Airport Management System

- Team:

1. Vinay Omkumar Khilwani (18BCP126)
2. Vasu Gondaliya (18BCP123)
3. Vaidehi Shah (18BCP121)
4. Umang Mehta (18BCP120)

- <u>Index</u>

Topic:	Page No.:
Experiment-1	3
Experiment-2	10
Experiment-3	19
Experiment-4	22
Experiment-5 to Experiment-10	30
FD of Tables	43
Normalization	47
Uniqueness of Project	49
PL/SQL	50
Conclusion & Future Work	52
References	53

1. Introduction

- Airport

  - Airport is a place that is used for the transport, by air. The transport includes transport of people as well as goods. Its facilities include runways, terminals, control tower. It may also include other facilities like lounges, transport services and emergency services for passenger convenience.

- Airport Management

  - Operating an airport needs proper insight and collaboration because of the increased passenger growth. Also, it requires the perfect time management to make the **right**, **efficient** and **correct** actions at the required and proper time, with the **correct**, **accurate**, **precise** information.

- ## Database Management System

  - A Database Management System (DBMS) includes all the below points:

    a) A cluster of data, which is interrelated.
    b) A group of programs that help in accessing that data.
    c) An IDE (Environment), which is convenient, sufficient and efficient to work on.

  - One single database, which is accessible to everyone, based on their type of accessibility.

- ## File System

  - File System: A system that is supported by the ancient operating system that stores records permanently in various files and uses different applications to extract them records and add new records to the appropriate files. This is the way organisations usually stored information in the olden days.

  - Everyone maintains their own data, and updates (upload/ delete) and extracts data on their system. The change may or may not get updated in other systems.

- <u>Limitations of File System</u>

  - Time Consuming
  - Error-prone Results
  - Less accuracy
  - Time inefficient
  - Uncertainty

- <u>Advantages of DBMS over the Existing System</u>

  - Access from anywhere around the world
  - Access at any time
  - Better Communication

- <u>DBMS v/s. File System</u>

1. **Data Redundancy:**

   - Data redundancy is the duplication of data that occurs in a system, wherein data is redundant unnecessarily, and hence takes up more space and time to access the data. Also it makes the tables complicated. It can also be simply defined as the presence of excessive data, along with the needful data.

   - <u>Example</u>: Let there be a relationship between two tables "Flight A" and "Flight B". The relation: "**B** is a connecting flight of **A**".
     i) Flight A (Name, Passport_Number, Emergency_Contact, Seat_Number-1)

ii) Flight B (Name, Passport_Number, Emergency_Contact, Seat_Number-2)

- All the attributes are same in both the tables, except the "Seat Number". Hence, the data "Name, Passport Number, Emergency Contact" is repeated unnecessarily.

2. Data Inconsistency:

- Data inconsistency is eventually a result of Data redundancy. In other words, the duplicate entries of the redundant data may no longer correspond to each other. That is, in simple terms, if a change is made to any data entry of a particular attribute (which is duplicated in two or more relations), then it is not sure that the same change will also occur in the other relations, where this attribute is duplicated.

- Example: If a person self-checkin's through his mobile phone beforehand, then that seat may not show up as "booked" in the database there at the "check-in" counter at the airport. This type of differing values in duplicate attributes lead to "Data Inconsistency" in "Airport Management System", when using "File Systems".

3. Difficulty in Accessing Data:

- There are many different criterion on which the users want to access the data. Hence, it is very difficult in "File System" to access the data with any criteria. For every criteria, the data-processing team has to write different programs. Hence, it is very difficult and insufficient to access the data in "File System".

- Example: Suppose if a manager of an airline, wants to get the details of the passengers, travelling in "Business Class" in a particular flight. Then, either he has to manually, note the passengers list, from the "passengers" table as a whole, or the other option is that the data-processing team writes a new program for it. This disadvantage isn't faced when we use "DBMS".

4. Data Isolation:

- The data is usually stored among various file systems and hence to access the data, writing a new program every time is essential, which is actually very difficult in the real world.

- Example: Suppose, one airline has its data of flights in a ".xls" format and some other airline has its data in ".csv" format. Then, we will have to write two different application programs, separately for both the airlines, to access the data. Hence, it is insufficient to use "File System" over "DBMS".

5. Integrity Problems:

- There are certain "consistency constraints" that data values must satisfy for them to be stored in the database. The developers assure this by coding the application programs. And when new constraints are added, the problem increases.

- Example: Suppose the date of "travel" in the database, used by passengers is in the format "DD/MM/YY", but however, it has to be changed to "MM/DD/YY". So, for

this change, the data team will face problems while enforcing this one-by-one in all the application programs.

6. **Atomicity:**

- Atomicity means to be or not to be. It is important that if a change is made in the database, and if before it is stored, something goes wrong and the changes made are partial, then either no change should be saved or all the required changes should be made and only then the system should terminate.

- Example: Consider a program to book a ticket at the airline's airport window. Suppose, at the payment page, the payment is done, and at the same time, the system crashes, before the passenger is allotted the seat(s). Hence, in case of the system crash, the change/update should happen entirely or should not happen at all. This is the main problem faced when we deal with "File Systems".

7. **Concurrent Access:**

- Different users may need to access the same file, at a given time. It is necessary that the system gives access to all these users at the same time and then saves all the changes made simultaneously. But, in file system, when different users update the data in the same file, it may happen that the change of one gets overridden by the change, made by the other user.

- Example: If two users try to self-checkin through their mobiles at the same time, for the same seat. It may happen that only the second user gets allotted that seat (actually), but "Seat Allocated" message is shown to both the

passengers. This type of problem will eventually lead to data inconsistency and hence is a major drawback of "File System".

8. **Security Problems:**

- All data cannot be available to all the people due to security constraints. If all the data is available to the people, people might misuse the data and this might lead to thefts and crimes Hence it is very necessary to ensure the security constraints.

- Example: If all the passengers have an access to all the "table/file" of all passengers on some flight, there will be some who will add some passengers to that list, who haven't paid and booked the seat. Also, there may be some passengers who will delete the data of other passengers, in order to get the booking done for his neighbours, colleagues, friends, etc. without any cost at all. Hence, different users should be given different type of access to data, based on their designation and position, which is a messy task in "File System".

