PROGETTO BASI DI DATI

Francesco De Matteis, 163439

Pavan Matteo, 163853

Roncadin Alessandro, 161904

Schiavoni Alex, 162475

Consegna

Si vuole progettare una base di dati di supporto alle attività di un’azienda alimentare di carni.

L’azienda opera su un certo numero di mercati, che possono essere sia **macrozone** (Triveneto, Isole) che singole **regioni** (Lombardia, Piemonte, etc.). Ogni **cliente** appartiene ad uno specifico **mercato**. I clienti sono seguiti da **agenti**, coordinati da **capi-area**. Le vendite sono fatte su **articoli**, che sono suddivisi in opportune **linee di prodotto**. Ogni cliente che intenda comprare della carne effettua un **ordine** in cui vengono specificati gli articoli che si vogliono acquistare e il loro prezzo. Gli articoli venduti appartengono a opportune **partite** di animali acquistate da un certo insieme di **fornitori**. Le partite possono essere di prima, seconda e terza scelta. Ogni partita contiene articoli appartenenti ad un’unica linea di prodotto ed è identificata univocamente da un opportuno codice. Nell’ambito di ciascuna partita, ogni articolo è identificato univocamente da un codice numerico (primo articolo della partita, secondo articolo della partita, etc.). Ad ogni partita è inoltre associato un costo diretto, suddiviso in costo di acquisto, spedizione e stoccaggio. Il costo del singolo articolo si ottiene dividendo il costo della partita cui appartiene per il numero di articoli in essa contenuti.

Si definisca uno schema Entità-Relazioni che descriva il contenuto informativo del sistema, illustrando con chiarezza le eventuali assunzioni fatte. Lo schema dovrà essere completato con attributi ragionevoli per ciascuna entità (identificando le possibili chiavi) e relazione. Vanno specificati accuratamente i vincoli di cardinalità e partecipazione di ciascuna relazione.

Documento di Specifiche

Si vuole progettare una base di dati per supportare le attività di un'azienda alimentare di carni. Le caratteristiche principali sono:

- **Mercati**: L'azienda opera su vari mercati, che possono essere **Macrozone** (es. Triveneto, Isole) o **Regioni** (es. Lombardia, Piemonte). Ogni mercato è quindi o una Macrozona o una Regione.
- **Clienti**: Ogni cliente appartiene a uno specifico mercato ed è seguito da un **Agente**. Gli agenti sono coordinati da **CapiArea**.
- **Articoli e Linee di Prodotto**: Le vendite riguardano articoli suddivisi in linee di prodotto. Gli articoli venduti provengono da **Partite** di animali acquistate da fornitori.
- **Partite**: Ogni partita è acquistata da un fornitore, appartiene a una sola linea di prodotto ed è di una determinata **Qualità** (Prima, Seconda o Terza scelta). Ogni partita ha un costo diretto composto da costo di acquisto, spedizione e stoccaggio.
- **Articoli**: All'interno di una partita, ogni articolo è identificato univocamente da un codice numerico sequenziale.
- **Ordini**: I clienti effettuano ordini specificando gli articoli che desiderano acquistare e il loro prezzo.

Glossario dei termini

Verranno riportati i termini fondamentali con una breve descrizione e i vari collegamenti.

TERMINI	DESCRIZIONE	COLLEGAMENTI
Azienda	Dominio della base di dati; l'azienda alimentare di carni	Mercato, Cliente, Fornitore, Partita, Articolo
Mercato	Area geografica in cui opera l'azienda, può essere una Macrozona o una Regione.	Cliente, Macrozona, Regione
Macrozona	Grande area geografica che comprende più regioni (es. Triveneto, Isole).	Mercato
Regione	Singola regione geografica (es. Lombardia, Piemonte).	Mercato
Cliente	Entità che acquista prodotti dall'azienda.	Mercato, Agente, Ordine
Agente	Persona che segue i clienti; coordinato da un CapoArea.	Cliente, CapoArea
CapoArea	Responsabile che coordina gli Agenti.	Agente
Fornitore	Entità che fornisce le partite di animali all'azienda.	Partita
Linea di Prodotto	Categoria di articoli prodotti dall'azienda (es. carne bovina, suina).	Partita, Articolo
Partita	Lotto di animali acquistato da un Fornitore; contiene articoli di una sola Linea di Prodotto e ha associata una Qualità.	Fornitore, Articolo
Articolo	Singolo prodotto venduto ai clienti; parte di una Partita.	Partita, Ordine
Ordine	Richiesta di acquisto effettuata da un Cliente; specifica gli Articoli acquistati e il loro prezzo.	Cliente, Articolo
Qualità	Classificazione delle Partite in base alla scelta (Prima, Seconda, Terza).	Partita
Costo	Spesa associata all'acquisto, spedizione e stoccaggio delle Partite; il costo dell'Articolo è il costo della Partita diviso per il numero di Articoli nella Partita.	Partita, Articolo

Strutturazione dei Requisiti

Di seguito verranno presentati i dettagli del dominio di tutti gli elementi presenti nella tabella del Glossario.

Frasi di natura generale:

- Si vuole progettare una base di dati per supportare le attività di un'azienda alimentare di carni.

Frasi relative a Mercato:

- L'azienda opera su un certo numero di mercati, che possono essere sia Macrozone che singole Regioni.
- Ogni mercato è una Macrozona o una Regione.

Frasi relative a Macrozona e Regione:

- Le Macrozone e le Regioni sono tipi specifici di Mercato.

Frasi relative a Cliente:

- Ogni cliente appartiene a uno specifico mercato.
- I clienti sono seguiti da agenti.

Frasi relative ad Agente:

- Gli agenti seguono i clienti.
- Gli agenti sono coordinati da capi-area.

Frasi relative a CapoArea:

- I capi-area coordinano gli agenti.

Frasi relative a Fornitore:

- Le partite sono acquistate da un certo insieme di fornitori.

Frasi relative a Linea di Prodotto:

- Gli articoli sono suddivisi in linee di prodotto.

Frasi relative a Partita:

- Ogni partita contiene articoli appartenenti a un'unica linea di prodotto.
- Le partite possono essere di prima, seconda e terza scelta.
- Ogni partita è identificata univocamente da un codice.
- Ad ogni partita è associato un costo diretto suddiviso in costo di acquisto, spedizione e stoccaggio.

