Introduction to Databases - Final Project

5 February 2021

Contents

1 Conceptual Design

1.1 Requirements

Objective of the database is to store the timetable of public transportation service within a single region. Following is a description of the information that is relevant and as such is represented in the database.

A **trip** (identified by a code) consist of an ordered sequence of **stops**, each with arrival and departure times. The first stop of a trip does not have an arrival time, and the last one does not have a departure time. It **runs** on a defined period of the year, normally on specific days of the week and optionally on other dates, in which it can be exceptional or suppressed. *For example, a trip could run within 14/09/2020 (included) and 26/06/2021 (excluded) from Monday to Friday, but not if the day is a public holiday. Another could run in the same dates range on Sunday, and also if the day is a public holiday.*

A trip is part of a **line** and is conducted by a driver of a specific **company**, of which the name and the headquarter city are of interest. Of a line, the short code *(e.g. 10A)*, a brief description *(e.g. Hospital-City Centre-Hospital)*, the color, the city in which it originates and the type of vehicle *(e.g. bus or train)* are of interest. The short code is unique only within a single city.

A stop is instead characterized by the name, the **city** and the coordinates. When considering bus or train stations, the stopping point should also contain the platform name. The stop name should be memorized in different languages *(e.g. in Italian and German)*. It is also the case that at least one translation is present, and if there is more than one translation, then all stop names are translated in all the possible languages.

1.2 ER Schema

See Figure 1 on next page.

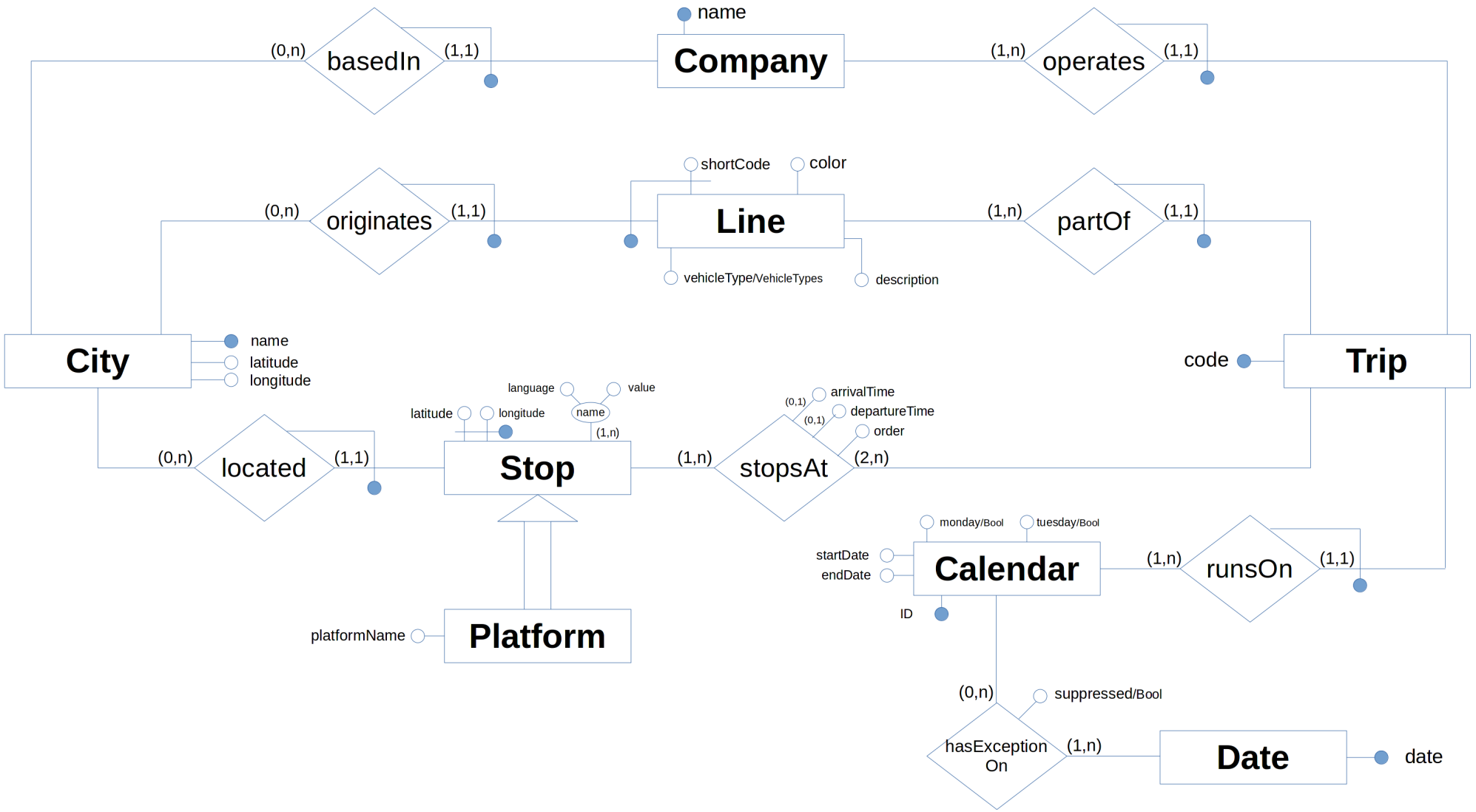

Figure 1: ER Schema

Note: not all attributes have been indicated in the schema. Please check the data dictionary for a complete overview