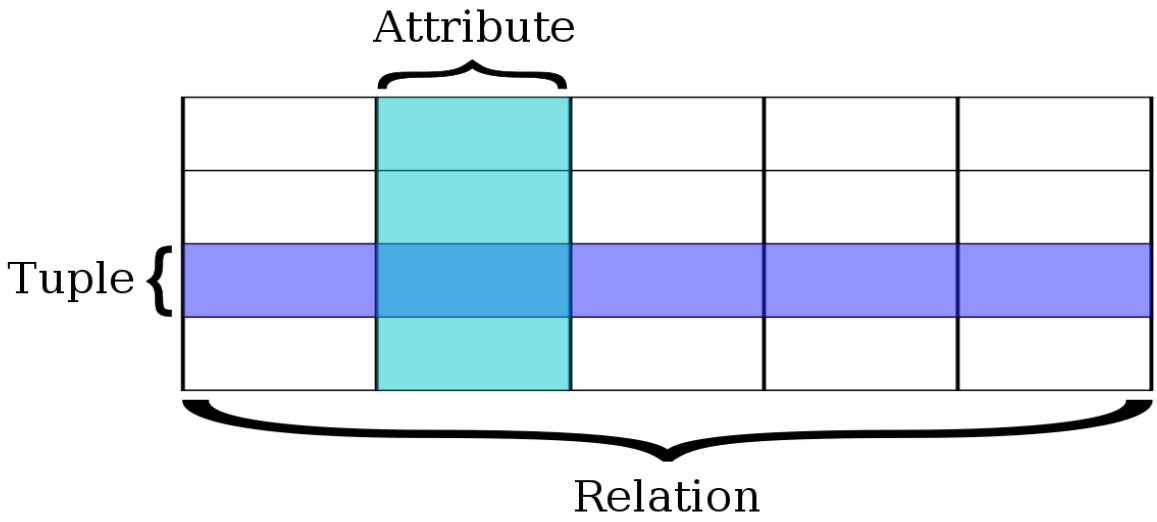
- Due to these huge disadvantages of "File System", the evolvement of "*DBMS*" over "*File System*" has increased exponentially, to manage the tasks in a more proper and efficient manner.

2. Relational Model

- Relational Database

  - A relational database is a collection of tables, all of whose names are unique.

  - Relation: Table.
  - Tuple: Row of a Table.
  - Attribute: Column of a Table.
  - Domain of Attributes: All the values that are allowed to be inserted in the table, according to the constraints.



- ## Schema-Instance Distinction

  - **Database Schema** - The logical layout of the whole database system.
  - **Database Instance** - The data in a database at a given time, corresponds to the temp "instantaneous".

  - Thus, database schema will not change with time while database instance can.

- ## Relations

  - **AIRLINE** (Airline_ID, Name, 3_Digit_Code)
  - **AIRPORT** (Airport_ID, Name)
  - **CITY** (City_ID, Name, City, State)
  - **FLIGHT** (F_Code, Type, Source, Destination, D_Hr, A_Hr, A_Min, D_Min, D_Date, A_Date, AT_No, DT_No)
  - **EMPLOYEE** (Employee_ID, First_Name, Middle_Name, Last_Name, Address_Line_1, Address_Line_2, City, Country, Pin_Code, Salary, Mobile_No, DOB, Job_Type, Gender)
  - **STORES** (Store_ID, Name, Store_Type)
  - **PASSENGER** (Passenger_ID, First_Name, Middle_Name, Last_Name, Address_Line_1, Address_Line_2, City, Country, Pin_Code, Mobile_No, DOB, ID_Type, Gender)
  - **TICKET** (Ticket_Number, Price, Class, Booking_Date, Seat_No)
  - **CONTAINS** (Airport_ID, Airline_ID)
  - **AIRPORT_T** (Airport_ID, T_No, Type)
  - **C_HAS_AP** (City_ID, Airport_ID)
  - **AL_HAS_FL** (Airline_ID, F_Code)
  - **CARRIES** (F_Code, Passenger_ID)
  - **AP_HAS_EMP** (Airport_ID, Employee_ID)

- **A_HAS_S** (Airport_ID, Store_ID)
- **BOOKS** (Ticket_Number, Passenger_ID)
- **CANCELS** (Cancellation_ID, Ticket_Number, Passenger_ID, Date, Charges)

• Keys

- **Primary Key:** It uniquely identifies all the table records due to the fact that:
  1. It has a UNIQUE value for each tuple.
  2. No NULL values entertained.

- A primary key is a column that exists in the table, sometimes following a given sequence.

- **Super Key:** A superkey is a set of single or multiple attributes that help us identify a tuple in a relation uniquely, when taken collectively.

- **Candidate Key:** Minima superkeys are called candidate keys, i.e., all the super keys that do not have a sub part of it that is a super key in itself is a candidate key.

- **Foreign Key:** A foreign key is a set of attribute(s) that is primary key of one of the tables and is referenced in the second table, hence establishing a linkage between the two relations of the schema.

• Foreign Keys in the Relations of our Project

- **CONTAINS:** Airline_ID, Airport_ID
- **C_HAS_AP:** Airport_ID, City_ID
- **AIRPORT_T:** Airport_ID
- **AL_HAS_FL:** Airline_ID, F_Code
- **AP_HAS_EMP:** Airport_ID, Employee_ID
- **A_HAS_S:** Airport_ID, Store_ID
- **CARRIES:** Passenger_ID, F_Code
- **CANCELS:** Ticket_Number, Passenger_ID

• Super Keys in the Relations of our Project

- **AIRLINE:**
  - Airline_ID, Name, 3_Digit_Code
  - Airline_ID, Name
  - Airline_ID, 3_Digit_Code
  - Name, 3_Digit_Code
  - Airline_ID
  - 3_Digit_Code

- **AIRPORT:**
  - Airport_ID, Name
  - Airport_ID

- **CITY:**
  - City_ID, Name, City, State
  - City_ID, City, State
  - City_ID, City, Name
  - City_ID, Name, State
  - City_ID, City
  - City_ID, Name
  - City_ID, State
  - City_ID

- **FLIGHT:**
  - F_Code, Type, Source, Destination, D_Hr, A_Hr, A_Min, D_Min, D_Date, A_Date, AT_No, DT_No
  - F_Code

- **AP_HAS_EMP:**
  - Employee_ID, First_Name, Middle_Name, Last_Name, Address_Line_1, Address_Line_2, City, Country, Pin_Code, Salary, Mobile_No, DOB, Job_Type, Gender
  - Employee_ID

- **STORES:**
  - Store_ID, Name, Store_Type
  - Store_ID, Name
  - Store_ID, Store_Type
  - Store_ID

- **PASSENGER:**
  - Passenger_ID, First_Name, Middle_Name, Last_Name, Address_Line_1, Address_Line_2, City, Country, Pin_Code, Mobile_No, DOB, ID_Type, Gender
  - Passenger_ID

- **TICKET:**
  - Ticket_Number, Price, Class, Booking_Date, Seat_No
  - Ticket_Number

- **CONTAINS:**
  - Airport_ID, Airline_ID

- **AIRPORT_T:**
  - Airport_ID, T_No, Type

- **C_HAS_AP:**
  - City_ID, Airport_ID

- **AL_HAS_FL:**
  - Airline_ID, F_Code

- **CARRIES:**
  - F_Code, Passenger_ID

- **AP_HAS_EMP:**
  - Airport_ID, Employee_ID

- **A_HAS_S:**
  - Airport_ID, Store_ID

- **BOOKS:**
  - Ticket_Number, Passenger_ID

- **CANCELS:**
  - Cancellation_ID, Ticket_Number, Passenger_ID, Date, Charges
  - Cancellation_ID