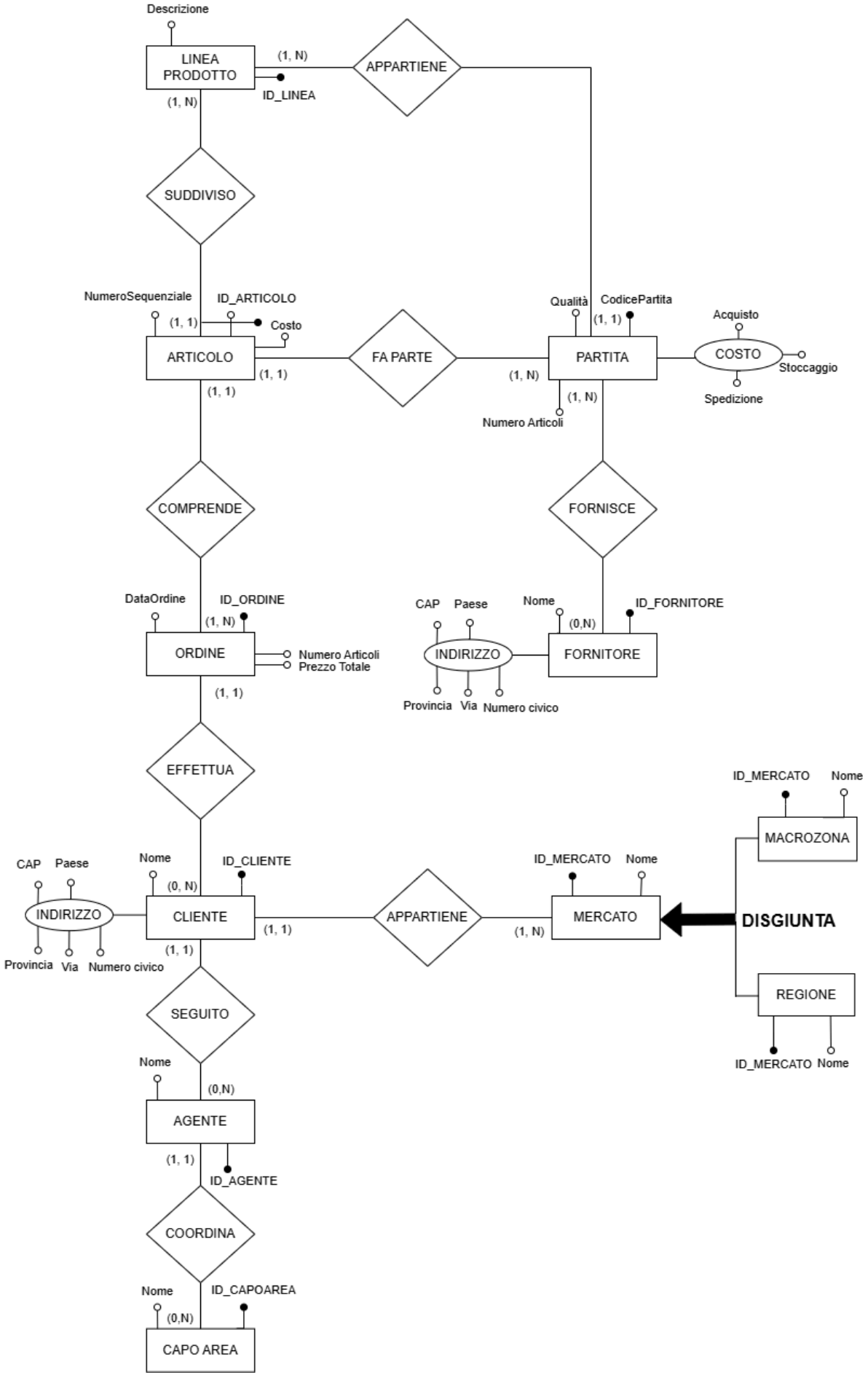
Frasi relative ad Articolo:

- Nell’ambito di ciascuna partita, ogni articolo è identificato univocamente da un codice numerico.
- Il costo del singolo articolo si ottiene dividendo il costo della partita per il numero di articoli in essa contenuti.

Frasi relative ad Ordine:

- Ogni cliente che intende acquistare della carne effettua un ordine.
- Negli ordini vengono specificati gli articoli che si vogliono acquistare e il loro prezzo.

Schema Entità-Relazione (ER)



Vincoli di integrità:

A questo punto verranno specificati i vincoli aggiuntivi aggiungendo requisiti non richiesti dalla consegna, ma dedotti.

1. **Generalizzazione di Mercato in Macrozona e Regione:**
    - **Disgiunzione**: Un mercato può essere o una **Macrozona** o una **Regione**, ma non entrambe.
    - **Totalità**: Ogni mercato deve essere o una Macrozona o una Regione.
2. **Ciclo partita-articolo-lineaProdotto:**
    - Una partita comprende solo articoli di una determinata linea, non è possibile che ci siano degli articoli che non hanno una linea di appartenenza nella stessa partita.
3. **Integrità dei Costi della Partita:**
    - **CostoAcquisto**, **CostoSpedizione**, **CostoStoccaggio**: devono essere non negativi.
    - **CostoTotale** della **Partita** è calcolato come la somma di **CostoAcquisto**, **CostoSpedizione**, e **CostoStoccaggio**.
    - **Costo dell'Articolo**: Derivato dividendo il **CostoTotale** della partita per il **NumeroArticoli**.
4. **Vincoli di Qualità per la Partita:**
    - **Qualità** della partita è vincolata a valori specifici: Prima Scelta, Seconda Scelta, Terza Scelta, per garantire la categorizzazione coerente delle partite.
5. **Indirizzo:**
    - L'entità **Indirizzo** contiene attributi come **CAP**, **Paese**, **Provincia**, **Via**, e **Numero Civico** che garantiscono che i dati di localizzazione siano strutturati e consistenti, ad esempio il CAP deve contenere esattamente 5 cifre
6. **DataOrdine e Quantità degli Articoli nell'Ordine:**
    - **DataOrdine** per **Ordine** deve essere una data valida e specifica quando l'ordine è stato effettuato.
    - **PrezzoTotale** di un ordine è derivato dalla somma del prezzo di ciascun articolo moltiplicato per la quantità nell'ordine.
7. **Relazione Comprende**
    - Ogni articolo che viene prodotto viene anche venduto

Tabella dei volumi:

La seguente tabella specifica il numero stimato di istanze per ogni entità (E) e relazione (R) dello schema, i valori sono approssimati.

