Hand-in 3

Dennis Thinh Tan Nguyen

November 6, 2015

Abstract

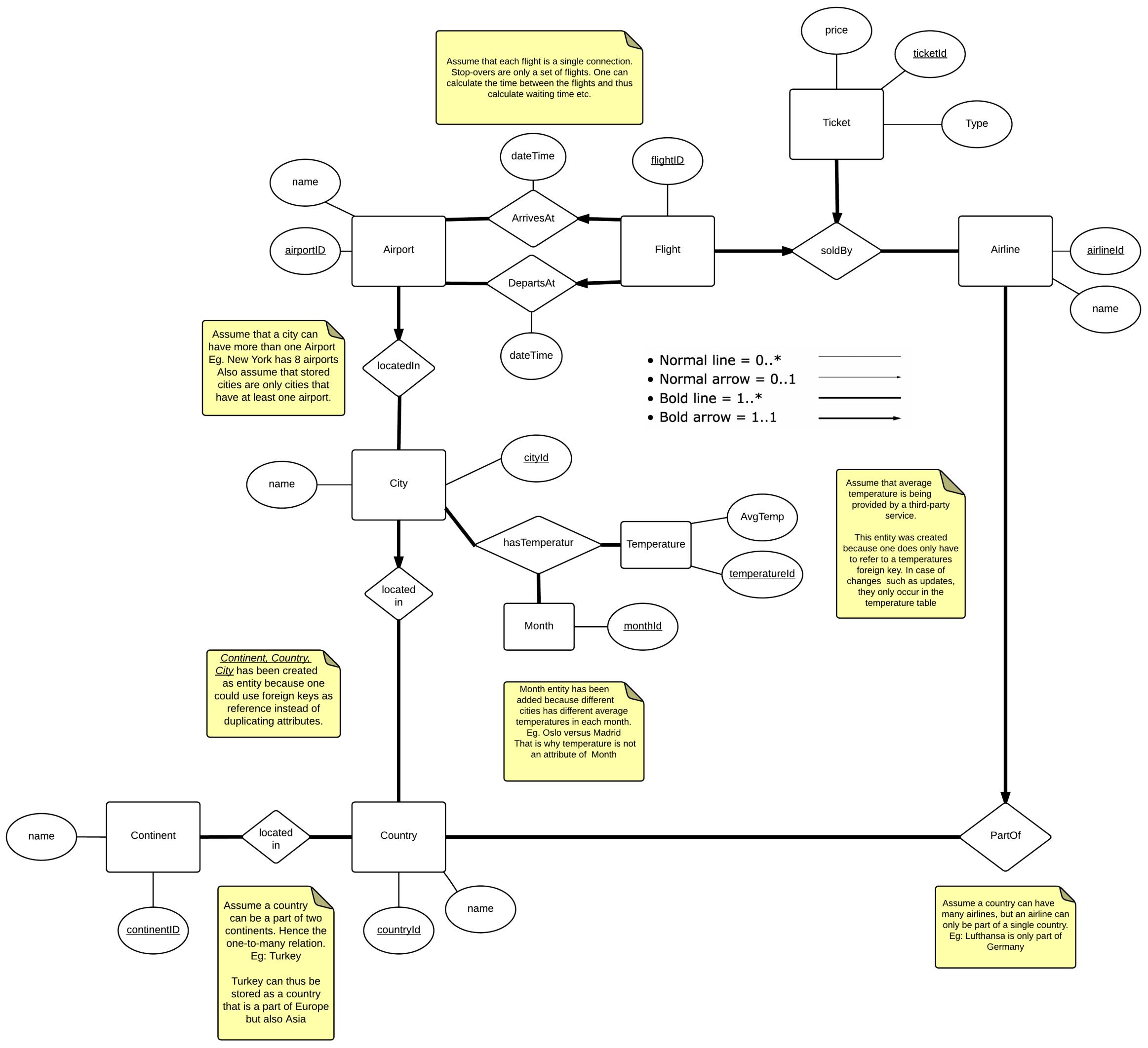
This is an individual assignment for Introduction to Database Design course, and is not a group assignment.

Contents

1 Problem 1: Data Modeling

1.1 ER-Model

Figure 1 is an E-R model for a databse that contain enough information to answer all of the given queries. Any assumptions are explicitly described in the yellow notes. For further description of the relational arrows, a list with definitions os shown on the center of the E-R model.



Assume that each flight is a single connection. Stop-overs are only a set of flights. One can calculate the time between the flights and thus calculate waiting time etc.

Assume that a city can have more than one Airport Eg. New York has 8 airports Also assume that stored cities are only cities that have at least one airport.

- Normal line = 0..*
- Normal arrow = 0..1
- Bold line = 1..*
- Bold arrow = 1..1

Assume that average temperature is being provided by a third-party service.

This entity was created because one does only have to refer to a temperatures foreign key. In case of changes such as updates, they only occur in the temperature table

Continent, Country, City has been created as entity because one could use foreign keys as reference instead of duplicating attributes.

