Introduction to Database Systems
Global Flight Information Database

Author:
Csutak Balázs

Consultant:
Gábor Csaba

Semester: 2017/18 fall

Contents

1 Introduction

The aim of this project is designing a database system for storing global flight information.

As air travel became available and affordable for the great public in the past half century, the number of flights and airports shows a tremendous growth. While different type of airliners offers various travelling possibilities, finding the one you really need can be a real pain.

By organizing the mass amount of air traffic related data, this database is primarily intended to give travel agencies and individuals an overview of the system, and help them in planning their routes and picking the flight most suitable for their needs. Nevertheless, data related to capacity utilization can also be useful for airlines to optimize their services.

2 Specification

In order to achieve the goals listed above, the database should store the following information:

Airports: a list of airports available in the database. It is required to store its identification number (IATA and ICAO code), name, location (country, city).

Flights: date, time, and airport of departure and arrival, length (time), airline, aircraft, passengers.

Airlines: airlines offering transportation services between the airports listed above. We should store its ID, name, category (discount or premium), aircrafts they use, employees. We must store data for pilots and for other flight personnel separately. For pilots, we need their rank, number of flight hours, and their pilot license number. To assure, that they can communicate with passengers, for stewardesses spoken languages should be stored.

Aircrafts: ID, manufacturer, model, capacity

Ticket offices: We should store where we can book a ticket for the flights.

3 Entity-relationship model

To meet requirements listed above, we created the system design as seen in the following E-R diagramm:

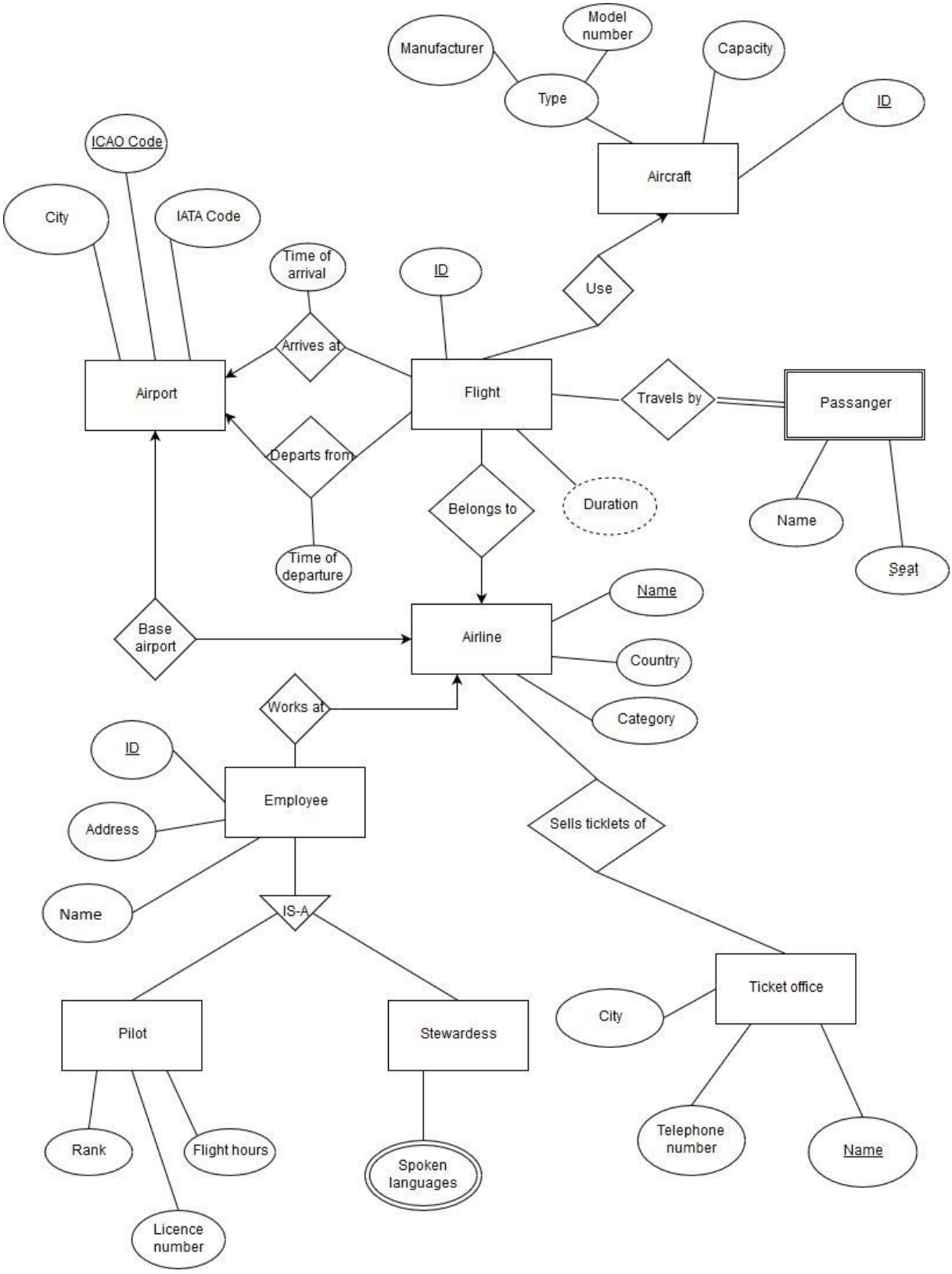

Figure 1: E-R diagram of the database system

4 Relational model

From the E-R model seen in picture 1 we created the following relational model:

Aircraft (ID, manufacturer, model number, capacity)

Airport (ICAO, IATA, name, city)

Flight (ID, airline, departure_time, departure_airport, arrival_time, arrival_airport, aircraft_id)

Passenger (flight_id, seat, name)

Airline (name, country, category, base_airport)

Employee (ID, name, address, airline)

Pilot (employee_id, license number, rank, flight hours)

Stewardess (employee_id, spoken_language)

Ticket Office (name, city, telephone number)

Sell_tickets (airline_id,office_name)

As model mainly contains 1-1 and 1-N relationships, transformation is pretty straightforward. Discussion of nontrivial cases comes below.

Passenger is a weak entity, thus a key of table **Flight** must be also inserted. This way, foreign key *flight_id* and discriminator *seat* together form a primary key.

Table **Aircraft** contains a composed attribute - this is present in the database as two separate attributes of the entity.

Airline and **Ticket Office** have a many-to-many relationship, so separate table must be created for this purpose. This is called **Sell_tickets**, and its primary key will be composed of primary keys of both sides.

Pilot and **Employee**, as well as **Stewardess** and **Employee** are in an is-a relationship. Having completely different attributes, these entities are handled in separate table, relation between them being assured by primary key social security number (*ssn*). This way, *ssn* becomes a primary key in all of the three tables, and at the same time it is a foreign key in **Pilot** and **Stewardess**.

5 Creating the database using SQL

SQL (Structured Query Language) is a data definition and database management language, widely used for creating, using, and managing complex database systems. For the sake of this project, database server `oracle.itk.ppke.hu` of the university was used.

5.1 Preparing the database

Before jumping into creating the database, we must be sure that no table with the names in our relational model is already present. For this reason, first of all we run `DROP` statements with `CASCADE CONSTRAINTS`

option to delete everything related to tables with these names.

Drop statements can be found in [1] in executable form.

5.2 Defining tables

To create a new table in SQL, `CREATE` statement must be used. In the statement, we must specify name of the columns - that is the name of the attributes, and the type of variables we want to use for storage. In the examples, we use `VARCHAR()` arrays for strings and `INTEGER`s for numbers. In the **Flight** table, we use the DATE type as well to store departure and arrival times.

Create statements can be found in [2] in executable format.

5.3 Altering tables

Most database management systems provide a way for modifying table structure after creation. This feature is absolutely needed in case of changes in specification, but becomes inevitable in adding constraints to tables referencing each other.In SQL, this can be done by the `ALTER TABLE` command, by specifying the table to alter, followed by a valid SQL command.

Alter statements can be found in [3].

5.4 Constraints

To meet database integrity requirements, constraints must be specified. Most common example of a constraint is the so called foreign key constraint, when key of a table is stored in another table as well. As purpose of this approach is storing relationships without a separate table, when populating the database, we must be sure, that the other side of the relation really exists. Moreover, when deleting referenced entities, what happens to the referencing one must be also handled.

To specify a constraint the `CREATE CONSTRAINT` statement can be used, by choosing a name for our constraint and specifying the referencing and the referenced attributes. In this database, only foreign key constraints are applied.

Constraint creating statements can be found in [3].

6 Populating the database

Now, as database is created and ready for use, we must fill it with the data we want to store and manage.

To add new data to a database, the `INSERT` statement is used, followed by a table name, and some values to be inserted in the respective columns. If the insert violates a rule specified in a constraint, operation is refused by the management system.

Insert statement used in this project can be found in [4].

7 Queries

To test our database, the following queries were checked. SQL implemetation of the queries can be found in [5].

7.1 Simple queries

Query 1: simple selection

English: List the employees working for airline 'MALEV'!

Magyar: Listázzuk ki a MALÉV alkalmazottakat!