CONCETTO	TIPO	VOLUME
Cliente	E	2000
Agente	E	50
Capo Area	E	20
Mercato	E	20
Macrozona	E	5
Regione	E	20
Ordine	E	5000
Articolo	E	10000
Linea Prodotto	E	10
Partita	E	1000
Fornitore	E	100
Coordina	R	50
Seguito	R	2000
Appartiene (Cliente - Mercato)	R	2000
Effettua	R	4000
Comprende	R	2000
Suddiviso	R	1000
Appartiene (LineaProdotto - Partita)	R	100
Fa parte	R	10000
Fornisce	R	1000

Suddiviso = Articoli / Linea Prodotto quindi avremo 1000 prodotti per ogni linea Prodotto
Appartiene (LineaProdotto - Partita) = Partita / Linea Prodotto

Analisi delle ridondanze:

Nello schema ci sono quattro ridondanze:

- Ordine
    - PrezzoTotale
    - NumeroArticoli
- Partita
    - Costo
    - NumeroArticoli

Tabella dei volumi:

CONCETTO	TIPO	VOLUME
Ordine	Entità	5000
Articolo	Entità	10000
Comprende	Relazione	2000
Partita	Entità	1000
Fa Parte	Relazione	10000

Tabella delle operazioni:

OPERAZIONE	FREQUENZA
1) Calcolo di **PrezzoTotale** per ogni ordine	100 volte alla settimana
2) Calcolo di **NumeroArticoli** per ogni ordine	100 volte alla settimana
3) Calcolo di **Costo** per ogni articolo in una partita	50 volte alla settimana
4) Controllo del **NumeroArticoli** in una partita	50 volte alla settimana

Costi operazioni:

1) Calcolo di **PrezzoTotale** per ogni ordine

In presenza di ridondanza:

CONCETTO	TIPO	N. ACCESSI	TIPO ACCESSO
Ordine	Entità	1	R

- *Costo*: 100 * 1R= 100 Read a settimana

In assenza di ridondanza:

(Prendiamo in considerazione una media di 2000 articoli per ordine dato da *Articolo/Ordine*. La richiesta dovrà passare dalla relazione “Comprende” e dall’entità “Articolo”)

CONCETTO	TIPO	N. ACCESSI	TIPO ACCESSO
Ordine	Entità	1	R
Comprende	Relazione	2000	R
Articolo	Entità	2000	R

- *Costo*: 100 * (1 + 2000+2000)R= 400100 Read a settimana

2) Calcolo di **NumeroArticoli** per ogni ordine

In presenza di ridondanza:

CONCETTO	TIPO	N. ACCESSI	TIPO ACCESSO
Ordine	Entità	1	R

- *Costo*: 100 * 1R= 100 Read a settimana

In assenza di ridondanza:

(Prendiamo in considerazione una media di 2000 articoli per ordine dato da *Articolo/Ordine*. La richiesta dovrà passare dalla relazione “Comprende” e dall’entità “Articolo”)

CONCETTO	TIPO	N. ACCESSI	TIPO ACCESSO
Ordine	Entità	1	R
Comprende	Relazione	2000	R
Articolo	Entità	2000	R

- *Costo*: 100 * (1 + 2000+ 2000)R= 400100 Read a settimana

3) Calcolo di **Costo** per ogni articolo in una partita

In presenza di ridondanza:

CONCETTO	TIPO	N. ACCESSI	TIPO ACCESSO
Partita	Entità	1	R

- *Costo*: 50 * 1R= 50 Read a settimana

In assenza di ridondanza:

(Consideriamo una media di 10 articoli per partita)

CONCETTO	TIPO	N. ACCESSI	TIPO ACCESSO
Articolo	Entità	10	R
Fa parte	Relazione	10	R
Partita	Entità	1	R

- *Costo*: 50 * (10 + 10 + 1)R= 1050 Read a settimana

4) Controllo del **NumeroArticoli** in una partita

In presenza di ridondanza:

CONCETTO	TIPO	N. ACCESSI	TIPO ACCESSO
Partita	Entità	1	R

- *Costo*: 50 * 1R= 50 Read a settimana

In assenza di ridondanza:

CONCETTO	TIPO	N. ACCESSI	TIPO ACCESSO
Articolo	Entità	10	R
Fa parte	Relazione	10	R
Partita	Entità	1	R

- *Costo*: 50 * (10 + 10 + 1)R= 1050 Read a settimana

Confronto dei Costi:

Confronteremo ora i costi delle operazioni in presenza e in assenza di ridondanza, per capire se converrà mantenere le informazioni ridondanti oppure no.

1) Calcolo di **PrezzoTotale** per ogni ordine

   In presenza di ridondanza: 100 accessi settimanali

   In assenza di ridondanza: 400100 accessi settimanali

2) Calcolo di **NumeroArticoli** per ogni ordine

   In presenza di ridondanza: 100 accessi settimanali

   In assenza di ridondanza: 400100 accessi settimanali

3) Calcolo di **Costo** per ogni articolo in una partita

   In presenza di ridondanza: 50 accessi settimanali

   In assenza di ridondanza: 1050 accessi settimanali

4) Controllo del **NumeroArticoli** in una partita

   In presenza di ridondanza: 300 accessi settimanali

   In assenza di ridondanza: 1050 accessi settimanali

Conclusione: Per tutti i casi analizzati singolarmente, l'assenza di ridondanza comporta un numero significativamente maggiore di accessi settimanali. Questo suggerisce che mantenere le informazioni ridondanti è vantaggioso per ridurre i costi di accesso, a meno che la ridondanza non introduca altri problemi di coerenza o spazio di archiviazione.

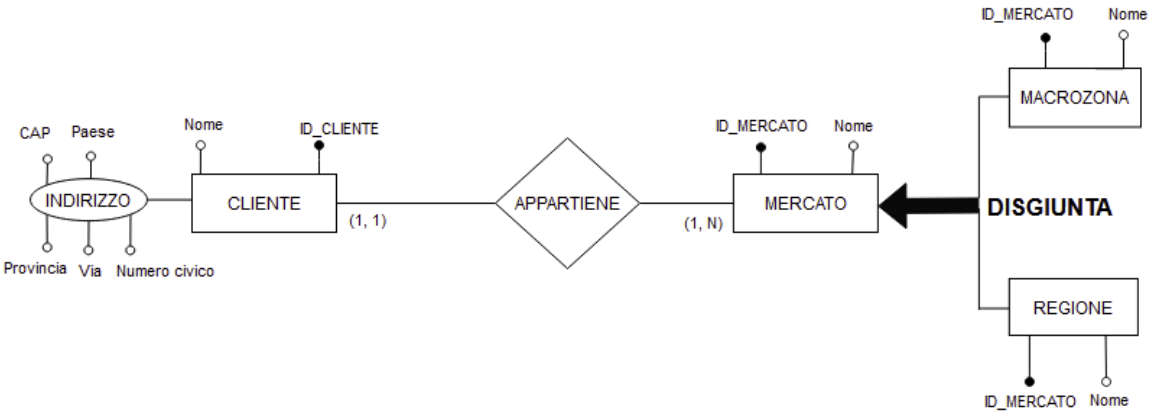
RISTRUTTURAZIONE ER

Eliminazione delle generalizzazioni:

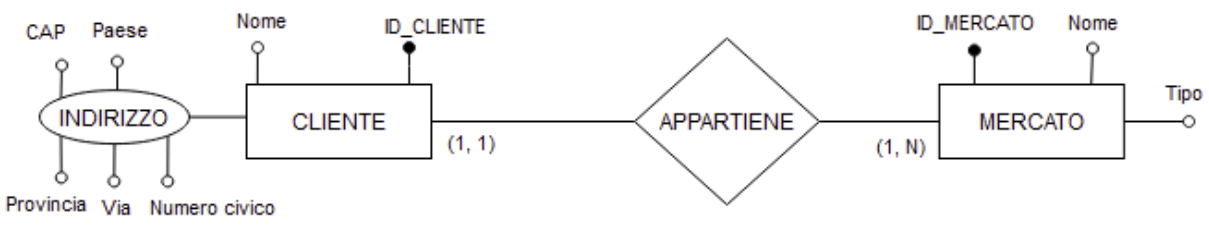
Nel nostro schema, la generalizzazione di **Mercato** in **Regione** e **Macrozona** è stata eliminata per adattarsi al modello relazionale. Abbiamo scelto di **accorpare le figlie nel genitore**, consolidando le informazioni di **Regione** e **Macrozona** in una singola entità **Mercato**. Questo implica che Regione e Macrozona non esistono più come tabelle separate; invece, l'entità Mercato include un nuovo attributo **Tipo**, che indica se il mercato è una Regione o una Macrozona. Gli attributi originari di Mercato come ID_Mercato e Nome restano invariati. Inoltre, tutte le relazioni che facevano riferimento a Regione o Macrozona, come **APPARTIENE (Cliente - Mercato)**, restano legate a Mercato, semplificando lo schema e riducendo il numero di tabelle e relazioni. Grazie a questo approccio:

- Le query possono essere eseguite direttamente su **Mercato**, senza bisogno di join complessi
- Gli attributi condivisi da **Regione** e **Macrozona** sono memorizzati una sola volta, evitando duplicazioni e garantendo coerenza.

CON GENERALIZZAZIONI:



SENZA GENERALIZZAZIONI:



Selezione identificatori primari:

ENTITA’	IDENTIFICATORE PRIMARIO
Mercato	ID_MERCATO
Regione	ID_MERCATO
Macrozone	ID_MERCATO

Alla specializzazione di **Mercato** sono collegate le entità **Regione** e **Macrozone** che non conviene mantenere separate quindi vengono raggruppate nell’entità **Mercato** e viene inserito l’attributo “Tipo” che può assumere solo i valori “Regione” e “Macrozona”, mentre l’attributo “Nome” comprende l’effettivo nome del mercato (Triveneto, Isole, Friuli Venezia Giulia, Veneto, Piemonte).

SCHEMA LOGICO:

- **CAPO AREA** (ID_CAPOAREA, Nome)
- **AGENTE** (ID_AGENTE, *ID_CAPO_AREA*, Nome)
  CapoArea ForeignKey
- **CLIENTE** (ID_CLIENTE, *ID_MERCATO*, *ID_AGENTE*, Nome, CAP, Provincia, Via, NumeroCivico, Paese)
  Agente, Mercato ForeignKey
- **MERCATO** (ID_MERCATO, Nome, Tipo)
- **ORDINE** (ID_ORDINE, *ID_CLIENTE*, DataOrdine, NumeroArticoli, PrezzoTotale)
  Cliente ForeignKey
- **ARTICOLO** (ID_ARTICOLO, *ID_LINEA*, *ID_ORDINE*, *CODICE_PARTITA*, NumeroSequenziale, Costo)
  Ordine, LineaProdotto, Partita ForeignKey
- **LINEA PRODOTTO** (ID_LINEA, Descrizione)
- **PARTITA** (CODICE_PARTITA, *ID_LINEA*, Qualità, NumeroArticoli, Acquisto, Stoccaggio, Spedizione)
  LineaProdotto ForeignKey
- **FORNISCE**(*CODICE_PARTITA, ID_FORNITORE*)
  Partita, Fornitore ForeignKey
- **FORNITORE** (ID_FORNITORE, Nome, CAP, Indirizzo, Provincia, Via, NumeroCivico, Paese)

Vincoli di chiave esterna:

AGENTE(CapoArea) → CAPO_AREA(ID_CAPOAREA)
CLIENTE(Agente) → AGENTE(ID_AGENTE)
CLIENTE(Mercato) → MERCATO(ID_MERCATO)
ORDINE(Cliente) → CLIENTE(ID_CLIENTE)
ARTICOLO(Ordine) → ORDINE(ID_ORDINE)
ARTICOLO(LineaProdotto) → LINEA PRODOTTO(ID_LINEA)
ARTICOLO(Partita) → PARTITA(CODICEPARTITA)
PARTITA(LineaProdotto) → LINEA PRODOTTO(ID_LINEA)
LINEA PRODOTTO(Partita) → PARTITA(CodicePartita)
FORNISCE(Partita) → PARTITA(CodicePartita)
FORNISCE(Fornitore) → FORNITORE(ID_FORNITORE)

Progettazione Fisica:

CAPO_AREA

```
CREATE TABLE CAPO_AREA (
    ID_CAPOAREA INT PRIMARY KEY,
    Nome VARCHAR(255) NOT NULL
);
```

AGENTE

```
CREATE TABLE AGENTE (
    ID_AGENTE INT PRIMARY KEY,
    ID_CAPO_AREA INT NOT NULL,
    Nome VARCHAR(255) NOT NULL,
    CONSTRAINT fk_agente_capoarea
        FOREIGN KEY (ID_CAPO_AREA) REFERENCES CAPO_AREA(ID_CAPOAREA)
            ON UPDATE CASCADE
            ON DELETE RESTRICT
);
```

CLIENTE

```
CREATE TABLE CLIENTE (
    ID_CLIENTE INT PRIMARY KEY,
    ID_MERCATO INT NOT NULL,
    ID_AGENTE INT NOT NULL,
    Nome VARCHAR(255) NOT NULL,
    CAP CHAR(5) NOT NULL CHECK (CAP ~ '^[0-9]{5}$'),
    Provincia VARCHAR(255) NOT NULL,
    Via VARCHAR(255) NOT NULL,
    NumeroCivico VARCHAR(10) NOT NULL,
    Paese VARCHAR(255) NOT NULL,
    CONSTRAINT fk_cliente_mercato
        FOREIGN KEY (ID_MERCATO) REFERENCES MERCATO(ID_MERCATO)
            ON UPDATE CASCADE
            ON DELETE RESTRICT,
    CONSTRAINT fk_cliente_agente
        FOREIGN KEY (ID_AGENTE) REFERENCES AGENTE(ID_AGENTE)
            ON UPDATE CASCADE
            ON DELETE RESTRICT
);
```

