

University of Trieste
Artificial Intelligence and Data Analytics

Database Project

Milan Public Transport Network

Margherita Necchi
Student ID SM3201404

June 6, 2024

Contents

1	Project overview	2
1.1	Introduction	2
1.2	Purpose and requirements	2
1.3	Glossary	3
1.4	Main operations	3
1.5	Grouping of requirements into homogeneous sentences	4
2	Conceptual design	5
2.1	E-R schema	5
2.2	Data dictionary	7
2.2.1	Entities	7
2.2.2	Relationships	8
2.3	Constraints not expressible in the schema	8
2.4	Volume table	9
3	Logical design	9
3.1	Preliminary considerations	9
3.2	Redundancy analysis	9
3.2.1	Derived attribute: “number of stops”	10
3.2.2	Cycle between route, priority lane and stop	10
3.3	Removal of generalisations	11
3.4	Multi-valued attributes	11
3.5	Choice of identifiers	11
3.6	Restructured E-R schema	12
3.7	Translation into the relational model	12
3.8	Logical schema	13
3.9	Normalisation	13
4	SQL implementation	14
4.1	Table creation	14
4.2	Operations	14
4.3	Triggers	16
4.4	Python interaction	17

1 Project overview

1.1 Introduction

The goal is to design a relational database that supports Azienda Trasporti Milanesi (hereafter ATM), Milan's public transport company, in a project aimed at optimising the city's public transport network on several fronts. The main purpose of the database is to provide fast and well-organised access to data on public transport traffic, so that the information needed for the optimisation can be derived easily.

Note on naming. The diagrams (E-R schemas and logical schema) and the SQL implementation use Italian names. In the text, the corresponding Italian name is given in parentheses the first time a concept appears, e.g. route (*tratta*).

1.2 Purpose and requirements

The optimisation process will be based on a “quality measure” estimated from four types of information: the polluting impact of a route, its average duration, its costs and the coverage of the city it provides.

Means of transport. The network uses three different means of transport: underground (metro), buses and trams. For each vehicle the database stores its capacity, whether drivers and ticket inspectors are needed, an estimate of the pollution it produces and its licence plate (unique identification code).

As a generalisation, the underground is considered driverless (2 lines out of 5 already are). Ticket inspectors are considered unnecessary, since they are only present at station entrances (for a cost analysis, an estimate of 4 inspectors per stop can be used).

A tram requires one driver and three ticket inspectors per run.

Buses are divided into two categories according to their engine: internal combustion (diesel) vehicles and electric buses. Like trams, they require one driver and three ticket inspectors per run.

Tickets and passes. To travel on public transport, a ticket must be validated; it can be either a paper ticket or a travel pass. The ticket is the same for every type of vehicle and is valid for 90 minutes. Once validated, it can be used for a single run.

A travel pass can be used once a subscription has been purchased. There are 4 different subscriptions: 10-trip carnet, 50-trip carnet, monthly and yearly.

Users. Each travel pass is associated with a user, whose first name, last name and e-mail address are always known. When the pass expires and is not renewed, its owner is removed from the database.

Routes, stops and priority lanes. The transport network is divided into routes. For each route the database stores: an identification code, the start zone, the end zone and all the zones it crosses, the stops it is made of and the priority lanes along its path.

Each stop has a name and a code. For a given route, it is also known whether the stop is a start, end or intermediate stop.

Priority lanes are identified by their start stop and their end stop.

Zones. The city is divided into zones, numbered in increasing order starting from 1 (city centre). For each zone the database stores its size, number of inhabitants and employment data.

Runs. Each run is uniquely identified by its route and by the day and time it is travelled. For each run, the number and validation time of the tickets are known. At least one run per day is guaranteed for every route.

1.3 Glossary

Term	Definition	Synonym	Related to
Coverage (<i>copertura</i>)	Ratio between a route and the zones it crosses		Route, zones
Polluting impact (<i>impatto inquinante</i>)	Pollution per km of a vehicle × km of the route		Route, means of transport
Means of transport (<i>mezzo di trasporto</i>)		Vehicle	Route
Capacity (<i>capienza</i>)	Number of people a vehicle can carry	Load	Means of transport
Licence plate (<i>targa</i>)	3 characters (TRM, BUS, MET) + 8 digits (vehicle within the fleet) + 1 optional character (bus type)	Unique identification code of a vehicle	Means of transport, route
Priority lanes (<i>corsie prioritarie</i>)	Road lanes reserved for public transport vehicles		Route, stop
Travel pass (<i>tessera</i>)	Card with which a user can buy a subscription; each validation generates a ticket		Run, owner
Validate (<i>convalidare</i>)	To stamp a ticket	Stamp (<i>timbrare</i>)	Paper ticket, travel pass
Zone (<i>zona</i>)	Portion of the metropolitan city		Route
Transit (<i>transito</i>)	Passage through a given stop	Passage	Stop

Table 1: Glossary of terms.