1.3 Glossary

Term	Description
Stop	Physical location in which passengers get on or off a vehicle
Station	Large stop consisting of many platform, where more than one vehicle can board passengers at the same time
Platform	Single stopping point inside a Station. It is represented by a number
Line	Group of trips that follow the same route and is identified to the public with the same number or code
Trip	Ordered sequence of two or more stops, each with arrival and departure times
Calendar	Defines when a trip runs in terms of date ranges and days of the week
Company	Firm that is responsible of operating a trip

Table 1: Glossary

1.4 Data Dictionary

1.4.1 External Constraints

For completeness, all possible constraints have ben enlisted. Note that some of them can not be enforced at database level, either due to the lack functionalities of the DBMS or for the high complexity that they have. In particular, only constraints 1-4 will be checked by the DBMS.

1. For each tuple in the entity `Calendar`, the attribute *endDate* must encode a date strictly greater than *startDate*.
2. For each tuple in the relationship `stopsAt`, the attribute *departureTime* must encode a time greater or equal than *arrivalTime*.
3. In the relationship `stopsAt`, for the first stop of each `trip` (with *order=1*), the attribute *arrivalTime* must be omitted and *departureTime* must be specified. For the last stop (with maximal *order* for the trip), the attribute *arrivalTime* must be specified and *departureTime* must be omitted. For all the other stops, both attributes must be present.
4. In the relationship `stopsAt`, the attribute *order* must be equal to `1` for the first stop of a trip, and should be incremented by one for subsequent stops.
5. In the relationship `stopsAt`, for every pair of tuples *T1* and *T2* within the same trip, if *T1[order]* < *T2[order]*, then *T1[departureTime]* < *T2[arrivalTime]*
6. For a `Calendar`, the exceptional dates defined through the `hasExceptionOn` relationship must be in between *startDate* and *endDate*. Moreover, if *suppressed=True* then the corresponding day of week has value *True* (that is, trips will be exceptionally suppressed on that day, since normally on that day of week they do run). On the opposite, if *suppressed=False* then the corresponding day of week has value *False* (that is, trips will exceptionally run on that day, since normally they do not run on that day of week).

1.4.2 Entities

See tables 2 and 3.

Entity	Description	Attributes	Identifiers
Calendar	List of dates for which a trip runs. It defines a date range, with start and end date, together with the days of week in which the trip normally runs. Each day of week is a boolean value, so that if the value is `true` the trip runs on that day, otherwise it does not.	ID, startDate, endDate, monday, tuesday, wednesday, thursday, friday, saturday, sunday	{ID}
City	Represents a city within the region of interest.	name, latitude, longitude	{name}
Company	Transportation company responsible for running a set of trips	name	{name}
Date	Specific date for which there is an exception (the corresponding trips are either suppressed or run exceptionally)	date	{date}
Line	A public transportation line, ran by a specific kind of vehicle	shortCode, description, color, vehicleType	{shortCode, city}
Stop	Point in which a vehicle stops to allow passengers to board or get off	name, latitude, longitude	{latitude, longitude}
Platform	Special kind of stop, represented by a code	platfornName	{latitude, longitude}
Trip	Ordered sequence of two or more stops, each with arrival and departure times	code	{code}

Table 2: Entities

1.4.3 Relationships

Relationship	Description	Components	Attributes	Identifiers
basedIn	Headquarter city of a company	City, Company	-	{company}
hasExceptionOn	Exception for a date in the corresponding calendar	Date, Calendar	suppressed *(true=suppressed, false=exceptional)*	{date, calendar}
located	City where a stop is located	Stop, City	-	{stop}
operates	Which company operates a specific trip	Company, Trip	-	{trip}
originates	City from where a line departs	City, Line	-	{line}
partOf	A trip is part of a line	Trip, Line	-	{trip}
runsOn	Days of week in which the trip normally runs	Trip, Calendar	-	{trip}
stopsAt	Stops traversed by a Trip, with time and order	Trip, Stop	arrivalTime, departureTime, order	{trip, stop}

Table 3: Relationships

1.5 Application Load

The indications of volume and frequency of operations are a rough estimation of a possible database instance in a small-sized region. The following assumptions were taken:

1. On average, each calendar has 15 exceptional days;
2. On average, each trip has nine stops;
3. On average, 15 trips halt at any stop in the database every day;
4. The stop names are fully translated in two languages. In general, it is assumed that there can be only *full* translations (i.e. all tuples are translated in all defined languages);
5. The timetable changes two times per year.

1.5.1 Table of Volumes

See table 4.