MERCATO

```
CREATE TABLE MERCATO (
    ID_MERCATO INT PRIMARY KEY,
    Nome VARCHAR(255) NOT NULL,
    Tipo VARCHAR(50) NOT NULL CHECK (Tipo IN ('Regione',
'Macrozona'))
);
```

ORDINE

```
CREATE TABLE ORDINE (
    ID_ORDINE INT PRIMARY KEY,
    ID_CLIENTE INT NOT NULL,
    DataOrdine DATE NOT NULL CHECK (DataOrdine <= CURRENT_DATE),
    NumeroArticoli INT NOT NULL,
    PrezzoTotale DECIMAL(10,2) NOT NULL,
    CONSTRAINT fk_ordine_cliente
        FOREIGN KEY (ID_CLIENTE) REFERENCES CLIENTE(ID_CLIENTE)
            ON UPDATE CASCADE
            ON DELETE RESTRICT
);
```

ARTICOLO

```
CREATE TABLE ARTICOLO (
    ID_ARTICOLO INT PRIMARY KEY,
    ID_ORDINE INT NOT NULL,
    ID_LINEA INT NOT NULL,
    CODICE_PARTITA INT NOT NULL,
    NumeroSequenziale INT NOT NULL,
    Costo DECIMAL(10,2) NOT NULL,
    CONSTRAINT fk_articolo_ordine
        FOREIGN KEY (ID_ORDINE) REFERENCES ORDINE(ID_ORDINE)
            ON UPDATE CASCADE
            ON DELETE RESTRICT,
    CONSTRAINT fk_articolo_lineaprodotto
        FOREIGN KEY (ID_LINEA) REFERENCES LINEA_PRODOTTO(ID_LINEA)
            ON UPDATE CASCADE
            ON DELETE RESTRICT,
    CONSTRAINT fk_articolo_partita
        FOREIGN KEY (CODICE_PARTITA) REFERENCES
PARTITA(CODICE_PARTITA)
            ON UPDATE CASCADE
            ON DELETE RESTRICT
);
```

LINEA_PRODOTTO

```
CREATE TABLE LINEA_PRODOTTO (
    ID_LINEA INT PRIMARY KEY,
    Descrizione VARCHAR(255) NOT NULL
);
```

PARTITA

```
CREATE TABLE PARTITA (
    CODICE_PARTITA INT PRIMARY KEY,
    ID_LINEA INT NOT NULL,
    Qualità VARCHAR(50) NOT NULL CHECK (Qualità IN ('Prima Scelta',
'Seconda Scelta', 'Terza Scelta')),
    NumeroArticoli INT NOT NULL,
    Acquisto DECIMAL(10,2) NOT NULL CHECK (Acquisto >= 0),
    Stoccaggio DECIMAL(10,2) NOT NULL CHECK (Stoccaggio >= 0),
    Spedizione DECIMAL(10,2) NOT NULL CHECK (Spedizione >= 0),
    CostoTotale DECIMAL(10, 2) NOT NULL,
    CONSTRAINT fk_partita_lineaprodotto
        FOREIGN KEY (ID_LINEA) REFERENCES LINEA_PRODOTTO(ID_LINEA)
            ON UPDATE CASCADE
            ON DELETE RESTRICT
);
```

FORNISCE

```
CREATE TABLE FORNISCE (
    CODICE_PARTITA INT NOT NULL,
    ID_FORNITORE INT NOT NULL,
    PRIMARY KEY (CODICE_PARTITA, ID_FORNITORE),
    CONSTRAINT fk_fornisce_partita
        FOREIGN KEY (CODICE_PARTITA) REFERENCES
PARTITA(CODICE_PARTITA)
            ON UPDATE CASCADE
            ON DELETE RESTRICT,
    CONSTRAINT fk_fornisce_fornitore
        FOREIGN KEY (ID_FORNITORE) REFERENCES
FORNITORE(ID_FORNITORE)
            ON UPDATE CASCADE
            ON DELETE RESTRICT
);
```

FORNITORE

```
CREATE TABLE FORNITORE (
    ID_FORNITORE INT PRIMARY KEY,
    Nome VARCHAR(255) NOT NULL,
    CAP CHAR(5) NOT NULL CHECK (CAP ~ '^[0-9]{5}$'),
    Indirizzo VARCHAR(255) NOT NULL,
    Provincia VARCHAR(255) NOT NULL,
    Via VARCHAR(255) NOT NULL,
    NumeroCivico VARCHAR(10) NOT NULL,
    Paese VARCHAR(10) NOT NULL
);
```

Trigger per rispettare vincoli di integrità ed evitare anomalie

1. Una partita comprende solo articoli di una determinata linea, non è possibile che ci siano degli articoli che non hanno una linea di appartenenza nella stessa partita.

```
CREATE OR REPLACE FUNCTION check_linea_match()
RETURNS TRIGGER AS $$
DECLARE
  linea_partita INT;
BEGIN
  SELECT p.id_linea
    INTO linea_partita
    FROM partita p
   WHERE p.codicepartita = NEW.codicepartita;

  IF NEW.id_linea != linea_partita THEN
     RAISE EXCEPTION
       'Linea di prodotto non coincidente (Partita: %, Articolo: %)',
       linea_partita, NEW.id_linea;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_linea_match
BEFORE INSERT OR UPDATE
ON ARTICOLO
FOR EACH ROW
EXECUTE FUNCTION check_linea_match();
```

Non è necessario gestire il DELETE, perché cancellando un articolo non dobbiamo più verificarne la coerenza di linea con la PARTITA.