1.4 Main operations

Table 2 lists the main operations the database must support, together with their execution type and estimated frequency.

ID	Operation	Type	Frequency
O1	Count the routes with more than two distinct priority lanes along their path	Interactive	2/month
O2	Ridership of a given route in a time slot (e.g. rush hour)	Interactive	2/month
O3	Removal of expired travel passes and of their owners	Batch	1/day
O4	Average travel time between two stops of a given run	Interactive	2/month
O5	Average number of routes per zone	Interactive	2/month

Table 2: Main operations.

1.5 Grouping of requirements into homogeneous sentences

The requirements text is split into groups of homogeneous sentences, each referring to a single concept.

General sentences

The goal is to design a relational database that supports ATM in a project aimed at optimising the city's public transport network on several fronts. The main purpose of the database is to provide fast and well-organised access to data on public transport traffic, so that the information needed for the optimisation can be derived easily.

The optimisation process will be based on a "quality measure" estimated from four types of information: the polluting impact of a route, its average duration, its costs and the coverage of the city it provides.

Sentences about means of transport

The network uses three different means of transport: underground (metro), buses and trams. For each vehicle the database stores its capacity, whether drivers and ticket inspectors are needed, an estimate of the pollution it produces and its licence plate (unique identification code).

As a generalisation, the underground is considered driverless (2 lines out of 5 already are). Ticket inspectors are considered unnecessary, since they are only present at station entrances (for a cost analysis, an estimate of 4 inspectors per stop can be used).

A tram requires one driver and three ticket inspectors per run.

Buses are divided into two categories according to their engine: internal combustion (diesel) vehicles and electric buses. Like trams, they require one driver and three ticket inspectors per run.

Sentences about tickets

To travel on public transport, a ticket must be validated; it can be either a paper ticket or a travel pass. The ticket is the same for every type of vehicle and is valid for 90 minutes. Once validated, it can be used for a single run.

A travel pass can be used once a subscription has been purchased. There are 4 different subscriptions: 10-trip carnet, 50-trip carnet, monthly and yearly.

Sentences about users

Each travel pass is associated with a user, whose first name, last name and e-mail address are always known. When the pass expires and is not renewed, its owner is removed from the database.

Sentences about routes

The transport network is divided into routes. For each route the database stores: an identification code, the start zone, the end zone and all the zones it crosses, the stops it is made of and the priority lanes along its path.

Sentences about stops

Each stop has a name and a code. For a given route, it is also known whether the stop is a start, end or intermediate stop.

Sentences about priority lanes

Priority lanes are identified by their start stop and their end stop.

Sentences about zones

The city is divided into zones, numbered in increasing order starting from 1 (city centre). For each zone the database stores its size, number of inhabitants and employment data.

Sentences about runs

Each run is uniquely identified by its route and by the day and time it is travelled. For each run, the number and validation time of the tickets are known. At least one run per day is guaranteed for every route.

2 Conceptual design

2.1 E-R schema

Figure 1 shows the Entity-Relationship schema derived from the requirements.