- <u>Candidate Keys in the Relations of our Project</u>

- **AIRLINE:**
  - Airline_ID
  - 3_Digit_Code

- **AIRPORT:**
  - Airport_ID

- **CITY:**

- City_ID

- **FLIGHT:**
  - F_Code

- **AP_HAS_EMP:**
  - Employee_ID

- **STORES:**
  - Store_ID

- **PASSENGER:**
  - Passenger_ID

- **TICKET:**
  - Ticket_Number

- **CONTAINS:**
  - Airport_ID, Airline_ID

- **AIRPORT_T:**
  - Airport_ID, T_No, Type

- **C_HAS_AP:**
  - City_ID, Airport_ID

- **AL_HAS_FL:**
  - Airline_ID, F_Code

- **CARRIES:**
  - F_Code, Passenger_ID

- **AP_HAS_EMP:**
  - Airport_ID, Employee_ID

- **A_HAS_S:**

- Airport_ID, Store_ID

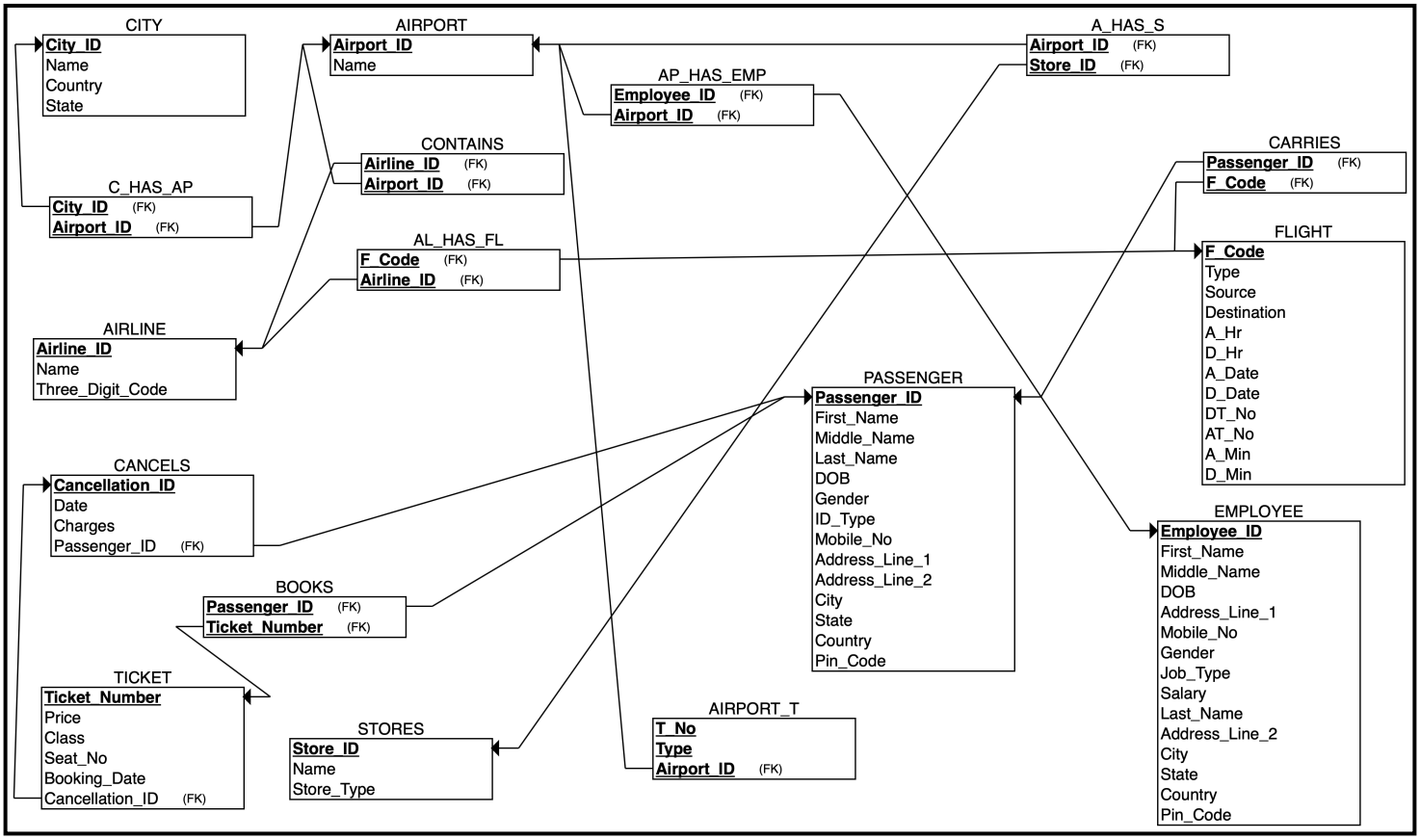
- **BOOKS:**
  - Ticket_Number, Passenger_ID

- **CANCELS:**
  - Cancellation_ID

• Primary Keys in the Relations of our Project

- **AIRLINE:** Airline_ID
- **AIRPORT:** Airport_ID
- **CITY:** City_ID
- **FLIGHT:** F_Code
- **AP_HAS_EMP:** Employee_ID
- **STORES:** Store_ID
- **PASSENGER:** Passenger_ID
- **TICKET:** Ticket_Number
- **CONTAINS:** Airport_ID, Airline_ID
- **AIRPORT_T:** Airport_ID, T_No, Type
- **C_HAS_AP:** City_ID, Airport_ID
- **AL_HAS_FL:** Airline_ID, F_Code
- **CARRIES:** F_Code, Passenger_ID
- **AP_HAS_EMP:** Airport_ID, Employee_ID
- **A_HAS_S:** Airport_ID, Store_ID
- **BOOKS:** Ticket_Number, Passenger_ID
- **CANCELS:** Cancellation_ID

- Relational Diagram



3. Entity-Relationship (ER) Model

- ER Model

  - The E-R model is useful to represent the logical structure of the given database.
  - The E-R model is used to map the real world interactions into a conceptual schema. It contains: entity sets, attributes and relationship sets.

- Entities and Entity Sets

  - Any object in the real world that can be differentiated from all other objects is called entity. It is identified with the help of its properties and behaviours, that is called "attributes" in technical terms.

  - An entity set is a collection of entities that are of the same type and that share the same attributes.

  - Weak Entity: Weak entity is an entity that is dependent on any other entity and does not have a key of its own. Double rectangle represents weak entity.

  - Attribute: Attributes are characteristics of the object or the entity. An eclipse is used to represent an attribute.

- Key Attribute: The main property of the object is given by its key attribute. Key Attributes are represented by underlying lines in the eclipse, and they propose the primary keys of the relation.

- Composite Attribute: When an attribute has its own specifications, or its personal attributes, then it is called composite attributes, i.e., a composition of all the attributes that form an attribute.

• Relationships and Relationship Sets

- A relationship shows how several entities are connected/ associated.