2. Vogliamo che ORDINE.NumeroArticoli corrisponda sempre al numero di record di ARTICOLO collegati a quell'ordine e che ORDINE.PrezzoTotale corrisponda al costo di ciascun articolo.

```
CREATE OR REPLACE FUNCTION trg_update_ordine_from_articolo()
RETURNS TRIGGER AS $$
BEGIN
   IF (TG_OP = 'INSERT') THEN
      IF NEW.ID_ORDINE IS NOT NULL THEN
         UPDATE ORDINE
           SET NumeroArticoli = NumeroArticoli+ 1,
              PrezzoTotale = PrezzoTotale + NEW.Costo
          WHERE ID_ORDINE = NEW.ID_ORDINE;
      END IF;

   ELSIF (TG_OP = 'DELETE') THEN
      IF OLD.ID_ORDINE IS NOT NULL THEN
         UPDATE ORDINE
           SET NumeroArticoli = NumeroArticoli - 1,
              PrezzoTotale = PrezzoTotale - OLD.Costo
          WHERE ID_ORDINE = OLD.ID_ORDINE;
      END IF;

   ELSIF (TG_OP = 'UPDATE') THEN
      IF (NEW.ID_ORDINE != OLD.ID_ORDINE) THEN
         IF OLD.ID_ORDINE IS NOT NULL THEN
            UPDATE ORDINE
              SET NumeroArticoli = NumeroArticoli - 1,
                 PrezzoTotale = PrezzoTotale - OLD.Costo
             WHERE ID_ORDINE = OLD.ID_ORDINE;
         END IF;
         IF NEW.ID_ORDINE IS NOT NULL THEN
            UPDATE ORDINE
              SET NumeroArticoli = NumeroArticoli + 1,
                 PrezzoTotale = PrezzoTotale + NEW.Costo
             WHERE ID_ORDINE = NEW.ID_ORDINE;
         END IF;
      ELSE
         IF NEW.Costo != OLD.Costo AND NEW.ID_ORDINE IS NOT NULL THEN
```

```
            UPDATE ORDINE
              SET PrezzoTotale = PrezzoTotale - OLD.Costo + NEW.Costo
            WHERE ID_ORDINE = NEW.ID_ORDINE;
          END IF;
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_ordine_from_articolo
AFTER INSERT OR UPDATE OR DELETE
ON ARTICOLO
FOR EACH ROW
EXECUTE FUNCTION trg_update_ordine_from_articolo();
```

3. Il campo CostoTotale in PARTITA è sempre dato dalla somma di CostoAcquisto + CostoSpedizione + CostoStoccaggio.

```
CREATE OR REPLACE FUNCTION set_costo_totale_partita()
RETURNS TRIGGER AS $$
BEGIN
    NEW.CostoTotale := NEW.Acquisto + NEW.Stoccaggio + NEW.Spedizione;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_set_costo_totale_partita
BEFORE INSERT OR UPDATE
ON PARTITA
FOR EACH ROW
EXECUTE FUNCTION set_costo_totale_partita();
```

Non serve gestire DELETE perchè cancellando una riga di PARTITA, il CostoTotale cessa di esistere.

4. Costo dell'Articolo: Derivato dividendo il CostoTotale della partita per il NumeroArticoli.

```
CREATE OR REPLACE FUNCTION trg_set_costo_articolo()
RETURNS TRIGGER AS $$
DECLARE
   costo_tot DECIMAL(10,2);
   num_art   INT;
BEGIN
   SELECT CostoTotale, NumeroArticoli
    INTO costo_tot, num_art
    FROM PARTITA
    WHERE CodicePartita = NEW.CodicePartita;

   IF num_art <= 0 THEN
      RAISE EXCEPTION 'Errore: NumeroArticoli in PARTITA = %, non posso calcolare
Costo', num_art;
   END IF;

   NEW.Costo := costo_tot / num_art;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_costo_articolo
BEFORE INSERT OR UPDATE
ON ARTICOLO
FOR EACH ROW
EXECUTE FUNCTION trg_set_costo_articolo();
```

Operazioni Richieste:

QUERY

1. **Media articoli in ordine**: Il numero medio di articoli di un ordine effettuato da clienti di una macrozona
2. **Visualizza Agenti di Clienti min-max**: Tutti gli agenti che seguono solo clienti che hanno effettuato almeno 10 e al massimo 30 ordini
3. **Visualizzare fornitori XYZ**: Il numero di fornitori che forniscono solo partite appartenenti alla linea di prodotto XYZ.
4. Visualizzare la macrozona con il fatturato maggiore
5. Visualizza il numero di ordini dei clienti eseguiti prima del 2024

INSERIMENTI

6. Aggiunge una nuova riga nella tabella CAPO_AREA con ID=100 e Nome=’Capo Area Nord’.
7. Aggiunge un nuovo agente (ID=200), collegato al CAPO_AREA con ID=100.
8. Crea un nuovo MERCATO con ID=300, Nome=’Nord-Est’ e Tipo=’Macrozona’.

AGGIORNAMENTI

9. Cambia il campo Tipo in ‘Regione’ per tutti quei mercati che prima erano ‘Macrozona’ e il cui nome inizia con ‘Nord-’.
10. Per ogni ordine risalente a 2 anni fa, aumenta di 5 unità il numero di articoli e diminuisci del 10% il prezzo totale
11. Incrementa del 15% il campo PrezzoTotale per tutti gli ordini effettuati da clienti che risiedono nella provincia di "Milano".

ELIMINAZIONI

12. Elimina dalla tabella PARTITA tutte le righe di qualità “Terza Scelta” che abbiano anche un costo totale non superiore a 100.
13. Elimina tutti gli agenti che non hanno clienti a loro assegnati (nessuna riga in CLIENTE con ID_AGENTE uguale a quello dell’agente).
14. Elimina tutti i fornitori il cui nome inizia per parola 'Bio'.

Svolgimento delle operazioni

1. Media articoli in ordine: Il numero medio di articoli di un ordine effettuato da clienti di una macrozona

```
SELECT AVG(O.NumeroArticoli) AS MediaArticoli
FROM ORDINE O
WHERE NOT EXISTS (
   SELECT *
   FROM CLIENTE C
   JOIN MERCATO M ON C.MERCATO = M.ID_MERCATO
   WHERE M.Tipo <> 'Macrozona'
     AND C.ID_CLIENTE = O.Cliente
);
```

2. Visualizza Agenti di Clienti min-max: Tutti gli agenti che seguono solo clienti che hanno effettuato almeno 10 e al massimo 30 ordini

```
CREATE VIEW CLIENTE_ORDINI_COUNT AS
SELECT C.ID_CLIENTE, COUNT(O.ID_ORDINE) AS NumeroOrdini
FROM CLIENTE C
LEFT JOIN ORDINE O ON C.ID_CLIENTE = O.Cliente
GROUP BY C.ID_CLIENTE;

SELECT A.*
FROM AGENTE A
WHERE NOT EXISTS (
   SELECT *
   FROM CLIENTE C
   JOIN CLIENTE_ORDINI_COUNT COC ON C.ID_CLIENTE = COC.ID_CLIENTE
   WHERE C.Agente = A.ID_AGENTE
     AND (COC.NumeroOrdini < 10 OR COC.NumeroOrdini > 30)
);
```