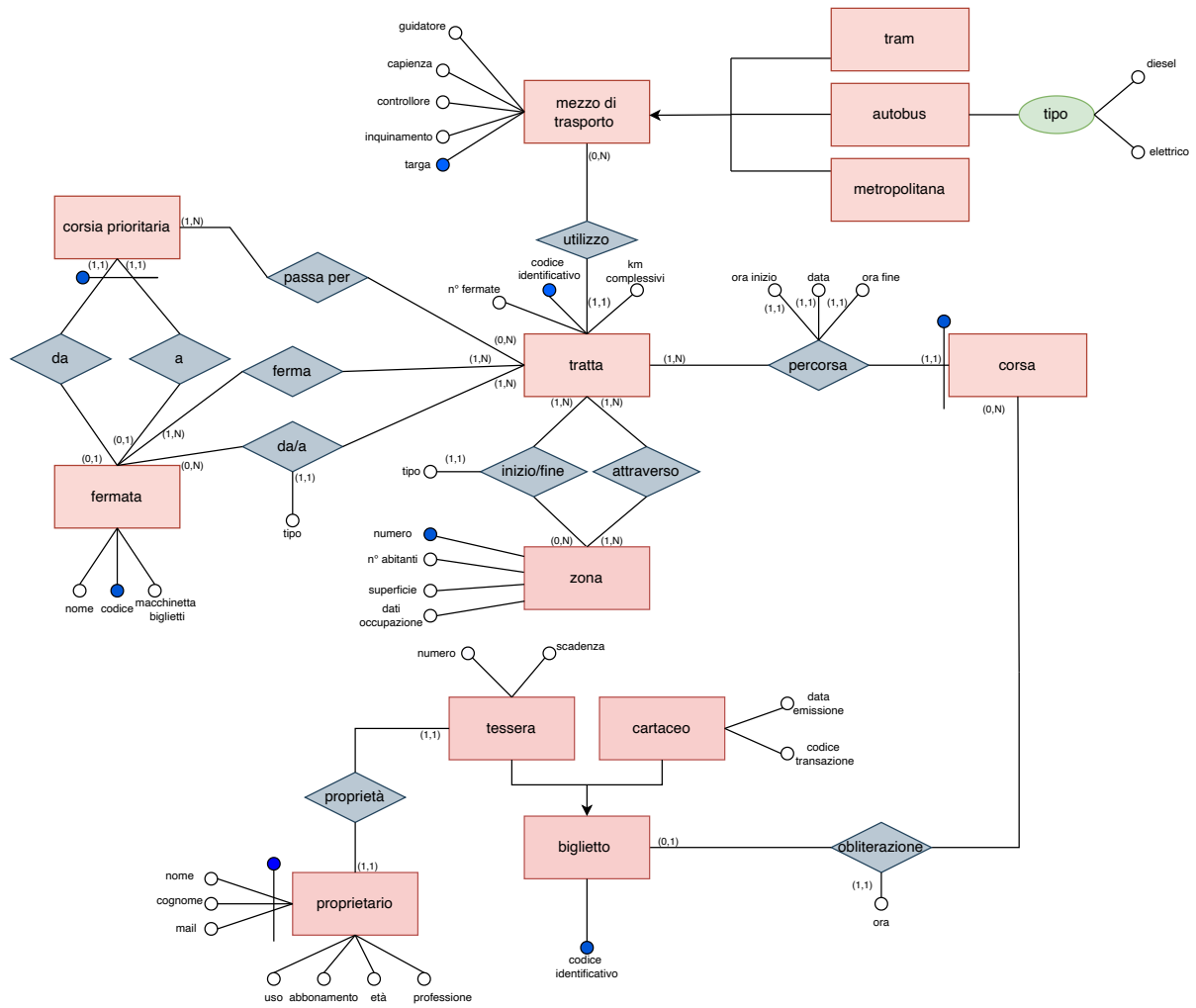


Figure 1: Initial E-R schema (labels in Italian, see the data dictionary below).

2.2 Data dictionary

2.2.1 Entities

Entity	Attributes	Identifier
Means of transport (<i>mezzo di trasporto</i>)	Licence plate, pollution, driver, capacity, inspector	Licence plate
Tram (<i>tram</i>)	Licence plate, pollution, driver, capacity, inspector	Licence plate
Underground (<i>metropolitana</i>)	Licence plate, pollution, driver, capacity, inspector	Licence plate
Bus (<i>autobus</i>)	Licence plate, pollution, driver, capacity, inspector, diesel, electric	Licence plate
Route (<i>tratta</i>)	ID, km, number of stops	ID
Zone (<i>zona</i>)	Number, number of inhabitants, area, employment data	Number
Run (<i>corsa</i>)		Percorsa (R)
Ticket (<i>biglietto</i>)	ID	ID
Travel pass (<i>tessera</i>)	ID, expiry date, number	ID
Owner (<i>proprietario</i>)	First name, last name, e-mail, subscription, usage, occupation, age	First name, last name, e-mail
Paper ticket (<i>cartaceo</i>)	ID, issue date, transaction code	ID
Stop (<i>fermata</i>)	Code, name, ticket machine	Code
Priority lane (<i>corsia prioritaria</i>)		Da (R), A (R)

Table 3: Data dictionary: entities.