Concept	Construct	Volume
City	Entity	30
Stop	Entity	75
Platform	Entity	20
Trip	Entity	500
Company	Entity	5
Line	Entity	25
Calendar	Entity	10
Date	Entity	200
basedIn	Relationship	5
operates	Relationship	500
originates	Relationship	25
partOf	Relationship	500
located	Relationship	75
stopsAt	Relationship	4500
runsOn	Relationship	500
hasExceptionOn	Relationship	150

Table 4: Table of Volumes

1.5.2 List of Operations

1. Get a list of *(maximum)* ten stops with name in a specific language matching the user-given string;
2. Get a list of *(maximum)* ten stops with name in a specific language that are contained in a set of coordinates;
3. Find the next ten trips, with line and company information, that depart from a specific stop at a given date and time *(assuming that the stop coordinates are known)*;
4. Find the next ten trips departing from one stop and arriving at another one, at a given date and time *(assuming that the stop coordinates are known)*;
5. Add a new company given the name and the headquarter city, *and considering that all possible cities are already loaded into the database*;
6. Add a new line given short code, color, type of vehicle and city, *and considering that all possible cities are already loaded into the database*;
7. Add a new trip given the code, the timetable, the operating company and the calendar;
8. Modify the schedule for a calendar knowing its code (add one exceptional date)

Op.	Type	Frequency
1	interactive	500 / day
2	interactive	200 / day
3	interactive	500 / day
4	interactive	750 / day
5	batch	5 / year
6	batch	25 / year
7	batch	500 / year
8	batch	500 / year

Table 5: Frequency of operations

2 Restructuring of the Conceptual Schema

2.1 Redundancy Analysis

No redundancies were found in the original schema of figure 1.

2.2 Restructured ER Schema

See Figure 2 on next page.

2.3 Data Dictionary

2.3.1 New entities and relationships

One new entity and two new relationships are added to the original data dictionary. They originate from the multivalued and composite attribute `Stop[name]` and from the elimination of ISA between `Stop` and `Platform`.

For the stop name, a modular approach was followed. In particular, the fact that the number of translations is not fixed, but can change over time was considered. This has lead to the following table, where each translation is associated with the corresponding language.

Entity	Description	Attributes	Identifiers
StopName	Translation of a stop name	language, value	{language, value}

Table 6: Newly-added entities

Relationship	Description	Components	Attributes	Identifiers
stopHasName	Connects the translations of the stop name of a line to the stop itself	Stop, StopName	-	{latitude, longitude, language, value}
ISA-S-P	Relationship connecting a platform to the corresponding stop tuple	Stop, Platform	-	{latitude, longitude}

Table 7: Newly-added relationships

2.3.2 New External Constraints

No new external constraints were added.