Month entity has been added because different cities has different average temperatures in each month. Eg. Oslo versus Madrid That is why temperature is not an attribute of Month

Assume a country can be a part of two continents. Hence the one-to-many relation. Eg: Turkey

Turkey can thus be stored as a country that is a part of Europe but also Asia

Assume a country can have many airlines, but an airline can only be part of a single country. Eg: Lufthansa is only part of Germany

Figure 1: ER-Model of database

1.2 Relational data model

Below are relational Schemas derived from the ER-Model (Figur 1).

Airport(airportId: integer, cityId: integer, name: string)

Flight(flightId: integer)

Ticket(ticketId: integer, price: decimal, type: string)

Airline(airlineId: integer, countryId: integer, name: string)

soldBy(flightId: integer, ticketId: integer, airlineId)

ArrivesAt(flightId: integer, airportId: integer, arrivalTime: datetime)

DepartsAt(flightId: integer, airportId: integer, departureTime: datetime)

City(cityId: integer, countryId: integer, name: string)

Country(countryId: integer, continentId: integer, name: string)

Continent(continentId: integer, name: string)

Month(monthId: integer)

HasTemperatur(monthId: integer, cityId: integer, temperatureId: Integer)

2 Problem 2: Indexing

2.1 A: Indexing to speed up query

Since we want to perform a range search one may choose to use a b+-tree since this structure allows us to efficiently locate all data entries within a desired range. As a baseline assume that primary keys are not indexed, one may want to index all of these. Regarding the query, following primary keys will be indexed: flightId and ticketId. To speed up the query following composite keys can be created: ArrivesAt(arrivalTime,flightId), DepartsAt(depatureTime,flightId) and soldBy(flightId,ticketId). The reason why these attributes have been chosen is because they are often involved in WHERE clauses when one is defining an SQL-query about finding flights.

2.2 B: Adding 8 to given B+-tree

Figur 2 is the resulting b+-tree after adding 8 to it.

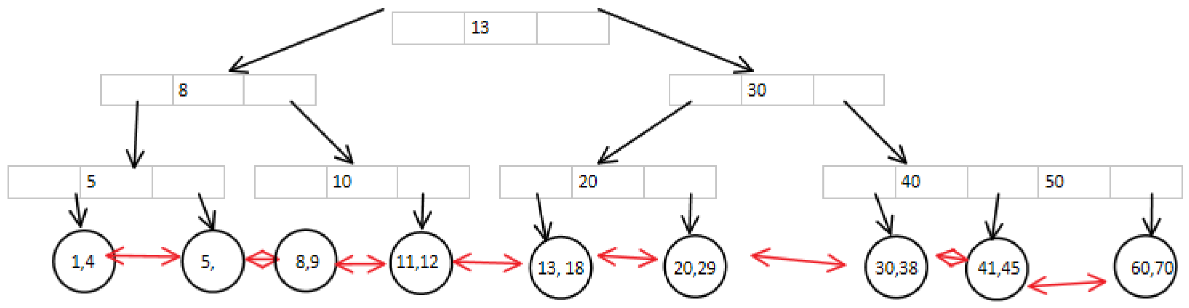


Figure 2: B+-tree after adding 8.

2.3 C: Adding 29 to to given B+-tree

Since 29 already exists, the resulting tree would be the same because duplicates are not allowed. The reason behind this is due to it being impossible to determine if 29 is greater or less than 29. Consequently, one can not insert a duplicated number in a B+-tree.

3 Problem 3: SQL queries

3.1 A: PID on all SAS airplanes built after 2000-01-01

```
SELECT pid FROM Plane WHERE buildDate >’2000-01-01’;
```

3.2 B: Number of employees who have inspected an SAS airplane

```
SELECT DISTINCT COUNT(eid)
FROM Employee as E
INNER JOIN InspectedBy as IB
ON E.eid = IB.employeeId
INNER JOIN Inspector
ON IB.inspectionId = Inspector.iid
INNER JOIN Plane
ON Inspector.planeId=pid
WHERE airline = ’SAS’;
```

3.3 C: Delete all information about inspections before 2000-01-01

```
DELETE InspectedBy.*, Inspector.*
FROM messages InspectedBy
INNER JOIN Inspector
ON Inspector.iid = InspectedBy.inspectionId
WHERE Inspector.date <’2000-01-01’;
```

3.4 D: Employees who have only worked with airplanes from one manufacturer

```
SELECT Employee.name
FROM (SELECT DISTINCT Employee.eid, Employee.name, Model.manufactor
FROM Employee as E
INNER JOIN InspectedBy as IB ON E.eid = IB.employeeId
INNER JOIN Inspector ON IB.inspectionId = Inspector.iid
INNER JOIN Plane ON Inspector.planeId=pid
INNER JOIN Model ON Plane.modelID = Model.mid) AS huge
WHERE count(huge.eid) = 1;
```

4 Problem 4: Normalization

4.1 A: Argue that modelId, buildDate is a key in Plane

By the given assumption and the definition of a key, since each manufacturer only builds one single plane per date, one may use modelId and buildDate in a query and will as result always be returned a single unique airplane. BuildDate will return a list of unique plans while modelId will filter out a specific airplane on a specific date.

4.2 B: Argue that this relation is not in BCNF.

By the definition and principles of BCNF and FD, modelId is not a key but since it will always return a unique modelname, manufacturer and enginetype. These attributes are functional dependent on modelId. Hence, this relation is not in BCNF.