2.2.2 Relationships

Relationship	Description	Components	Attributes
Utilizzo (<i>use</i>)	Vehicle used on a route	Means of transport, route	
Percorsa (<i>travelled</i>)	Specific run of a given route	Route, run	Start time, date, end time
Inizio/fine (<i>start/end</i>)	Start zone and end zone of a route	Route, zone	Type (start or end)
Attraverso (<i>crosses</i>)	Further zones crossed by a route	Route, zone	
Obliterazione (<i>validation</i>)	Use of a ticket on board a run	Run, ticket	Time
Proprietà (<i>ownership</i>)	A person owning a given travel pass	Travel pass, owner	
Da/a (<i>from/to</i>) (<i>route-stop</i>)	Start and end stops	Route, stop	Type (from or to)
Ferma (<i>stops at</i>)	Intermediate stops	Route, stop	
Passa per (<i>passes through</i>)	Priority lanes along a route	Route, priority lane	
Da (<i>from</i>)	Start stop	Priority lane, stop	
A (<i>to</i>)	End stop	Priority lane, stop	

Table 4: Data dictionary: relationships (Italian name as in the diagrams, English translation in parentheses).

2.3 Constraints not expressible in the schema

- C1. The end time of a run must be later than its start time.
- C2. At least one run per day must be guaranteed for every route.
- C3. Each priority lane must be defined by exactly one start stop and one end stop.
- C4. The validation time of a ticket must fall between the start time and the end time of the run.

Constraints C1, C3 and C4 are implemented through triggers (Section 4.3).

2.4 Volume table

Concept	Type	Volume
Means of transport	E	1630
Tram	E	200
Underground	E	80
Bus	E	1350
Route	E	145 (17 tram + 123 bus + 5 metro)
Zone	E	9
Run	E	8900 (1 bus/tram every 20 minutes, 1 metro every 5 minutes)
Ticket	E	800,000
Travel pass	E	500,000
Paper ticket	E	300,000
Stop	E	7122 (122 metro + 850 tram + 6150 bus)
Priority lane	E	70
Utilizzo	R	145
Percorsa	R	8900
Inizio/fine	R	290
Attraverso	R	145 (on average a route crosses 3 zones)
Proprietà	R	500,000
Obliterazione	R	378,300
Da/a	R	290
Ferma	R	7250 (50 per route, upper-bound estimate)
Passa per	R	85
Da	R	70
A	R	70

Table 5: Volume table (E = entity, R = relationship).

3 Logical design

3.1 Preliminary considerations

The (1,N) cardinality of the relationship “percorsa” on the side of the entity “route” guarantees that every route has at least one run. However, a minimum *daily* cardinality, ensuring at least one run per day for every route (constraint C2), cannot be expressed in the schema.

The (1,1) cardinality of the relationship “proprietà” on the side of the entity “owner” ensures that every person stored in the database is associated with a travel pass.

For convenience, I also add an ID attribute that uniquely identifies runs.

3.2 Redundancy analysis

Each redundancy is evaluated by estimating the number of accesses required by the operations involved, with and without the redundancy (R = read access).

3.2.1 Derived attribute: “number of stops”

The attribute “number of stops” of the entity “route” can be derived from the entity “stop”. I study this redundancy through operation O4, “average travel time between two stops of a given run”, assuming an ideal route with 20 stops and a frequency of twice a month.

Concept	Constr.	Accesses	Type
Run	E	1	R
Percorsa	R	1	R
Route	E	1	R
Total		3	

Table 6: O4 with redundancy.

Concept	Constr.	Accesses	Type
Run	E	1	R
Percorsa	R	1	R
Route	E	1	R
Stop	E	22	R
Da/a	R	2	R
Ferma	R	20	R
Total		47	

Table 7: O4 without redundancy.

Although the operation is executed only twice a month, the redundancy considerably reduces the number of accesses: 6 read accesses per month with the redundancy, compared with 94 without it. I therefore keep the redundancy.

3.2.2 Cycle between route, priority lane and stop

There is a further redundancy, caused by a cycle involving the entities “route”, “priority lane” and “stop” and the relationships “passa per”, “da”, “a”, “ferma” and “da/a”. I evaluate whether it is worth keeping the relationship “passa per” by estimating the accesses required by operation O1, “count the routes with more than two distinct priority lanes along their path”, with a frequency of twice a month. The underground is excluded from the search, so the routes considered are $145 - 5 = 140$.

Let:

- N be the number of stops of a route that are also the start or end stop of a priority lane ($0 \leq N \leq \text{number of stops of the route}$);
- M be the number of priority lanes crossed by a route ($0 \leq M \leq 5$, since a route has at most 5 distinct priority lanes along its path).

Concept	Constr.	Accesses	Type
Route	E	140	R
Passa per	R	M	R
Priority lane	E	M	R

Table 8: O1 with redundancy.

Concept	Constr.	Accesses	Type
Route	E	140	R
Ferma	R	N	R
Da/a	R	N	R
Stop	E	N	R
Da/a	R	N	R
Priority lane	E	N	R

Table 9: O1 without redundancy.