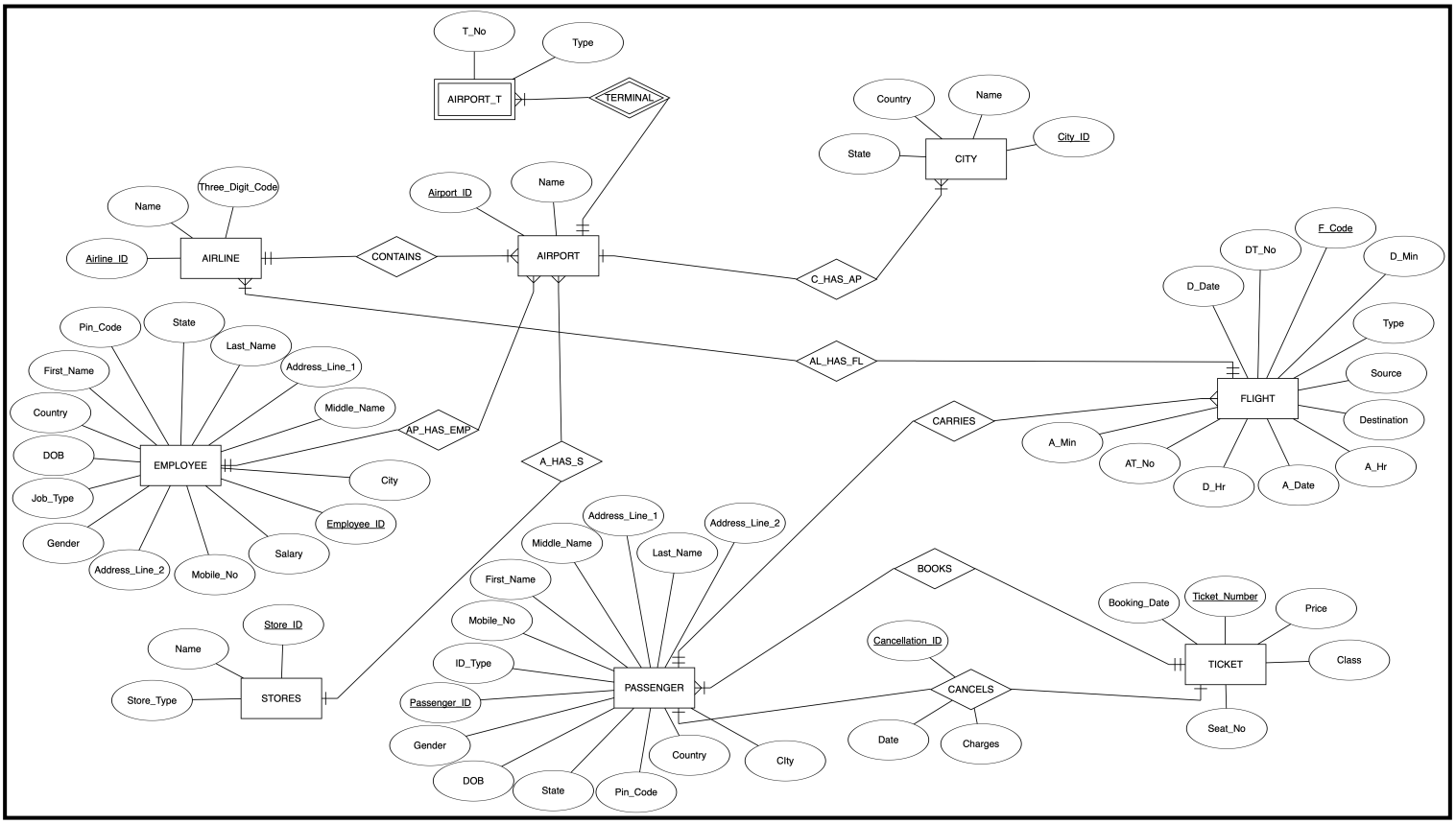
- A relationship set is a set of relationships of the same type. There are three types of relationships that exist between entities:

    1. Binary Relationship: A relation between two entities is called binary relationship. It has the following categories:
        1. One to One
        2. One to Many
        3. Many to One
        4. Many to Many

    2. Recursive Relationship: Self relation is called recursive relation, i.e., when an entity is related to itself.

3. <u>Ternary Relationship</u>: When the degree of a relationship is three, it is called ternary relation.

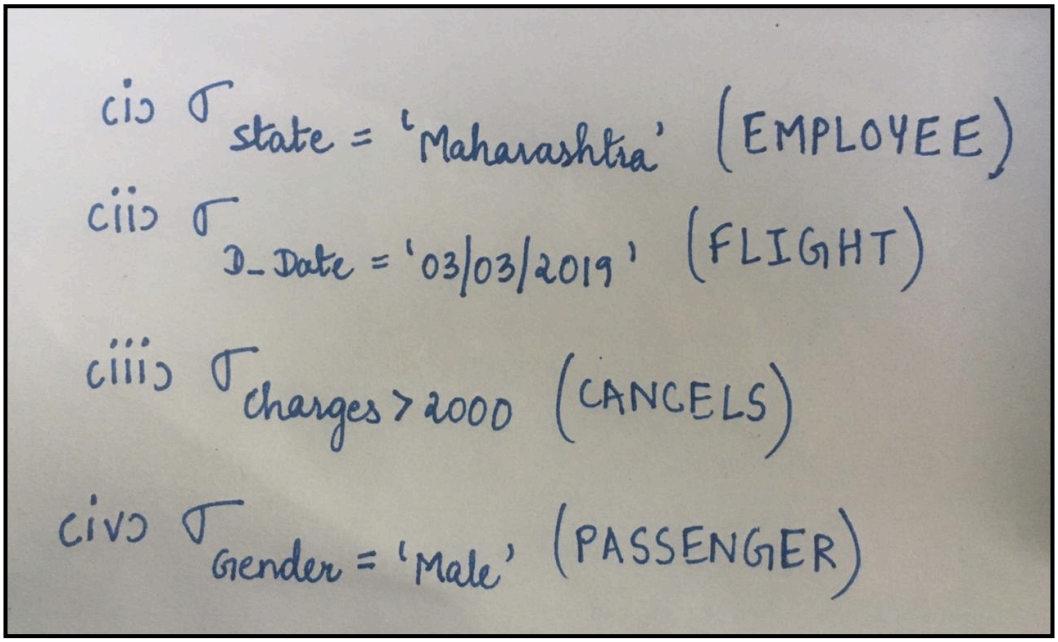
- <u>ER Diagram</u>



4. Relational Algebra Queries

- SELECT: It is used to select the tuples(rows) of a relation, with corresponding conditions.

(i) Select all the employees from "Maharashtra".
(ii) Select all the flights, departing on "03/03/2019".
(iii) Select all the Cancellations, whose Cancellation Charges are greater than "Rs.2000".
(iv) Select all the "Male" passengers.



- PROJECT: It is used to project the corresponding columns of the relations.

(i) Project the mobile number of all the employees.
(ii) Project the names of all the stores.
(iii) Project the Arrival Time of all the flights on "15/02/2019".
(iv) Project the DOB of all the passengers.

(i) $\pi_{Mobile_No}(EMPLOYEE)$

(ii) $\pi_{Name}(STORES)$

(iii) $\pi_{A_Time}(\sigma_{A_Date = '15/02/2019'}(FLIGHT))$

(iv) $\pi_{DOB}(PASSENGER)$

- <u>CARTESIAN PRODUCT</u>: The pairing of the tuples of one relation with each and every row of another relation is called cartesian product.

(i) Project the "Employee_ID" and "Passenger_ID", for all the employees, whose last name and cities are same.
(ii) Project the "Employee_ID" and "Salary", of the employees, who have travelled at least once in any airplane.
(iii) Project the F_Code Pairs whose "Destination" and "D_Date" are same.
(iv) Project the Sources of the pair of flights, which depart from and arrive to the same terminals.