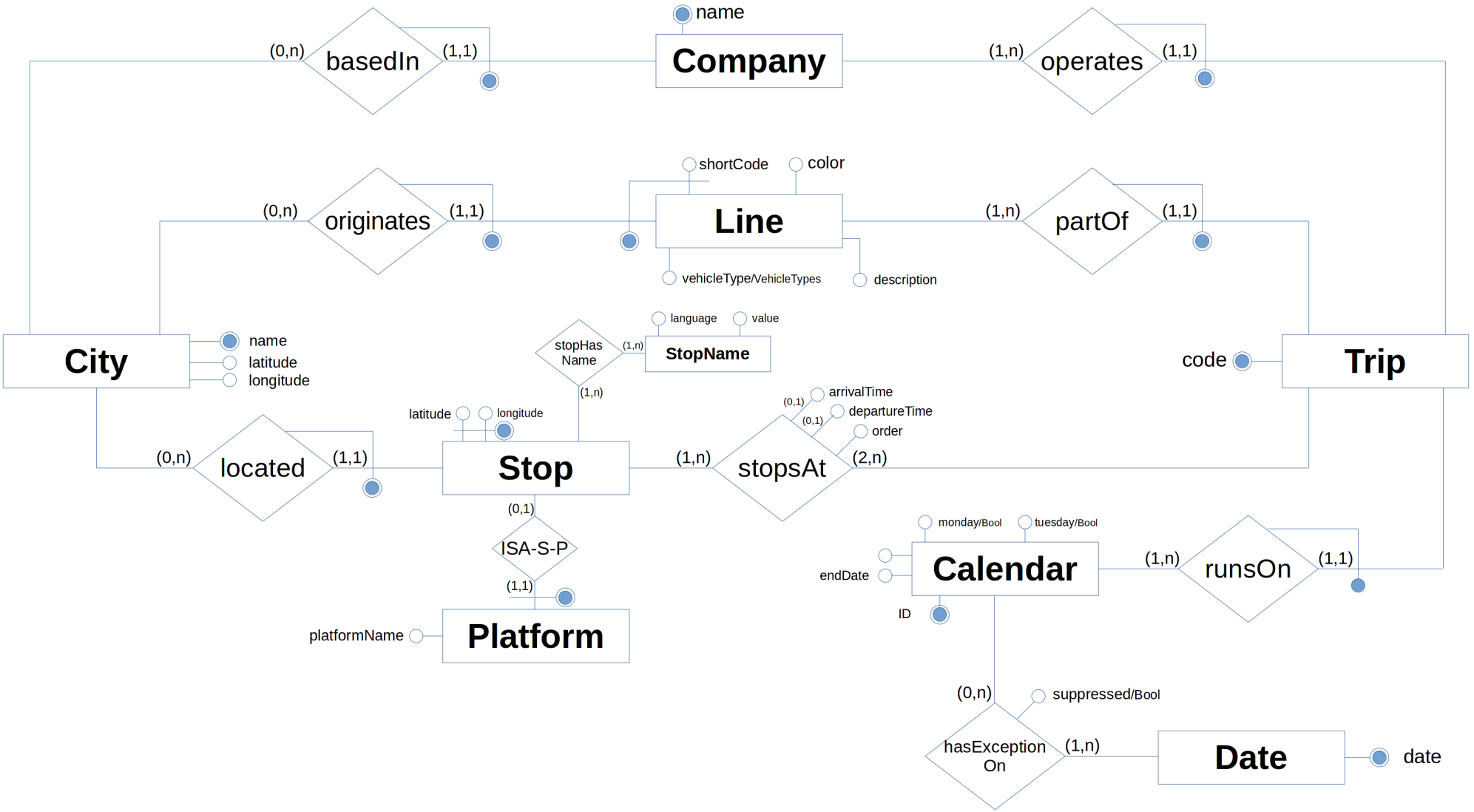

Figure 2: Restructured ER Schema

Note: not all attributes have been indicated in the schema. Please check the data dictionary for a complete overview

2.4 Reformulated Application Load

The following data is based on the original table of volumes and on the assumptions, particularly on point 4 about the translations. Regarding access tables, costs were written by considering the worst possible scenario (i.e. the one with maximal computational expense).

2.4.1 Table of Volumes

Concept	Construct	Volume
City	Entity	30
Stop	Entity	75
StopName	Entity	150
Platform	Entity	20
Trip	Entity	500
Company	Entity	5
Line	Entity	25
Calendar	Entity	10
Date	Entity	200
basedIn	Relationship	5
operates	Relationship	500
originates	Relationship	25
partOf	Relationship	500
located	Relationship	100
stopsAt	Relationship	4500
runsOn	Relationship	500
hasExceptionOn	Relationship	150
stopHasName	Relationship	150
ISA-S-P	Relationship	30

Table 8: Restructured Table of Volumes

2.4.2 Access Tables

(1) Get a list of *(maximum)* ten stops with name in a specific language matching the user-given string

Concept	Construct	Accesses	Type
StopName	Entity	75	Read
stopHasName	Relationship	10	Read
Stop	Entity	10	Read
ISA-S-P	Relationship	10	Read
Platform	Entity	10	Read
located	Relationship	10	Read
City	Entity	10	Read

Table 9: Access table for operation 1

(2) Get a list of *(maximum)* ten stops with name in a specific language that are contained in a set of coordinates

Concept	Construct	Accesses	Type
Stop	Entity	75	Read
stopHasName	Relationship	10	Read
StopName	Entity	10	Read
located	Relationship	10	Read
City	Entity	10	Read
ISA-S-P	Relationship	10	Read
Platform	Entity	10	Read

Table 10: Access table for operation 2

(3) Find the next ten trips, with line and company information, that depart from a specific stop at a given date and time *(assuming that the stop coordinates are known)*

Concept	Construct	Accesses	Type
stopsAt	Relationship	15	Read
Trip	Entity	15	Read
runsOn	Relationship	15	Read
Calendar	Entity	15	Read
hasExceptionOn	Relationship	225	Read
Date	Relationship	225	Read
partOf	Relationship	10	Read
Line	Entity	10	Read
operates	Relationship	10	Read
Company	Entity	10	Read

Table 11: Access table for operation 3

(4) Find the next ten trips departing from one stop and arriving at another one, at a given date and time *(assuming that the stop coordinates are known)*

Concept	Construct	Accesses	Type
stopsAt	Relationship	30	Read
Trip	Entity	30	Read
runsOn	Relationship	30	Read
Calendar	Entity	30	Read
hasExceptionOn	Relationship	225	Read
Date	Relationship	225	Read
partOf	Relationship	10	Read
Line	Entity	10	Read
operates	Relationship	10	Read
Company	Entity	10	Read

Table 12: Access table for operation 4

(5) Add a new company given the name and the headquarter city, *and considering that all possible cities are already loaded into the database*

Concept	Construct	Accesses	Type
City	Entity	1	Read
Company	Entity	1	Write
basedIn	Relationship	1	Write

Table 13: Access table for operation 5

(6) Add a new line given short code, color, type of vehicle and city, *and considering that all possible cities are already loaded into the database*

Concept	Construct	Accesses	Type
City	Entity	1	Read
Line	Entity	1	Write
originates	Relationship	1	Write