Here too the difference is clear: removing the cycle, the monthly read accesses go from $280 + 4M$ to $280 + 10N$. It is therefore convenient to keep the relationship “passa per”, in order to limit the accesses required by this kind of operation.

3.3 Removal of generalisations

Means of transport. The first generalisation has “means of transport” as parent entity and “tram”, “bus” and “underground” as child entities. Only one child entity has its own attributes: “bus”, with the multi-valued attribute “type”. I merge the child entities into the parent entity, adding a new attribute “type” with three possible values: underground, tram, bus. The handling of the multi-valued attribute originally associated with “bus” is described in the next section.

Ticket. The second generalisation involves the parent entity “ticket” and the child entities “travel pass” and “paper ticket”. Here, keeping “ticket” is essential for quantitative analyses of route ridership (e.g. finding the run with the highest ridership of the day). On the other hand, removing the child entities would be logically inconsistent, since they are strongly differentiated and have attributes that cannot be compared. I therefore keep all the entities involved and add two new relationships (“tipologia 1” and “tipologia 2”) linking the child entities to the parent entity.

3.4 Multi-valued attributes

The only multi-valued attribute belongs to the entity “bus” and specifies the engine type. Introducing an additional entity to represent it is unnecessary: instead, I add an attribute “propulsion” (*propulsione*) to the entity “means of transport”, which takes the value “electric” for trams and the underground.

3.5 Choice of identifiers

In the restructured schema (Figure 2) the identifier of each entity is marked with a blue dot:

- **means of transport** (which merges bus, tram and underground): licence plate;
- **route**: identification code (ID);
- **run**: route (external identifier);
- **ticket**: identification code (ID);
- **paper ticket**: issue date + transaction code;
- **travel pass**: number;
- **owner**: first name + last name + e-mail;
- **zone**: number;
- **stop**: code;
- **priority lane**: start and end stops (external identifier).