(i) $\pi_{\text{EMPLOYEE_ID, PASSENGER_ID}}\left(\sigma_{\text{EMPLOYEE.Last_Name = PASSENGER.Last_Name} \wedge \text{EMPLOYEE.City = PASSENGER.City}}\left(\text{EMPLOYEE} \times \text{PASSENGER}\right)\right)$

(ii) $\pi_{\text{Salary, Employee_ID}}\left(\sigma_{\text{Employee_ID = Passenger_ID}}\left(\text{EMPLOYEE} \times \text{PASSENGER}\right)\right)$

(iii) $\pi_{\text{A.F_Code, B.F_Code}}\left(\sigma_{\text{A.Destination = B.Destination} \wedge \text{A.DDate = B.D_Date}}\left(\rho_A(\text{FLIGHT}) \times \rho_B(\text{FLIGHT})\right)\right)$

(iv) $\pi_{\text{A.Source, B.Source}}\left(\sigma_{\text{A.AT_No = B.AT_No} \wedge \text{A.DT_No = B.DT_No} \cancel{\wedge} \cancel{\text{A.Destination = B.Destination} \wedge \text{A.Source = B.Source}}}\left(\rho_A(\text{FLIGHT}) \times \rho_B(\text{FLIGHT})\right)\right)$

- <u>UNION</u>: It is used to add the data of two or more different Relational Algebra queries, without duplicates.

(i) Project the "Employee_ID" of all the Employees, whose salary is more than "10000" or they are an "Engineer".
(ii) Project the "Passenger_ID" for all the Passengers, whose ID_Type is "Aadhar" or they are "Male".
(iii) Project the "F_Code" of all the Flights, whose type is "Domestic" or their Destination is "DEL1".
(iv) Project the "Passenger_ID" of all the Passengers, who either are leaving from their own City or are arriving to their own City.

(i) $\pi_{Employee_ID}(\sigma_{Salary > 10000})(EMPLOYEE)$

$\cup \quad \pi_{Employee_ID}(\sigma_{Job_Type = 'Engineer'})(EMPLOYEE)$

(ii) $\pi_{Passenger_ID}(\sigma_{ID_Type = 'Aadhar'})(PASSENGER)$

$\cup \quad \pi_{Passenger_ID}(\sigma_{Gender = 'Male'})(PASSENGER)$

(iii) $\pi_{F_Code}(\sigma_{Type = 'Domestic'}(FLIGHT)) \cup$

$\pi_{F_Code}(\sigma_{Destination = 'DEL1'}(FLIGHT))$

(iv) $\pi_{Passenger_ID}(\sigma_{Source = city}(PASSENGER \bowtie (CARRIES \bowtie FLIGHT))) \cup$

$\pi_{Passenger_ID}(\sigma_{Destination = City}(PASSENGER \bowtie (CARRIES \bowtie FLIGHT)))$

- <u>SET DIFFERENCE</u>: It is used to project the data present in the first Query, but not in the second.

(i) Project the "Passenger_ID" of all the Indian Passengers, who haven't submitted "Passport" as their ID.
(ii) Project the "Employee_ID" of all the Male Employees, who are not from "Chennai".
(iii) Project the "Passenger_ID" of all the Passengers travelling in flight, with "F_Code" as "A01".
(iv) Project the "Employee_ID" of all the Indian Employees, with salary less than "10000".

(ii) $\pi_{Passenger_ID}(\sigma_{Country = 'India'}(PASSENGER)) - \pi_{Passenger_ID}(\sigma_{ID_Type = 'Passport'}(PASSENGER))$

(i) $\pi_{Employee_ID}(\sigma_{Gender = 'Male'}(EMPLOYEE)) - \pi_{Employee_ID}(\sigma_{city = 'Chennai'}(EMPLOYEE))$

(iii) $\pi_{Passenger_ID}(\sigma_{F_code = 'A01'}(\cancel{CANCELS \bowtie} CARRIES \bowtie PASSENGER) - \pi_{Passenger_ID}(\sigma_{F_code = 'A01'}(CANCELS \bowtie CARRIES \bowtie PASSENGER))$

(iv) $\pi_{Employee_ID}(\sigma_{country = 'India'}(EMPLOYEE)) - \pi_{Employee_ID}(\sigma_{salary > 10000}(EMPLOYEE))$

- <u>NATURAL JOIN</u>: It is used to merge the data of two relations, where the common attribute of both the relations has same value.

(i) Project the "Employee_ID" of all the Employees, whose salary is greater than 10000 and are from "Mumbai".
(ii) Project the "F_Code" of all Flights, which will arrive in "Delhi" on "21/12/2018".
(iii) Project the "Name" of all the Stores of "Clothing" Type in any airport of "Delhi".
(iv) Project the "Passenger_ID", who are going back to their "HOME" City, on "31/12/2018".

(i) $\pi_{Employee_ID}\left(\sigma_{salary > 10000 \wedge city.Name = 'Mumbai'}\left(CITY \bowtie AIRPORT \bowtie EMPLOYEE\right)\right)$

(ii) $\pi_{F_Code}\left(\sigma_{_Date = '21/12/2018' \wedge Destination = Airport_ID \wedge CITY.Name = 'Delhi'}\left(CITY \bowtie AIRPORT \bowtie FLIGHT\right)\right)$

(iii) $\pi_{STORES.Name}\left(\sigma_{STORE_Type = 'Clothing' \wedge city.Name = 'Delhi'}\left(CITY \bowtie AIRPORT \bowtie STORES\right)\right)$

(iv) $\pi_{Passenger_ID}\left(\sigma_{PASSENGER.City = CITY.Name \wedge Date = '31/12/2018' \wedge Destination = Airport_ID}\left(CITY \bowtie AIRPORT \bowtie FLIGHT \bowtie PASSENGER\right)\right)$

- <u>ANY 2 of the ABOVE OPERATORS</u>

(i) Project the "Airport_ID" of all Airports, which have more than "1" Terminal.
(ii) Project the "Passenger_ID" of all the Passengers from "India".
(iii) Project the "Salary" and "Employee_ID" of all the Male Employees.
(iv) Project the "Name" of all the "Clothing" Type Stores.

(ii) $\pi_{\text{Airport_ID}}\left(\sigma_{\text{T_No} >= 1}\left(\text{AIRPORT}\right)\right)$

(iii) $\pi_{\text{Passenger_ID}}\left(\sigma_{\text{country} = \text{'India'}}\left(\text{PASSENGER}\right)\right)$

(iii) $\pi_{\text{Salary, Employee_ID}}\left(\sigma_{\text{Gender} = \text{'Male'}}\left(\text{EMPLOYEE}\right)\right)$

(iv) $\pi_{\text{Name}}\left(\sigma_{\text{store_type} = \text{'clothing'}}\left(\text{STORES}\right)\right)$

- <u>ANY 3 of the ABOVE OPERATORS</u>

(i) Project the "Passenger_ID" of all the Male Passengers, travelling in Flight "A01".
(ii) Project the "Employee_ID" of all Employees, at "B00"

(i) $\Pi_{\text{PASSENGER_ID}}\left(\sigma_{\text{gender} = \text{'Male'} \wedge \text{F_code} = \text{'A01'}}\left(\text{PASSENGER} \bowtie \text{CARRIES}\right)\right)$

(ii) $\Pi_{\substack{\cancel{\text{EMP_CODE}} \\ \text{EMPLOYEE_ID}}}\left(\sigma_{\substack{\cancel{\text{F_code}} \\ \text{Airport_ID}} = \text{'B00'}}\left(\text{AP_HAS_EMP} \bowtie \text{EMPLOYEE}\right)\right)$

(iii) $\Pi_{\text{AIRPORT_ID}}\left(\sigma_{\text{city_code} = \text{'ABC'}}\left(\text{AIRPORT} \bowtie \text{C_HAS_AP}\right)\right)$

(iv) $\Pi_{\text{AIRLINE_ID}}\left(\sigma_{\text{airport_ID} = \text{'}\cancel{\text{L98}}\text{'}}\left(\text{CONTAINS} \bowtie \text{AIRLINE}\right)\right)$

MUM1

Airport.
(iii) Project the "Airport_ID" of all the Airports, of City "ABC", as its City_ID.