3. Visualizzare fornitori XYZ: Il numero di fornitori che forniscono solo partite appartenenti alla linea di prodotto con descrizione “XYZ”.

```
SELECT COUNT(*) AS NumeroFornitori
FROM FORNITORE F
WHERE NOT EXISTS (
   SELECT *
   FROM FORNISCE FO
   JOIN PARTITA P ON FO.PARTITA = P.CODICEPARTITA
   JOIN LINEA_PRODOTTO L ON P.LineaProdotto = L.ID_LINEA
   WHERE FO.FORNITORE = F.ID_FORNITORE
     AND L.Descrizione <> 'XYZ'
);
```

4. Visualizza la macrozona che ha il fatturato maggiore

```
CREATE VIEW FATTURATO_MACROZONA AS
SELECT M.ID_MERCATO,M.Nome,SUM(O.PrezzoTotale) AS FatturatoTotale
FROM MERCATO M
JOIN CLIENTE C ON M.ID_MERCATO = C.MERCATO
JOIN ORDINE O ON C.ID_CLIENTE = O.Cliente
WHERE M.Tipo = 'Macrozona'
GROUP BY M.ID_MERCATO, M.Nome;

SELECT FM.*
FROM FATTURATO_MACROZONA FM
WHERE NOT EXISTS (
   SELECT *
   FROM FATTURATO_MACROZONA FM2
   WHERE FM2.FatturatoTotale > FM.FatturatoTotale
);
```

5. Visualizza il numero di ordini dei clienti eseguiti prima del 2024

```
SELECT C.ID_CLIENTE, C.Nome, COUNT(O.ID_ORDINE) AS NumeroOrdini
FROM CLIENTE C
LEFT JOIN ORDINE O ON C.ID_CLIENTE = O.ID_CLIENTE
WHERE O.DataOrdine < '2024-01-01'
GROUP BY C.ID_CLIENTE, C.Nome
ORDER BY NumeroOrdini DESC;
```

6. Aggiunge una nuova riga nella tabella CAPO_AREA con ID=100 e Nome=’Capo Area Nord’.

```
INSERT INTO CAPO_AREA (ID_CAPOAREA, Nome)
VALUES (100, 'Capo Area Nord');
```

7. Aggiunge un nuovo agente (ID=200), collegato al CAPO_AREA con ID=100.

```
INSERT INTO AGENTE (ID_AGENTE, Nome, ID_CAPOAREA)
VALUES (200, 'Agente Mario Rossi', 100);
```

8. Crea un nuovo MERCATO con ID=300, Nome=’Nord-Est’ e Tipo=’Macrozona’.

```
INSERT INTO MERCATO (ID_MERCATO, Nome, Tipo)
VALUES (300, 'Nord-Est', 'Macrozona');
```

9. Cambia il campo Tipo in ‘Regione’ per tutti quei mercati che prima erano ‘Macrozona’ e il cui nome inizia con ‘Nord-’.

```
UPDATE MERCATO
SET Tipo = 'Regione'
WHERE Tipo = 'Macrozona'
  AND ID_MERCATO IN (
      SELECT ID_MERCATO
      FROM MERCATO
      WHERE Nome LIKE 'Nord-%'
  );
```

10. Per ogni ordine risalente a 2 anni fa, aumenta di 5 unità il numero di articoli e diminuisci del 10% il prezzo totale

```
UPDATE ORDINE
SET NumeroArticoli = NumeroArticoli + 5,
    PrezzoTotale   = PrezzoTotale * 0.9
WHERE DataOrdine < '2023-01-01';
```

11. Incrementa del 15% il campo PrezzoTotale per tutti gli ordini effettuati da clienti che risiedono nella provincia di "Milano".

```
UPDATE ORDINE
SET PrezzoTotale = PrezzoTotale * 1.15
WHERE ID_CLIENTE IN (
    SELECT ID_CLIENTE
    FROM CLIENTE
    WHERE Provincia = 'Milano'
);
```

12. Elimina dalla tabella PARTITA tutte le righe di qualità “Terza Scelta” che abbiano anche un costo totale non superiore a 100.

```
DELETE FROM PARTITA
WHERE Qualità = 'Terza Scelta'
  AND CostoTotale <= 100;
```

13. Elimina tutti gli agenti che non hanno clienti a loro assegnati (nessuna riga in CLIENTE con ID_AGENTE uguale a quello dell’agente).

```
DELETE FROM AGENTE
WHERE NOT EXISTS (
    SELECT *
    FROM CLIENTE C
    WHERE C.ID_AGENTE = AGENTE.ID_AGENTE
);
```

14. Elimina tutti i fornitori il cui nome inizia per parola 'Bio'.

```
DELETE FROM FORNITORE
WHERE ID_FORNITORE IN (
   SELECT ID_FORNITORE
   FROM FORNITORE
   WHERE Nome LIKE 'Bio%'
);
```

Popolamento del DataBase

Per la popolazione del database è stata adottata una strategia in due fasi.
Nella prima fase, è stata utilizzata la libreria Python Faker per generare dati simulati per ciascuna delle entità previste nel modello. I dati generati sono stati esportati in file CSV, uno per ogni tabella, garantendo coerenza e rispettando i vincoli del modello concettuale.
Ad esempio, per l'entità "CAPO AREA":

```
capo_area = []
for i in range(1, 26):
    row = {
        "id_capoarea": i,
        "nome": fake.name()
    }
    capo_area.append(row)

write_csv("dati/capo_area.csv", ["id_capoarea", "nome"], capo_area)
```