3.6 Restructured E-R schema

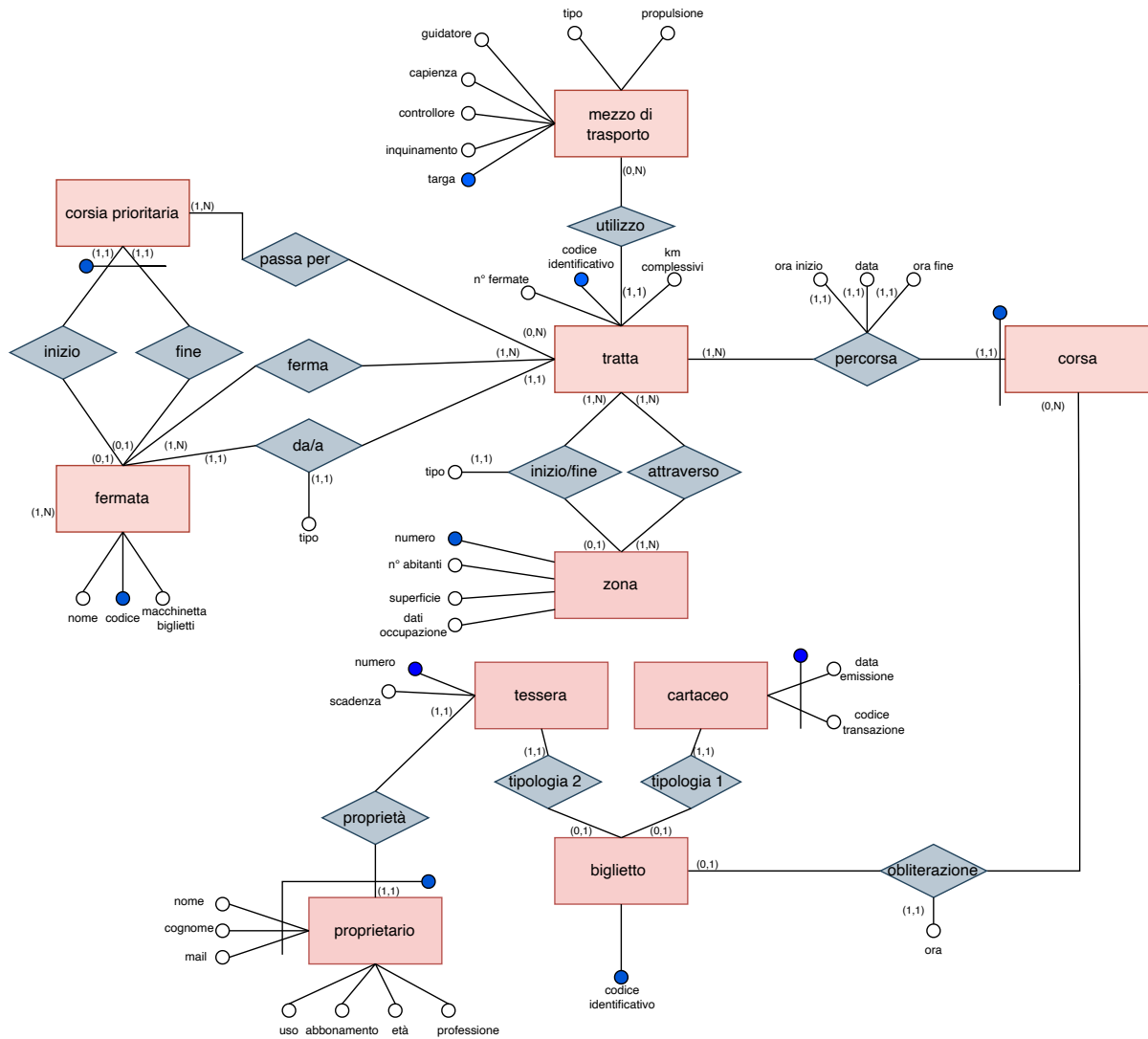


Figure 2: Restructured E-R schema.

3.7 Translation into the relational model

The restructured schema is translated into the following relations (primary keys are underlined; names are those used in the SQL implementation):

- Mezzo_di_trasporto (targa, controllore, capienza, inquinamento, guidatore, tipo, propulsione) – *means of transport*
- Tratta (ID, km, n_fermate, mezzo, zona_attraversata, inizio, fine) – *route*
- Zona (numero, n_abitanti, superficie, dati_occupazione) – *zone*
- Corsa (ID, ora_inizio, data, ora_fine, tratta) – *run*
- Biglietto (ID, corsa, ora_obliterazione) – *ticket*
- Zona_di_inizio_o_fine (tratta, zona, tipo) – *start or end zone*
- Fermata (codice, macchinetta_biglietti, nome) – *stop*

- Tratta_Fermata (tratta, fermata) – *route-stop*
- Corsia_prioritaria (ID, inizio_corsia, fine_corsia) – *priority lane*
- Tratta_CP (CP, tratta) – *route-priority lane*
- Tessera (numero, data_scadenza, obliterazione_tessera) – *travel pass*
- Biglietto_cartaceo (data_emissione, codice_transazione, obliterazione) – *paper ticket*
- Proprietario (nome, cognome, mail, tessera, uso, abbonamento, età, professione) – *owner*