Table 14: Access table for operation 6

(7) Add a new trip given the code, the timetable, the operating company and the calendar

Concept	Construct	Accesses	Type
Trip	Entity	1	Write
stopsAt	Relationship	9	Write
partOf	Relationship	1	Write
operates	Relationship	1	Write
runsOn	Relationship	1	Read

Table 15: Access table for operation 7

(8) Modify the schedule for a calendar knowing its code (add one exceptional date)

Concept	Construct	Accesses	Type
hasException	Relationship	1	Write
Date	Entity	1	Write

Table 16: Access table for operation 8

3 Direct Translation

3.1 Relational schema

City(<u>name</u>, latitude, longitude)

Company(<u>name</u>)
Inclusion: Company[name] ⊆ Operates[company]
Foreign Key: Company[name] ⊆ BasedIn[company]

Line(<u>shortCode, cityName</u>, color, vehicleType, description)
Foreign Key: Line[cityName] ⊆ City[name]
Inclusion: Line[shortCode, cityName] ⊆ PartOf[lineCode, lineCity]

Trip(<u>code</u>)
Inclusion: Trip[code] ⊆ StopsAt[trip]
Foreign Key: Trip[code] ⊆ PartOf[trip]
Foreign Key: Trip[code] ⊆ Operates[trip]
Foreign Key: Trip[code] ⊆ RunsOn[trip]

Calendar(<u>ID</u>, startDate, endDate, monday, tuesday, wednesday, thurdsay, friday, saturday, sunday)

Inclusion: Calendar[ID] ⊆ RunsOn[calendar]

Date(<u>date</u>)
Inclusion: Date[date] ⊆ HasExceptionOn[date]

Stop(<u>latitude, longitude</u>)
Inclusion: Stop[latitude, longitude] ⊆ StopsAt[latitude, longitude]
Inclusion: Stop[latitude, longitude] ⊆ StopHasName[latitude, longitude]
Foreign key: Stop[latitude, longitude] ⊆ Located[stopLat, stopLon]

StopName(<u>language, value</u>)
Inclusion: StopName[language, value] ⊆ StopHasName[language, value]

Platform(<u>latitude, longitude</u>, platformName)
Foreign Key: Platform[latitude, longitude] ⊆ Stop[latitude, longitude]

StopsAt(<u>latitude, longitude, trip</u>, order, arrivalTime*, departureTime*)
Foreign Key: StopsAt[latitude, longitude] ⊆ Stop[latitude, longitude]
Foreign Key: StopsAt[trip] ⊆ Trip[code]

StopHasName(<u>latitude, longitude, language, value</u>)
Foreign Key: StopHasName[latitude, longitude] ⊆ Stop[latitude, longitude]
Foreign Key: StopHasName[language, value] ⊆ StopName[language, value]

Located(<u>stopLat, stopLon</u>, cityName)
Foreign Key: Located[stopLat, stopLon] ⊆ Stop[latitude, longitude]
Foreign Key: Located[cityName] ⊆ City[name]

RunsOn(trip, calendar)
Foreign Key: RunsOn[trip] ⊆ Trip[code]
Inclusion: RunsOn[calendar] ⊆ Calendar[ID]

HasExceptionOn(date, calendar, suppressed)
Foreign Key: HasExceptionOn[date] ⊆ Date[date]
Foreign Key: HasExceptionOn[calendar] ⊆ Calendar[ID]

BasedIn(company, city)
Foreign Key: BasedIn[city] ⊆ City[name]
Foreign Key: BasedIn[company] ⊆ Company[name]

Operates(trip, company)
Foreign Key: Operates[trip] ⊆ Trip[code]
Foreign Key: Operates[company] ⊆ Company[name]

PartOf(trip, lineCode, lineCity)
Foreign Key: PartOf[trip] ⊆ Trip[code]
Foreign Key: PartOf[lineCode, lineCity] ⊆ Line[shortCode, city]

3.2 External Constraints

Constraint 7) was added. It is originated from the fact that a trip must have at least two stops, which was previously indicated in the conceptual schema with minimum cardinality equal to 2.

1. For each tuple in the entity `Calendar`, the attribute *endDate* must encode a date strictly greater than *startDate*.

2. For each tuple in the relationship `stopsAt`, the attribute *departureTime* must encode a time greater or equal than *arrivalTime*.

3. In the relationship `stopsAt`, for the first stop of a trip (with *order=1*), the attribute *arrivalTime* must be omitted and *departureTime* must be specified. For the last stop (with maximal *order*), the attribute *arrivalTime* must be specified and *departureTime* must be omitted. For all the other stops, both attributes must be present.

4. In the relationship `stopsAt`, the attribute *order* must be equal to `1` for the first stop of a trip, and should be incremented by one for subsequent stops.

