ID and a flight ID, representing the pilots assigned to each flight.

**Assumption C:** The entities—flight, aircraft, and pilot—must be connected through unique identifiers.

* **Implementation:** Unique identifiers are fundamental in linking these entities. Each entity (Aircraft, Flight, Pilot) has a primary key (AircraftID, FlightID, PilotID) that serves as a unique identifier to maintain referential integrity. These identifiers are crucial for establishing and enforcing the relationships defined between the tables, ensuring that data remains consistent and accessible according to the business logic defined by the airline's operational needs.

By implementing these relationships and structures in the ER diagram, the database design ensures efficient management of data concerning flights, aircraft, and pilots, aligning with the operational requirements and data integrity standards of the airline.

**Normalization:**

The design of this database adheres to normalization principles to ensure data integrity, minimize redundancy, and improve efficiency. Normalization is achieved by organizing the data into multiple tables with appropriate relationships, ensuring the database is in at least **Third Normal Form (3NF)**. Below are examples of how normalization was applied across the database:

**1. First Normal Form (1NF)**

**Definition**: A table is in 1NF if all its columns contain atomic (indivisible) values, and each entry is uniquely identifiable by a primary key.

**Application**: In the design, every table has a primary key (AircraftID, FlightID, PilotID, etc.) that uniquely identifies each record. Additionally, all attributes in each table contain atomic values.

* + For example, the Aircraft table contains atomic attributes such as Model, Capacity, Manufacturer, etc. Each field represents a single piece of information, and there are no repeating groups or multiple values within a single column.

**2. Second Normal Form (2NF)**

**Definition**: A table is in 2NF if it is in 1NF and all non-key attributes are fully dependent on the primary key.

**Application**: Each table’s non-key attributes are fully dependent on the primary key. In cases where tables involve composite keys, such as the associative table PilotFlight, all non-key attributes (like Role) are dependent on the composite key (PilotID, FlightID).

For example, in the Flight table, attributes like FlightNumber, DepartureTime, and AircraftID depend entirely on the FlightID, ensuring no partial dependencies exist. Similarly, in the PilotFlight table, the Role column depends on the combination of PilotID and FlightID.

**3. Third Normal Form (3NF)**

**Definition**: A table is in 3NF if it is in 2NF and all attributes are only dependent on the primary key (i.e., there are no transitive dependencies).

**Application**: The design eliminates transitive dependencies by ensuring that all non-key attributes depend solely on the primary key of their respective tables.

For instance, in the Flight table, attributes such as Gate and Status are directly dependent on the FlightID, not on other non-key attributes like AircraftID. This ensures that any changes to the aircraft (in the Aircraft table) do not affect flight-specific data in the Flight table.

Another example is the Destination table, where attributes such as AirportName, City, and Country are directly dependent on the DestinationID and do not rely on other non-primary key attributes.

**Elimination of Redundancy**

By normalizing the database, we eliminate redundancy. For instance:

**Aircraft Information**: Instead of repeating aircraft details (e.g., Model, Capacity) for every flight, the Aircraft table stores this information once, and the Flight table references it through the AircraftID. This reduces redundant data and ensures consistency.

**Pilot Information**: The same pilot may operate multiple flights. Instead of storing pilot details in the Flight table, these details are stored once in the Pilot table, with the many-to-many relationship handled by the PilotFlight associative table. This ensures that changes to a pilot’s details need to be made in only one place.

**Handling Many-to-Many Relationships**

The many-to-many relationships between entities (like Pilot and Flight, or Flight and Destination) are normalized by introducing associative tables (PilotFlight and FlightDestination). These tables break down the many-to-many relationships into two one-to-many relationships, ensuring the database remains normalized without redundancy.

For example, the PilotFlight table ensures that each flight and pilot combination is recorded without duplicating data in either the Pilot or Flight tables.

**Normalization Across Tables**

Each table focuses on one entity and its attributes, which improves maintainability:

* The Aircraft table focuses on aircraft-specific details.
* The Flight table handles flight-specific details but references the Aircraft and Destination through foreign keys.
* The Pilot table holds data specific to pilots, while the PilotFlight table manages the relationship between pilots and flights.

Through normalization, the design achieves data integrity, reduces redundancy, and ensures the database is flexible for updates or expansions in the future.