3.8 Logical schema

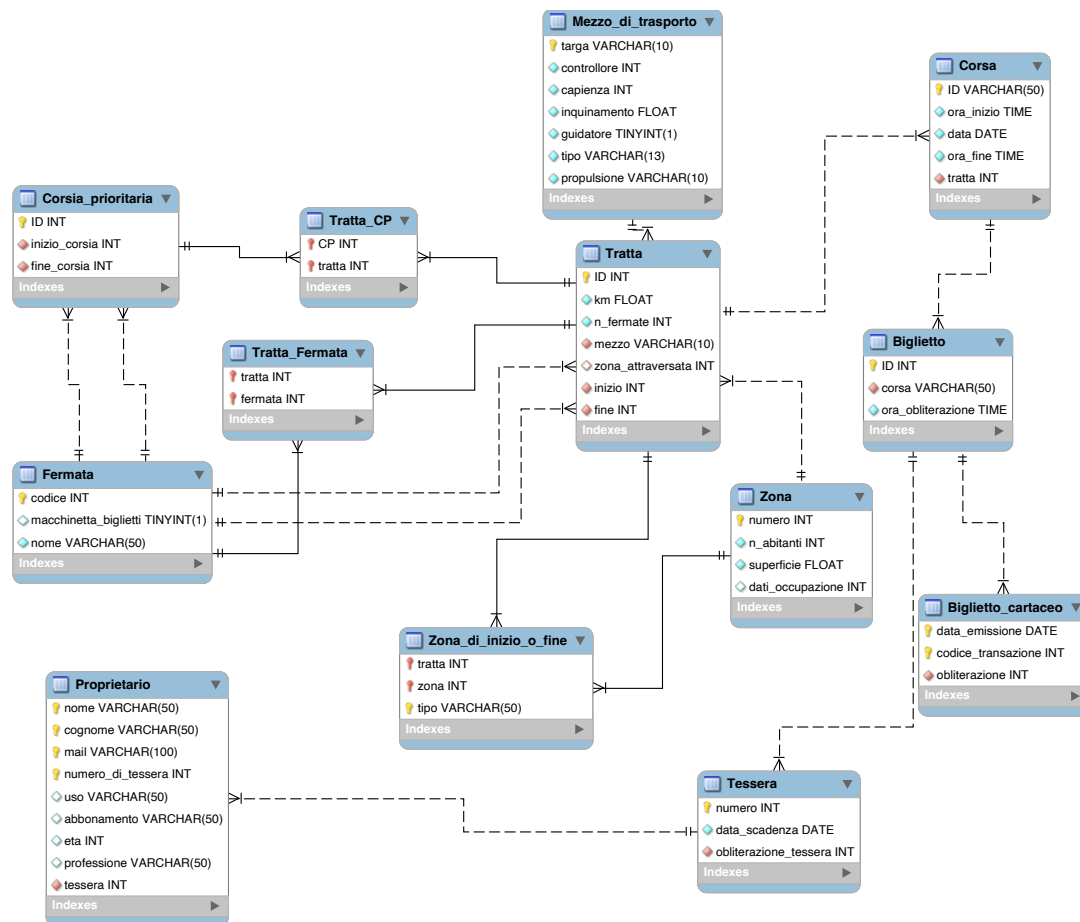


Figure 3: Logical schema.

3.9 Normalisation

- **First normal form.** All attributes are atomic: each column holds a single value and there are no repeating groups.
- **Second normal form.** The second normal form is satisfied: non-key attributes depend on the whole primary key.
- **Third normal form.** The third normal form is also satisfied: there are no transitive dependencies between non-key attributes.

4 SQL implementation

4.1 Table creation

I do not consider any CREATE statement particularly relevant to report in this section. The full SQL code used to create the database is available in the GitHub repository: <https://github.com/marghenecchi/Database-project.git>

4.2 Operations

Since all the implemented operations were identified as frequent, each of them is implemented as a stored procedure saved in the DBMS.

Operation O1 – Routes with more than two distinct priority lanes

```
DELIMITER //
CREATE PROCEDURE tratte2cp()
BEGIN
    SELECT COUNT(*) AS NumeroTratte
    FROM (
        SELECT tratta
        FROM Tratta_CP
        GROUP BY tratta
        HAVING COUNT(DISTINCT CP) > 2
    ) AS tratte2cp;
END //
```

Operation O2 – Ridership of a route in a time slot

The procedure counts the tickets validated on the runs of a given route that start within the given time slot on a given day. If the time slot is invalid (start time later than end time), an error is raised.

```
CREATE PROCEDURE affluenza(IN tratta_id INT, IN orario_inizio TIME,
IN orario_fine TIME, IN giorno DATE)
BEGIN
    IF orario_inizio > orario_fine THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid time slot';
    END IF;

    SELECT COUNT(*) AS affluenza
    FROM Biglietto b
    JOIN Corsa c ON b.corsa = c.ID
    WHERE c.tratta = tratta_id
        AND c.ora_inizio BETWEEN orario_inizio AND orario_fine
        AND c.data = giorno;
END //
```

Operation O3 – Removal of expired travel passes (batch)

The procedure first deletes the owners of expired travel passes and then the passes themselves. An event runs it automatically once a day (this requires the MySQL `event_scheduler` to be enabled).

```
CREATE PROCEDURE rimuovi_tessere_scadute()
BEGIN
    DELETE FROM Proprietario
    WHERE tessera IN (
        SELECT numero FROM Tessera WHERE data_scadenza < CURDATE()
    );

    DELETE FROM Tessera
    WHERE data_scadenza < CURDATE();
END //

CREATE EVENT pulizia_tessere
ON SCHEDULE EVERY 1 DAY
DO CALL rimuovi_tessere_scadute() //
```

Operation O4 – Average travel time between two stops

The average time between two consecutive stops of a run is the duration of the run divided by the number of intervals between the stops of its route. The procedure uses the redundant attribute `n_fermate`, which was kept precisely for this operation (Section 3.2.1).

```
CREATE PROCEDURE TempoPercorrenzaMedio(IN corsa_id VARCHAR(50))
BEGIN
    SELECT TIME_TO_SEC(TIMEDIFF(c.ora_fine, c.ora_inizio)) / 60
           / NULLIF(t.n_fermate - 1, 0)
    AS TempoMedioPercorrenza
    FROM Corsa c
    JOIN Tratta t ON c.tratta = t.ID
    WHERE c.ID = corsa_id;
END //
```