5. In the relationship `stopsAt`, for every pair of tuples *T1* and *T2* within the same trip, if *T1[order] < T2[order]*, then *T1[departureTime] < T2[arrivalTime]*

6. For a `Trip`, the exceptional dates defined through the `hasExceptionOn` relationship must be in between *startDate* and *endDate* defined by the corresponding `Calendar` instance. Moreover, if *suppressed=True* then the corresponding day of week in the `Calendar` instance has value *True*. At the opposite, if *suppressed=False* then the corresponding day of week in `Calendar` has value *False*;

7. In the relation `StopsAt`, the minimum number of tuples having the same value for the *trip* attribute is two.

3.3 Application Load

(1) Get a list of *(maximum)* ten stops with name in a specific language matching the user-given string

Concept	Accesses	Type
StopName	75	Read
StopHasName	10	Read
Stop	10	Read
ISA-S-P	10	Read
Platform	10	Read
Located	10	Read
City	10	Read

Table 17: Access table for operation 1

(2) Get a list of *(maximum)* ten stops with name in a specific language that are contained in a set of coordinates

Concept	Accesses	Type
Stop	75	Read
StopHasName	10	Read
StopName	10	Read
Located	10	Read
City	10	Read
ISA-S-P	10	Read
Platform	10	Read

Table 18: Access table for operation 2

(3) Find the next ten trips, with line and company information, that depart from a specific stop at a given date and time *(assuming that the stop coordinates are known)*

Concept	Accesses	Type
StopsAt	15	Read
Trip	15	Read
RunsOn	15	Read
Calendar	15	Read
HasExceptionOn	225	Read
Date	225	Read
PartOf	10	Read
Line	10	Read
Operates	10	Read
Company	10	Read

Table 19: Access table for operation 3

(4) Find the next ten trips departing from one stop and arriving at another one, at a given date and time *(assuming that the stop coordinates are known)*

Concept	Accesses	Type
StopsAt	30	Read
Trip	30	Read
RunsOn	30	Read
Calendar	30	Read
HasExceptionOn	225	Read
Date	225	Read
PartOf	10	Read
Line	10	Read
Operates	10	Read
Company	10	Read

Table 20: Access table for operation 4

(5) Add a new company given the name and the headquarter city, *and considering that all possible cities are already loaded into the database*

Concept	Accesses	Type
City	1	Read
Company	1	Write
basedIn	1	Write

Table 21: Access table for operation 5

(6) Add a new line given short code, color, type of vehicle and city, *and considering that all possible cities are already loaded into the database*

Concept	Accesses	Type
City	1	Read
Line	1	Write
originates	1	Write

Table 22: Access table for operation 6

(7) Add a new trip given the code, the timetable, the operating company and the calendar

Concept	Accesses	Type
Trip	1	Write
stopsAt	9	Write
partOf	1	Write
Operates	1	Write
RunsOn	1	Read

Table 23: Access table for operation 7

(8) Modify the schedule for a calendar knowing its code (add one exceptional date)

Concept	Accesses	Type
HasException	1	Write
Date	1	Write

Table 24: Access table for operation 8

4 Restructuring of the relational schema

The obtained relational schema can be further optimized.

Firstly, some redundant relations are found: `StopName` and `HasName`, and also `Date` and `HasExceptionOn`. In both cases, one may see that the first relations consist of just one attribute, and that the same attribute is already included in the second relations as a foreign key. It is therefore reasonable to delete `StopName` and `Date`, because the relevant data can be queried by using the `HasName` and `HasExceptionOn`.

In general, it was noticed that the complexity of queries for interactive operations (i.e. 1-4) was quite high and it involved a considerable number of joins between relations. For this reason, the decision taken was to delete those relations deriving from relationships in the original schema, where one role had cardinality *(1,1)*, for example *RunsOn, PartOf, ...* These relations have been merged into the role with cardinality *(1,1)*; the inclusion, foreign key and external constraints have been modified accordingly.

Another context in which a relation may be deleted is the one of `Stop` and `Platform`. The latter relation contains the name of the platform, plus a reference to the Stop relation. To reduce the number of joins needed to perform the various operations, it has been chosen to merge those relations together. The `Stop` relation will then gain a new *nullable* attribute, `platformName`. Even though this increases the number of pages in the relation, it is also reducing the queries complexity, in particular the one of operations 3 and 4. Since those operations are interactive, more importance has been given to execution time rather than space efficiency.

4.1 Restructured schema

City(<u>name</u>, latitude, longitude)

Company(<u>name</u>, headquarterCity)
Foreign key: Company[headquarterCity] ⊆ City[name]

Line(<u>shortCode, departingCity</u>, color, vehicleType, description)
Foreign Key: Line[departingCity] ⊆ City[name]
Inclusion: Line[shortCode, departingCity] ⊆ Trip[lineCode, lineCity]

Trip(<u>code</u>, calendar, company, lineCode, lineCity)
Inclusion: Trip[code] ⊆ StopsAt[trip]
Foreign Key: Trip[calendar] ⊆ Calendar[ID]
Foreign Key: Trip[company] ⊆ Company[name]
Foreign Key: Trip[lineCode,lineCity] ⊆ Line[shortCode,departingCity]