(iv) Project the "Airline_ID" of all the airlines at Airport "MUM1", as its "Airport_ID".

5. Exp.-5 to Exp.-10

- Some softwares that can be used to run SQL Queries are as follows:

  1. SQuirrel SQL
  2. SQLite Database Browser
  3. HeidiSQL

  and many more…

  - We used "SQLite Database Browser" for our project. It is an open-source application, available for creation Database and evaluation of SQL Queries.

- SQL: It is a computer language, used to manipulate, store and retrieve the data of a database management system.

- DDL: Data definition language or data description language is used for defining

database schemas, similar to syntax used for defining data structures in a computer programming language. Its commands include: CREATE, DROP, ALTER, TRUNCATE, COMMENT, RENAME, etc.

- DML: Data Manipulation Language is used for modifying, deleting and adding data in a database system. Its commands include: INSERT, UPDATE, DELETE, etc.

- DCL: Data control language mainly deals with the controls of the database, i.e., grants or revokes accesses to the database users. Its commands include: GRANT, REVOKE.

- Database and Table Creation: The CREATE TABLE statement is used for creating a table.

- Alteration: Alteration means editing (modifying) a table. ALTER TABLE statement is used to add, delete and modify columns that exist in a table.

- Constraints: The rules, that are enforced on the attributes of a relation to ensure the accuracy, efficiency and reliability of data are called constraints. Constraints can either be

on table level or column level. Most used constraints are : NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, DEFAULT.

- Primary Key Constraint: It is used to uniquely identify each tuple in a relation. All the values for the primary key attribute should be unique and not null.

- Foreign Key Constraint: We need a key to relate two relations, and that key is then, known as the foreign key of one relation, which is actually the primary key of another relation. It is also used to restrict actions that would destroy links between two relations. There are two ways to maintain the integrity of data when a record is deleted in the referencing relation: 1) on delete cascade 2) on delete null

- UNIQUE Constraint: It ensures that all the values for a particular attribute should be unique in that whole relation.

- NOT NULL Constraint: Restricts an attribute of a record (tuple) from having a NULL Value.

- CHECK Constraint: It restricts the value of a column between a range. This is done by condition checking while inserting data.

- DEFAULT Constraint: It is used to provide a default value to a column which will be added to all the records if any other value is not specified during insertion. DEFAULT can be used in ALTER, DROP and CREATE TABLE.

- IN Constraint: It allows to check if a given expression matches any value in a list of values, while insertion of data. If it is not, the SQL application throws an error.

- Aggregate Functions: They are used to produce output, using the aggregated (overall) data of a particular attribute. Example: min, max, sum, avg, etc. The SQL Query example is as follows:

  - Find the avg. cancellation charges on "26/12/2019".

- select avg(charges)
  from cancels
  where date= ' 26/12/2019 '

- Built-in Functions: They are the functions, which make our work easy. Examples include: **Date** Function, **Numeric** Functions, **String** Functions, etc.

  - Numeric Functions: They return numeric values. Examples include, RAND, ROUND, etc.
  - Date function: It is used to fix the format of the date in the attribute, which is going to contain a date.
  - String Functions: They are used to manipulate Strings.

- Set Operations:

  a) Union: It is used to combine the result of two or more results, without duplicates.
  b) Union All: It does the some function as Union, but also prints the duplicate tuples.

c) Intersect: It is used to print the common data from two or more queries.
d) Minus: It is similar to Set Difference (Relational Algebra Operator).

- Sub-queries: Queries inside a query.

- Correlated Sub-queries: It is used to select data from a table, referenced in the outer query.

- Group by: It is used to group a number of tuples (>=1) into a single block.

- Having: It is used to select the tuples, satisfying the corresponding condition of "having" clause.

- Order by: It is used to order the output of a query in ascending or descending order, using one or a set of attributes of the used relations.

- Join operations: Inner Join, Outer Join, etc.

- Exists & Not Exists: It gives a boolean output whether a set of attributes are present in the provided sub-query or not.

- Any: It chooses to output "true" when the comparison of two sets of attributes gives "true" for at least one of the comparisons.

- All: It chooses to output "True" only when the comparison of two sets of attributes gives "True" for all comparisons that take place.