RA: $\sigma_{airline='\text{MALEV}'}(Employee)$

SQL:

```
SELECT ssn,
       name
FROM employee
WHERE airline = 'MALEV';
```

```
SSN         NAME
--------    -----------
001         Peter Istvan
002         Kovacs Janos
003         Kerekes Peter
004         Dombi Botond
005         Nemeth Petra
006         Tanko Karola
```

Query 2: selection, projection, multiple tables

English: Find the names of co-pilots stored in the database!

Magyar: Keressük meg az összes másodpilóta nevét!

RA: $\Pi_{Employee.name}(\sigma_{Pilot.rank='\text{co-pilot}' \text{ and } Employee.ssn=Pilot.ssn}(Employee \times Pilot))$

SQL:

```
SELECT name
FROM Pilot,
     Employee
WHERE Pilot.ssn = employee.ssn
AND rank='co-pilot';
```

```
NAME
------------
Kovacs Janos
Ian Tariceanu
```

Query 3: selection, projection, comparision, multiple table

English: List the name and the amount of flight hours of the pilots with more than 5000 flight hours.

Magyar: Listázzuk ki az 5000 óránál többet repült pilóták neveit és a repült órák számát!

RA: $\Pi_{Employee.name}(\sigma_{Pilot.flight\_hours>5000 \text{ and } Employee.ssn=Pilot.ssn}(Employee \times Pilot))$

SQL:

```
SELECT name
FROM Pilot,
     Employee
WHERE Pilot.ssn = employee.ssn
AND flight_hours > 5000;
```

```
NAME
--------------
Peter Istvan
Kerekes Peter
Livia Popescu
```

7.2 More andvanced queries

Query 4: cross product and rename

English: List the social security numbers of captain and co-pilot pairs who can fly an aircraft. Order by captain's ssn!

Magyar: Listázzuk ki azokat a kapitány-másodpilóta párokat (a kapitány személyi száma szerint) akik együtt elvezethetnek egy repülőt!

RA: $\Pi_{p.ssn,r.ssn}(\sigma_{p.rank='\text{captain}' \text{ and } r.rank='\text{co-pilot}'}(\varrho_p(Pilot) \times \varrho_r(Pilot)))$

SQL:

```
SELECT p.ssn,
       r.ssn
FROM Pilot p,
     Pilot r
WHERE p.rank = 'captain'
  AND r.rank = 'co-pilot';
```

```
SSN         SSN
----------  ----------
001         002
003         002
007         002
001         008
003         008
007         008
```

Query 5: aggregate functions and group

English: How many employees do airlines have?

Magyar: Melyik légitársaságnak hány alkalmazottja van?

RA: $\sigma_{airline,emp\_count}(airline\ \mathbf{g}\ count(name)(Employees))$

SQL:

```
SELECT airline.name,
       count(*)
FROM airline,
     employee
WHERE employee.airline = airline.name
GROUP BY airline.name;
```

```
NAME    COUNT(*)
------  ----------
MALEV   6
TAROM   5
```

Query 6: embedded select statetemt

English: Which airlines have employees speaking four languages?

Magyar: Melyik légitársaságnak van négy nyelvet beszélő alkalmazottja?

RA:
$fl \leftarrow \Pi_{ssn}(\sigma_{spoken\_language4 \neq \text{null}}(Stewardess))$
$\Pi_{airline}\sigma_{Employee.ssn=fl.ssn}(Employee \times fl)$

SQL:

```
SELECT name
FROM airline a
WHERE
    ( SELECT count(*)
     FROM employee,
          stewardess
     WHERE employee.ssn = stewardess.ssn
     AND stewardess.spoken_language4 IS NOT NULL
        AND employee.airline = a.name) > 0;
```

```
NAME
------
MALEV
```

Query 7: division

English: Which airlines have flights departing from all of the airports?

Magyar: Mely légitársaságok indítanak játarot az összes repülőtérről?

RA: $\Pi_{airline,departure\_airport}(Flight) \div \Pi_{ICAO}(Airport)$

SQL:

```
SELECT name
FROM airline a
WHERE
    (SELECT count(*)
     FROM airport) =
    (SELECT count(DISTINCT departure_airport)
     FROM flight
     WHERE flight.airline = a.name);
```

```
no rows selected
```

Query 8: ANY operator

English: To which cities does MALEV or Wizzair has flights?

Magyar: Mely városokba indít járatot a MALÉV vagy a WizzAir?