Operation O5 – Average number of routes per zone

The procedure counts the routes crossing each zone and averages the counts over all zones (zones with no routes count as zero).

```
CREATE PROCEDURE nMedioPerZona()
BEGIN
    SELECT AVG(IFNULL(TrattePerZona.NumeroTratte, 0))
    AS nMedioTratte
    FROM Zona
    LEFT JOIN (
        SELECT zona_attraversata, COUNT(*) AS NumeroTratte
        FROM Tratta
        GROUP BY zona_attraversata
    ) AS TrattePerZona
    ON Zona.numero = TrattePerZona.zona_attraversata;
END //
```

```
DELIMITER ;
```

Additional operation – Inserting a new run

Insertion of a run made on 06/06/2024 at 9:30 on route 94.

```
INSERT INTO Corsa (ID, ora_inizio, data, ora_fine, tratta)
VALUES ('C026', '09:30:00', '2024-06-06', '12:23:04', 94);
```

4.3 Triggers

Trigger 1 – Consistent run times (constraint C1)

When a run is inserted, it checks that the end time is later than the start time.

```
DELIMITER //
CREATE TRIGGER check_corsa BEFORE INSERT ON Corsa
FOR EACH ROW
BEGIN
    IF NEW.ora_fine <= NEW.ora_inizio THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid times';
    END IF;
END //
```

Trigger 2 – Ticket validation time (constraint C4)

When a validated ticket is inserted, it checks that the validation falls within the time span of the run.

```
CREATE TRIGGER ora_biglietto BEFORE INSERT ON Biglietto
FOR EACH ROW
BEGIN
    DECLARE inizio_corsa TIME;
    DECLARE fine_corsa TIME;

    SELECT ora_inizio, ora_fine INTO inizio_corsa, fine_corsa
    FROM Corsa WHERE ID = NEW.corsa;

    IF NEW.ora_obliterazione < inizio_corsa OR
    NEW.ora_obliterazione > fine_corsa THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid validation time';
    END IF;
END //
```

Trigger 3 – Stops of a priority lane (constraint C3)

When a priority lane is inserted, it checks that the lane is defined by a start stop and an end stop that are both present and distinct. Since each of them is a single attribute of

`Corsia_prioritaria`, this guarantees exactly one start stop and one end stop; their existence is ensured by the foreign keys to `Fermata`.

```
CREATE TRIGGER fermate_cp BEFORE INSERT ON Corsia_prioritaria
FOR EACH ROW
BEGIN
    IF NEW.inizio_corsia IS NULL OR NEW.fine_corsia IS NULL
        OR NEW.inizio_corsia = NEW.fine_corsia THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'invalid_stops';
    END IF;
END //
DELIMITER ;
```

4.4 Python interaction

Finally, a short Python script that interacts with the `atm` database.

Database connection

```
import mysql.connector
from mysql.connector import Error

connection = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="*****",
    database="atm"
)
```

Stored procedure: percentage of paper tickets

Creation of a stored procedure that computes the percentage of paper tickets used in a day with respect to all tickets.

```
try:
    cursor = connection.cursor()

    create_sp_query = """
CREATE PROCEDURE perc_cartacei(IN giorno DATE)
BEGIN
    DECLARE totale_biglietti INT;
    DECLARE biglietti_cartacei INT;
    DECLARE percentuale DECIMAL(5,2);

    SELECT COUNT(*) INTO totale_biglietti
    FROM Biglietto b
    JOIN Corsa c ON b.corsa = c.ID
    WHERE c.data = giorno;

    SELECT COUNT(*) INTO biglietti_cartacei
    FROM Biglietto_cartaceo bc
    JOIN Biglietto b ON bc.oblitterazione = b.ID
    JOIN Corsa c ON b.corsa = c.ID
```

```
WHERE c.data = giorno;

IF totale_biglietti > 0 THEN
    SET percentuale = (biglietti_cartacei / totale_biglietti) * 100;
ELSE
    SET percentuale = 0;
END IF;

SELECT percentuale AS PercentualeBigliettiCartacei;
END
"""
cursor.execute(create_sp_query)
print("Stored procedure created successfully :)")

except Error as err:
    print("Error:", err)
```

Usage example

Computing the percentage for 26/05/2003, my birthday. The stored procedure is invoked with `callproc` and its result is read from `stored_results()`; the connection is closed only at the end.

```
try:
    cursor.callproc("perc_cartacei", ("2003-05-26",))
    for result in cursor.stored_results():
        print(result.fetchall())

except Error as err:
    print("Error:", err)

finally:
    if connection.is_connected():
        cursor.close()
        connection.close()
```