- CREATE Commands

```
CREATE TABLE CITY
(
        Name VARCHAR(25) NOT NULL,
        Country VARCHAR(25) NOT NULL,
        State VARCHAR(25) NOT NULL,
        City_ID CHAR(5) NOT NULL,
        PRIMARY KEY (City_ID)
);

CREATE TABLE AIRPORT
(
        Airport_ID CHAR(4) NOT NULL,
        Name VARCHAR(70) NOT NULL,
        PRIMARY KEY (Airport_ID)
);
```

```
CREATE TABLE EMPLOYEE
(
	First_Name VARCHAR(25) NOT NULL,
	Middle_Name VARCHAR(25) NOT NULL,
	DOB DATE NOT NULL,
	Employee_ID CHAR(5) NOT NULL,
	Address_Line_1 VARCHAR(75) NOT NULL,
	Mobile_No INT NOT NULL,
	Gender CHAR(1) NOT NULL CHECK(Gender in
		('M','F','O')),
	Job_Type VARCHAR(25) NOT NULL,
	Salary FLOAT NOT NULL,
	Last_Name VARCHAR(25) NOT NULL,
	Address_Line_2 VARCHAR(75) NOT NULL,
	City VARCHAR(25) NOT NULL,
	State VARCHAR(25) NOT NULL,
	Country VARCHAR(25) NOT NULL,
	Pin_Code INT NOT NULL,
	PRIMARY KEY (Employee_ID)
);

CREATE TABLE AIRLINE
(
	Airline_ID CHAR(5) NOT NULL,
	Name VARCHAR(30) NOT NULL,
	Three_Digit_Code CHAR(3) NOT NULL,
	PRIMARY KEY (Airline_ID)
);

CREATE TABLE FLIGHT
(
	F_Code CHAR(4) NOT NULL,
	Type VARCHAR(20) NOT NULL,
	Source CHAR(4) NOT NULL,
	Destination CHAR(4) NOT NULL,
	A_Hr INT NOT NULL CHECK(A_Hr>=0 and A_Hr<=23),
```

```
        D_Hr INT NOT NULL CHECK(D_Hr>=0 and D_Hr<=23),
        A_Min INT NOT NULL CHECK(A_Min>=0 and
                A_Min<=59),
        D_Min INT NOT NULL CHECK(D_Min>=0 and
                D_Min<=59),
        A_Date DATE NOT NULL,
        D_Date DATE NOT NULL,
        DT_No INT NOT NULL,
        AT_No INT NOT NULL,
        PRIMARY KEY (F_Code)
);

CREATE TABLE PASSENGER
(
        First_Name VARCHAR(25) NOT NULL,
        Middle_Name VARCHAR(25) NOT NULL,
        Last_Name VARCHAR(25) NOT NULL,
        DOB DATE NOT NULL,
        Gender CHAR(1) NOT NULL CHECK(Gender in
            ('M','F','O')),
        Passenger_ID CHAR(5) NOT NULL,
        ID_Type VARCHAR(15) NOT NULL,
        Mobile_No INT NOT NULL,
        Address_Line_1 VARCHAR(75) NOT NULL,
        Address_Line_2 VARCHAR(75) NOT NULL,
        City VARCHAR(25) NOT NULL,
        State VARCHAR(25) NOT NULL,
        Country VARCHAR(25) NOT NULL,
        Pin_Code INT NOT NULL,
        PRIMARY KEY (Passenger_ID)
);

CREATE TABLE STORES
(
        Name VARCHAR(30) NOT NULL,
        Store_ID CHAR(5) NOT NULL,
```

```
        Store_Type VARCHAR(30) NOT NULL,
        PRIMARY KEY (Store_ID)
);

CREATE TABLE AIRPORT_T
(
        T_No INT NOT NULL,
        Type VARCHAR(20) NOT NULL,
        Airport_ID CHAR(4) NOT NULL,
        FOREIGN KEY (Airport_ID) REFERENCES
                AIRPORT(Airport_ID) ON DELETE CASCADE
);

CREATE TABLE CONTAINS
(
        Airline_ID CHAR(5) NOT NULL,
        Airport_ID CHAR(4) NOT NULL,
        FOREIGN KEY (Airline_ID) REFERENCES
                AIRLINE(Airline_ID) ON DELETE CASCADE,
        FOREIGN KEY (Airport_ID) REFERENCES
                AIRPORT(Airport_ID) ON DELETE CASCADE
);

CREATE TABLE C_HAS_AP
(
        City_ID CHAR(5) NOT NULL,
        Airport_ID CHAR(4) NOT NULL,
        FOREIGN KEY (City_ID) REFERENCES CITY(City_ID)
                ON DELETE CASCADE,
        FOREIGN KEY (Airport_ID) REFERENCES
                AIRPORT(Airport_ID) ON DELETE CASCADE
);

CREATE TABLE A_HAS_S
(
        Airport_ID CHAR(4) NOT NULL,
```

```
        Store_ID CHAR(5) NOT NULL,
        FOREIGN KEY (Airport_ID) REFERENCES
            AIRPORT(Airport_ID) ON DELETE CASCADE,
        FOREIGN KEY (Store_ID) REFERENCES
            STORES(Store_ID) ON DELETE CASCADE
);

CREATE TABLE AL_HAS_FL
(
        F_Code CHAR(4) NOT NULL,
        Airline_ID CHAR(5) NOT NULL,
        FOREIGN KEY (F_Code) REFERENCES
            FLIGHT(F_Code) ON DELETE CASCADE,
        FOREIGN KEY (Airline_ID) REFERENCES
            AIRLINE(Airline_ID) ON DELETE CASCADE
);


CREATE TABLE CARRIES
(
        Passenger_ID CHAR(5) NOT NULL,
        F_Code CHAR(4) NOT NULL,
        FOREIGN KEY (Passenger_ID) REFERENCES
            PASSENGER(Passenger_ID) ON DELETE
            CASCADE,
        FOREIGN KEY (F_Code) REFERENCES
            FLIGHT(F_Code) ON DELETE CASCADE
);

CREATE TABLE CANCELS
(
        Cancellation_ID CHAR(5) NOT NULL,
        Date DATE NOT NULL,
        Charges FLOAT NOT NULL,
        Passenger_ID CHAR(5) NOT NULL,
        Ticket_Number CHAR(6) NOT NULL,
```

```
        PRIMARY KEY (Cancellation_ID),
        FOREIGN KEY (Passenger_ID) REFERENCES
            PASSENGER(Passenger_ID) ON DELETE
            CASCADE,
        FOREIGN KEY (Ticket_Number) REFERENCES
            TICKET(Ticket_Number) ON DELETE CASCADE
);

CREATE TABLE AP_HAS_EMP
(
        Employee_ID CHAR(5) NOT NULL,
        Airport_ID CHAR(4) NOT NULL,
        FOREIGN KEY (Employee_ID) REFERENCES
            EMPLOYEE(Employee_ID) ON DELETE CASCADE,
        FOREIGN KEY (Airport_ID) REFERENCES
            AIRPORT(Airport_ID) ON DELETE CASCADE
);
CREATE TABLE TICKET
(
        Ticket_Number CHAR(6) NOT NULL,
        Price FLOAT NOT NULL,
        Class VARCHAR(15) NOT NULL,
        Seat_No VARCHAR(4) NOT NULL,
        Booking_Date DATE NOT NULL,
        PRIMARY KEY (Ticket_Number)
);

CREATE TABLE BOOKS
(
        Passenger_ID CHAR(5) NOT NULL,
        Ticket_Number CHAR(6) NOT NULL,
        FOREIGN KEY (Passenger_ID) REFERENCES
            PASSENGER(Passenger_ID) ON DELETE
            CASCADE,
        FOREIGN KEY (Ticket_Number) REFERENCES
            TICKET(Ticket_Number) ON DELETE CASCADE
```

```
);
```

- INSERT Commands (Some Examples)

```
INSERT INTO `Airline`
        (`Airline_ID`,`Name`,`Three_Digit_Code`) VALUES
        (99001,"Hukqasuuof","HTM");
INSERT INTO `Airline`
        (`Airline_ID`,`Name`,`Three_Digit_Code`) VALUES
        (99002,"Wetxocuvas","WHZ");
INSERT INTO `Airline`
        (`Airline_ID`,`Name`,`Three_Digit_Code`) VALUES
        (99003,"Devueuexaa","VOO");
```

6. Functional Dependencies (FD's)

- Passenger (A,B,C,…,N):
  - A -> BCDEFGHIJKLMN
  - I -> JKLM
  - J -> KLM
  - K -> LM
- Employee (A,B,C,…,N,O):
  - A -> BCDEFGHIJKLMN
  - I -> JKLM
  - J -> KLM
  - K -> LM
- Terminal (A,B):
  - A -> A
  - B -> B
- Airport (A,B):
  - A -> B

- Flight (A,B,C,D,E,F,G,H,I,J):

  - A -> BCDEFGHIJ
  - CD -> B

- Cancels (A,B,C):

  - A -> BC

- Ticket (A,B,C,D,E):

  - A -> BCDE
  - D -> C

- Stores (A,B,C):

  - A -> BC

- Airline (A,B,C):

  - A -> BC
  - C -> AB

- City (A,B,C,D):

- A -> BCD
- B -> CD
- D -> C

- <u>C_HAS_AP (A,B)</u>:

  - No FD's

- <u>BOOKS (A,B)</u>:

  - No FD's

- <u>AL_HAS_FL (A,B)</u>:

  - No FD's

- <u>CONTAINS (A,B)</u>:

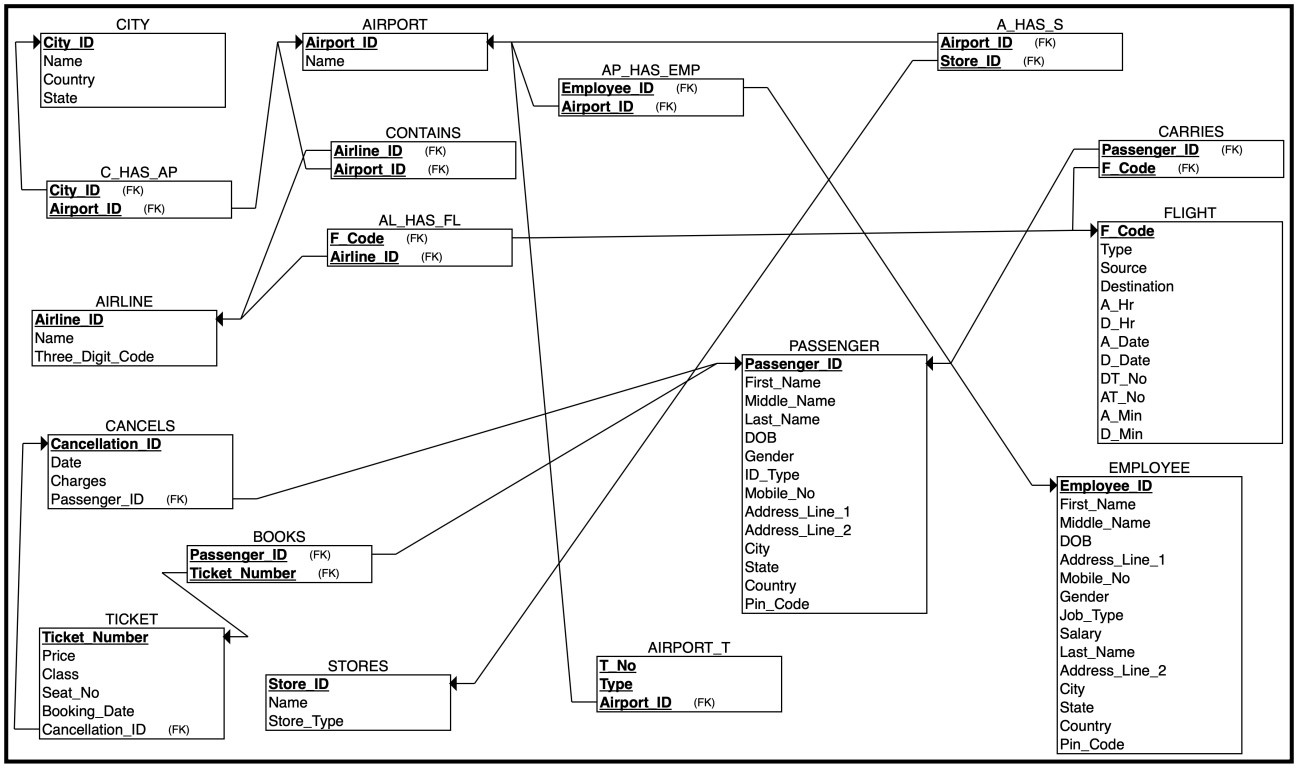
  - No FD's

- <u>A_HAS_S (A,B)</u>:

  - No FD's

- <u>CARRIES (A,B)</u>:
  - No FD's

- <u>AIRPORT_T (A,B,C)</u>:
  - No FD's



7. Normalisation

- For the table to be in 1NF:

  - It should be atomic.
  - Values stored should be of the same type in the same column.
  - All the columns in the relation are to be uniquely named.

- For the table to be in 2NF:

  - It should be in 1NF.
  - No partial dependencies should exist.

- For the table to be in 3NF:

  - It should be in 2NF.
  - No transitive dependencies should exist.

- For the table to be in BCNF:

- It should be in 3NF.
- For every functional dependency (A -> B), 'A' must be a super-key.

- <u>All the FD's of all the relations, described in (6) are already in 1NF, 2NF, 3NF and BCNF.</u>

8. Uniqueness

- The database can be used in the future for traffic predictions at a particular airport and also, vacancy of a flight on a particular day of the year can be predicted, using the available data, in the database.

- Also, we can add the number of store visits by a passenger to a particular store in our database, so that we can get the data about how well the store is functioning.

9. PL/SQL

- PL/SQL is a block structured language that combines the power of SQL with the procedural commands. The processing speed increases because all blocks are passed directly to the compilation engine.

- **Disadvantages of SQL**:

  1. SQL doesn't provide the programmers with a facility to check conditions, or add loops and branches.
  2. SQL has no facility of checking of errors during manipulation and editing of data.

- **Differences between SQL and PL/SQL**:

SQL	PL/SQL
SQL supports single query that is used to perform all the DML and DDL operations.	PL/SQL is a block of codes for functioning of a block of data together.
It is declarative, and defines only what has to be done and not how it has to be done.	It is procedural, and hence defines the process of how things are to be done.
Executes wholly as one statement.	Executes as a whole block of multiple statements.
Mainly used to manipulate data.	Mainly used to create an application.

Cannot contain PL/SQL code in it, but the voice-overs is true.	It can contain SQL statements within the blocks, as it is an extension of SQL.

- **Structure of PL/SQL Block**:

  - PL/SQL is an extension of SQL. It adds the constructs that are used in the procedural language, which results in a more powerful execution. The basic unit of it is the PL/SQL block which can be nested within each other.

- **Example of PL/SQL (Code)**:

  - SET SERVEROUTPUT ON;
  - DECLARE
    ```
    a INTEGER := 5;
    b REAL;
    ```
  - BEGIN DECLARE
    ```
    dbms_output.put_line(a);
    ```
  - END;
  - /

  - The above code will declare two variables, 'a' and 'b' and will print the value of 'a', after that.

10. Conclusion & Future Work

- Every airport requires a management system to function properly, for the public satisfaction. A database management system, for an airport with more future data predictions, reliable data sharing and security. It will also lead to better data integration and access and will increase end user productivity.

- Our Future plans include:

  - Maybe, developing a backend for a passenger-centred application, which extends the use of the data retrieved and modified in the database (past-data).

  - Also, if possible, to submit and publish a research paper on this project, with some minor changes. The use of QR codes on each store of an airport will provide us with the data of number of visits on that store in a particular time period. This visitor count will help us find which stores should be replaced by another good-working stores. And also, some other minor changes.

11. References

- wikipedia.org, (2009). *Airport*. [online] Available at: https://en.wikipedia.org/wiki/Airport [Accessed 26 Nov. 2019].

- tutorialspoint.com, (2017). *PL/SQL Tutorial*. [online] Available at: https://www.tutorialspoint.com/plsql/index.htm [Accessed 26 Nov. 2019].

- geeksforgeeks.org, (2010). *PL/SQL Introduction*. [online] Available at: https://www.geeksforgeeks.org/plsql-introduction/ [Accessed 26 Nov. 2019].