Calendar(<u>ID</u>, startDate, endDate, monday, tuesday, wednesday, thurdsay, friday, saturday, sunday)

Inclusion: Calendar[ID] ⊆ Trip[calendar]

Stop(<u>latitude, longitude</u>, city, platformName*)
Inclusion: Stop[latitude, longitude] ⊆ StopsAt[latitude, longitude]
Inclusion: Stop[latitude, longitude] ⊆ StopHasName[latitude, longitude]
Foreign Key: Stop[city] ⊆ City[name]

StopsAt(<u>latitude, longitude, trip</u>, order, arrivalTime*, departureTime*)
Foreign Key: StopsAt[latitude, longitude] ⊆ Stop[latitude, longitude]
Foreign Key: StopsAt[trip] ⊆ Trip[code]

StopHasName(<u>latitude, longitude, language</u> value)
Foreign Key: StopHasName[latitude, longitude] ⊆ Stop[latitude, longitude]

HasExceptionOn(<u>date, calendar</u>, suppressed)
Foreign Key: HasExceptionOn[calendar] ⊆ Calendar[ID]

4.2 External Constraints

1. For each tuple in the entity `Calendar`, the attribute *endDate* must encode a date strictly greater than *startDate*.

2. For each tuple in the relationship `stopsAt`, the attribute *departureTime* must encode a time greater or equal than *arrivalTime*.

3. In the relationship `stopsAt`, for the first stop of a trip (with *order=1*), the attribute *arrivalTime* must be omitted and *departureTime* must be specified. For the last stop (with maximal *order*), the attribute *arrivalTime* must be specified and *departureTime* must be omitted. For all the other stops, both attributes must be present.

4. In the relationship `stopsAt`, the attribute *order* must be equal to `1` for the first stop of a trip, and should be incremented by one for subsequent stops.

5. In the relationship `stopsAt`, for every pair of tuples *T1* and *T2* within the same trip, if *T1[order]* < *T2[order]*, then *T1[departureTime]* < *T2[arrivalTime]*

6. For a `Trip`, the exceptional dates defined through the `hasExceptionOn` relationship must be in between *startDate* and *endDate* defined by the corresponding `Calendar` instance. Moreover, if *suppressed=True* then the corresponding day of week in the `Calendar` instance has value *True*. At the opposite, if *suppressed=False* then the corresponding day of week in `Calendar` has value *False*;

7. In the relation `StopsAt`, the minimum number of tuples having the same value for the *trip* attribute is two.

4.3 Application Load

(1) Get a list of *(maximum)* ten stops with name in a specific language matching the user-given string

Concept	Accesses	Type
StopHasName	75	Read
Stop	10	Read

Table 25: Access table for operation 1

(2) Get a list of *(maximum)* ten stops with name in a specific language that are contained in a set of coordinates

Concept	Accesses	Type
Stop	75	Read
StopHasName	10	Read

Table 26: Access table for operation 2

(3) Find the next ten trips, with line and company information, that depart from a specific stop at a given date and time *(assuming that the stop coordinates are known)*

Concept	Accesses	Type
StopsAt	15	Read
Trip	15	Read
Calendar	15	Read
HasExceptionOn	225	Read
Line	10	Read

Table 27: Access table for operation 3

(4) Find the next ten trips departing from one stop and arriving at another one, at a given date and time *(assuming that the stop coordinates are known)*

Concept	Accesses	Type
StopsAt	30	Read
Trip	30	Read
Calendar	30	Read
HasExceptionOn	225	Read
Line	10	Read

Table 28: Access table for operation 4

(5) Add a new company given the name and the headquarter city, *and considering that all possible cities are already loaded into the database*

Concept	Accesses	Type
City	1	Read
Company	1	Write

Table 29: Access table for operation 5

(6) Add a new line given short code, color, type of vehicle and city, *and considering that all possible cities are already loaded into the database*

Concept	Accesses	Type
City	1	Read
Line	1	Write

Table 30: Access table for operation 6

(7) Add a new trip given the code, the timetable, the operating company and the calendar

Concept	Accesses	Type
Trip	1	Write
stopsAt	9	Write

Table 31: Access table for operation 7

(8) Modify the schedule for a calendar knowing its code (add one exceptional date)

Concept	Accesses	Type
HasException	1	Write

Table 32: Access table for operation 8

5 SQL Specification

Three attatched files are present in the `sql` directory:

- `base.sql` contains the definition of the various relations
- `data.sql` contains some sample data
- `full.sql` is the dump of the complete database, left for fallback

They may be imported into a local instance with the following commands:

```
psql
postgres=# \i base.sql
publictransport_17573=# \i data.sql
```

As it may be seen, tuple and external constraints have been implemented either as CHECK operations or as triggers with stored procedures. Note that external constraints 5-6 were considered too difficult and as such they have not been implemented. Constraint 4 was implemented by relying on postgres' `Serial` data type, which represents an increasing integer that start from 1. The domain `VehicleType` for a *Line* has been created as a custom type.