RA:
$arp \leftarrow \Pi_{arrival\_airport}(\sigma_{airline=\text{'MALEV' or } airline=\text{'WizzAir'}}(Flight))$
$res \leftarrow \Pi_{city}(\sigma_{icao=arrival\_airport}(Airport \times arp))$

SQL:

```
SELECT city
FROM airport
WHERE icao = any
    ( SELECT arrival_airport
     FROM flight
     WHERE airline = 'MALEV'
      OR airline = 'WizzAir' );
```

```
CITY
----------
Berlin
Budapest
Bukarest
```

Query 9: ALL operator - finding maximum

English: Which airlines have the maximum number of employees?

Magyar: Mely légitársaság(ok)nak van a legtöbb alkalmazottja?

RA:
$epc1 \leftarrow \sigma_{airline,emp\_count}(airline\ \mathbf{g}\ count(name)(Employees))$
$epc2 \leftarrow \sigma_{airline,emp\_count}(airline\ \mathbf{g}\ count(name)(Employees))$
$bad \leftarrow \Pi_{epc1.airline}(\sigma_{epc1.emp\_count<epc2.emp\_count}(epc1 \times epc2))$
$res \leftarrow \Pi_{name}(Airline) - bad$

SQL:

```
SELECT name,
       country
FROM airline a
WHERE
    (SELECT count(*)
     FROM employee
     WHERE employee.airline = a.name) >=
     ALL
    (SELECT count(*)
     FROM employee
     GROUP BY airline);
```

```
SSN         NAME
--------    -----------
001         Peter Istvan
002         Kovacs Janos
003         Kerekes Peter
004         Dombi Botond
005         Nemeth Petra
006         Tanko Karola
```

Query 10: UNION operator

English: List the names and addresses of MALEV pilots and TAROM stewardesses!

Magyar: Írjuk ki a MALÉV pilóták és a TAROM stewardessek neveit és lakcímét!

RA:
$mp \leftarrow \Pi_{name,address}(\sigma_{airline='\text{MALEV}' \text{ and } Employee.ssn=Pilot.ssn}(Pilot \times Employee))$
$ts \leftarrow \Pi_{name,address}(\sigma_{airline='\text{TAROM}' \text{ and } Employee.ssn=Pilot.ssn}(Employee \times Stewardess))$
$res \leftarrow mp \cup ts$

SQL:

```
SELECT name,
       address
FROM employee
WHERE employee.ssn IN
    (SELECT ssn
     FROM pilot)
  AND employee.airline = 'MALEV'
UNION
SELECT name,
       address
FROM employee
WHERE employee.ssn IN
    (SELECT ssn
     FROM stewardess)
  AND employee.airline = 'TAROM';
```

```
NAME            ADDRESS
----------      --------------
Iulia Neagra    Bucuresti Main street 10
Kerekes Peter   Budaors Fulemule u. 8
Kovacs Janos    Budapest Sziv u. 15
Peter Istvan    Budapest Margareta u. 5
Vlad Nicolae    Bucuresti Main street 11
```

8 Data manipulation

8.1 Delete

Deleting rows from a table can be done using the `DELETE` statement. As this operation cannot be easily undone, special care is required when writing the condition after the where statement. Not surprisingly, rows referenced from other tables cannot be deleted without further modification. As a default setting, the DBMS restricts erasure of such entries, however we can specify cascading behavior as well.

In our example, simulating a technical difficulty, we want to cancel all flights served by Aircraft with ID AC1. As a result, we would like all ticket reservations for these flights to be erased as well. To achieve this behavior, we add `on delete cascade` option to the foreign key constraint.

Running example of the statement above can be found in [6].

8.2 Update

In a real application, sometimes we must change the value of an attribute in one or multiple entities. For this operation, the `UPDATE` statement can be used. In the statement, we specify the table whose rows we want to manipulate, the new value for the selected attribute, and a condition about which entries we want to modify.

In our example, simulating unpleasant weather conditions, we delay every flight arriving to Budapest's Liszt Ferenc airport. Running example statement can be found in [6].

8.3 View creation

Views are logical representations of the data stored in the database, useful in case we need some data stored in different parts together. In our example, we create a view from the last query presented above. SQL code can be found in [6].

9 Normalization

References

[1] **drop.sql**
`users.itk.ppke.hu/~csuba/files/intro_dbms/project/drop.sql`

[2] **create.sql**
`users.itk.ppke.hu/~csuba/files/intro_dbms/project/create.sql`

[3] **alter.sql**
`users.itk.ppke.hu/~csuba/files/intro_dbms/project/alter.sql`

[4] **insert.sql**
`users.itk.ppke.hu/~csuba/files/intro_dbms/project/insert.sql`

[5] **query.sql**
users.itk.ppke.hu/~csuba/files/intro_dbms/project/query.sql

[6] **manip.sql**
users.itk.ppke.hu/~csuba/files/intro_dbms/project/manip.sql