5.1 Queries related to the operations

(1) Get a list of *(maximum)* ten stops with name in a specific language matching the user-given string. *Parameters: stop name, language*

```
SELECT S.city, SN.value AS name, S.latitude, S.longitude
FROM Stop S, StopHasName SN
WHERE S.latitude=SN.latitude AND S.longitude=SN.longitude
AND SN.value ILIKE _name_ AND SN.language=_language_;
```

(2) Get a list of *(maximum)* ten stops with name in a specific language that are contained in a set of coordinates. *Parameters: latitude, longitude, radius, language*

```
SELECT S.city, SN.value AS name, S.latitude, S.longitude
FROM Stop S, StopHasName SN
WHERE SN.latitude=S.latitude AND SN.longitude=S.longitude
AND acos( cos(radians(_latitude_)) * cos(radians(S.latitude)) *
cos(radians(S.longitude) - radians(_longitude_)) + sin(radians(_latitude_)) *
sin(radians(S.latitude)) ) * 6371393 <= _radius_
AND S.platformName IS NULL AND SN.language=_language_;
```

(3) Find the next ten trips, with line and company information, that depart from a specific stop at a given date and time. *Parameters: stop latitude, stop longitude, date and time*

```
SELECT T.company, L.shortCode AS Line, L.vehicleType, T.code as "Trip Code",
  SA.departureTime AS "Departure Time"
FROM trip T, calendar C, stopsAt SA, line L
WHERE SA.latitude=_stopLat_ AND SA.longitude=_stopLon
AND SA.departureTime>=_time_ AND T.calendar=C.ID
AND T.code=SA.trip AND T.lineCode=L.shortCode
AND (
((C._dayOfWeek_=True) AND (_date_ NOT IN (SELECT date FROM HasExceptionOn E
        WHERE suppressed=True AND E.calendar=C.ID)))
OR (_date_ IN (SELECT date FROM HasExceptionOn E
        WHERE suppressed=False AND E.calendar=C.ID))
)
ORDER BY SA.arrivalTime
LIMIT 10;
```

(4) Find the next ten trips departing from one stop and arriving at another one, at a given date and time. *Parameters: latitude and longitude for the two stops, date and time*

```
SELECT T.company, L.shortCode AS Line, L.vehicleType, T.code as "Trip Code",
  S1.departureTime AS "Departure Time", S2.arrivalTime AS "Arrival Time",
  (S2.arrivalTime-S1.departureTime) AS duration
FROM trip T, calendar C, stopsAt S1, stopsAt S2, line L
WHERE S1.latitude=_s1lat_ AND S1.longitude=_s1lat_ AND S2.latitude=_s2lat_
AND S2.longitude=_s2lon_ AND S1.departureTime>=_time_
AND S1.trip=S2.trip AND T.calendar=C.ID AND T.code=S1.trip
AND T.lineCode=L.shortCode
AND (
((C._dayOfWeek_=True) AND (_date_ NOT IN (SELECT date FROM HasExceptionOn E
        WHERE suppressed=True AND E.calendar=C.ID)))
OR (_date_ IN (SELECT date FROM HasExceptionOn E
        WHERE suppressed=False AND E.calendar=C.ID))
)
ORDER BY S1.arrivalTime
LIMIT 10;
```

(5) Add a new company given the name and the headquarter city. *Parameters: company name, company city*

```
INSERT INTO Company(name, headquarterCity) VALUES(...);
```

(6) Add a new line given short code, color, type of vehicle and city. *Parameters: line short code, departing city, color, vehicle type*

```
INSERT INTO Line(shortCode, color, vehicleType, cityName) VALUES (...);
```

(7) Add a new trip given the code, the timetable, the operating company and the calendar. *Parameters: trip code, calendar ID, line code and departing city, company name. List of stops with the timetable (arrival and departure time)*

```
INSERT INTO Trip(code,calendar,lineCode,lineCity,company) VALUES (...);
INSERT INTO StopsAt(latitude,longitude,arrivalTime,departureTime) VALUES (...);
```

(8) Modify the schedule for a calendar knowing its code (add one exceptional date). *Parameters: date, calendar ID, suppression state*

```
INSERT INTO HasExceptionOn(date, calendar, suppressed) VALUES (...);
```

6 Java Applcation

The Java code is available in the `app/code.zip` file. A ready-to-be-run file is found in the `app` directory. It may be executed in a terminal with:

```
java -jar app/publictransport_17573.jar
```

Note that Java 15 is required. It may be needed to specify the path to postgresql driver with `-classpath` argument. The connection parameters may be altered by modifying the `database.properties` file.

The application allows to execute the queries related to the table of operations. After the initialization, the user is presented with a menu, from which all operations can be run:

```
****** Introduction to Databases project ******

Connecting... successfully connected to the database


------------ MENU ------------
1) Stops by name
2) Stops by coordinates
3) Trips per Stop
4) Trips between two stops
5) Add new company
6) Add new line
7) Add new trip
8) Modify calendar schedule
9) Exit
------------------------------
Please choose an item:
```