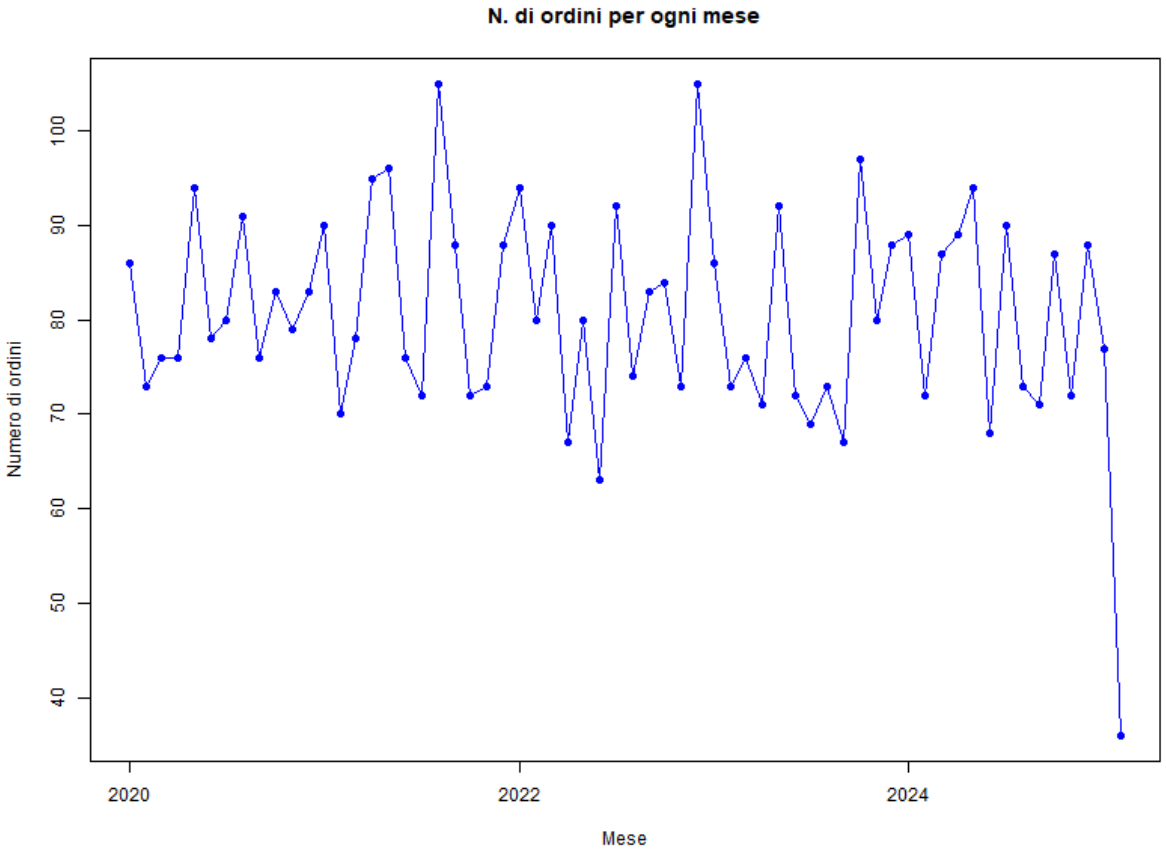
Nella seconda fase, è stato impiegato uno script R per leggere i file CSV e caricarli nel database SQL. Mediante il pacchetto RPostgres e le relative funzioni (come dbWriteTable), i dati sono stati importati nelle tabelle preesistenti. La parte di codice principale che ha permesso il caricamento è la seguente:

```
carica_csv <- function(file_path, table_name) {
  df <- read.csv(file_path, header=TRUE)
  dbWriteTable(con, table_name, df, append=TRUE, row.names=FALSE)
  cat(paste("Dati caricati in", table_name, "da", file_path, "/n"))
}
```

Analisi dei dati in R

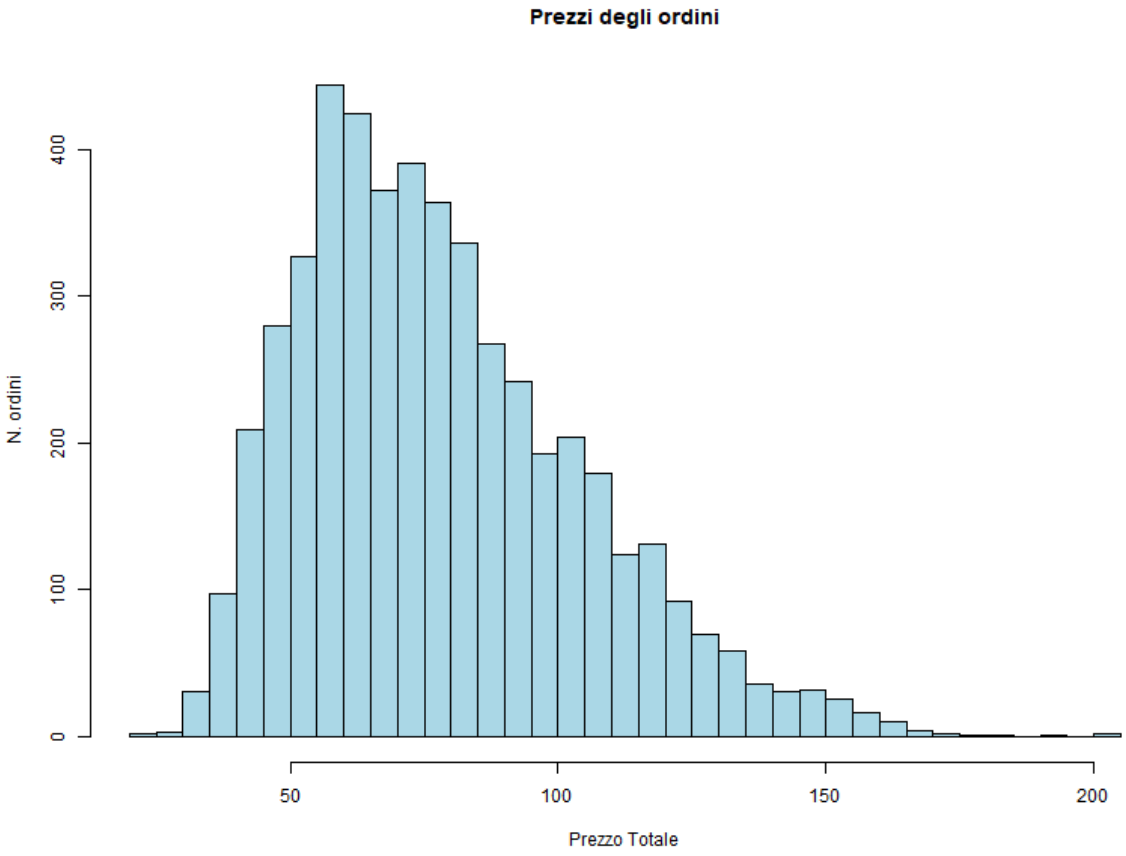
1. Quanti ordini vengono effettuati ogni mese

```
query1 <- "SELECT date_trunc('month', dataordine) AS month, COUNT(*) AS orders FROM
ordine GROUP BY month ORDER BY month;"
ordini_mese <- dbGetQuery(con, query1)
ordini_mese$month <- as.Date(ordini_mese$month)

png("grafici/1_trend_ordini_linee.png", width = 800, height = 600)
plot(ordini_mese$month, ordini_mese$orders, type = "o", col = "blue", pch = 19,
    xlab = "Mese", ylab = "Numero di ordini", main = "N. di ordini per ogni mese")
dev.off()
```



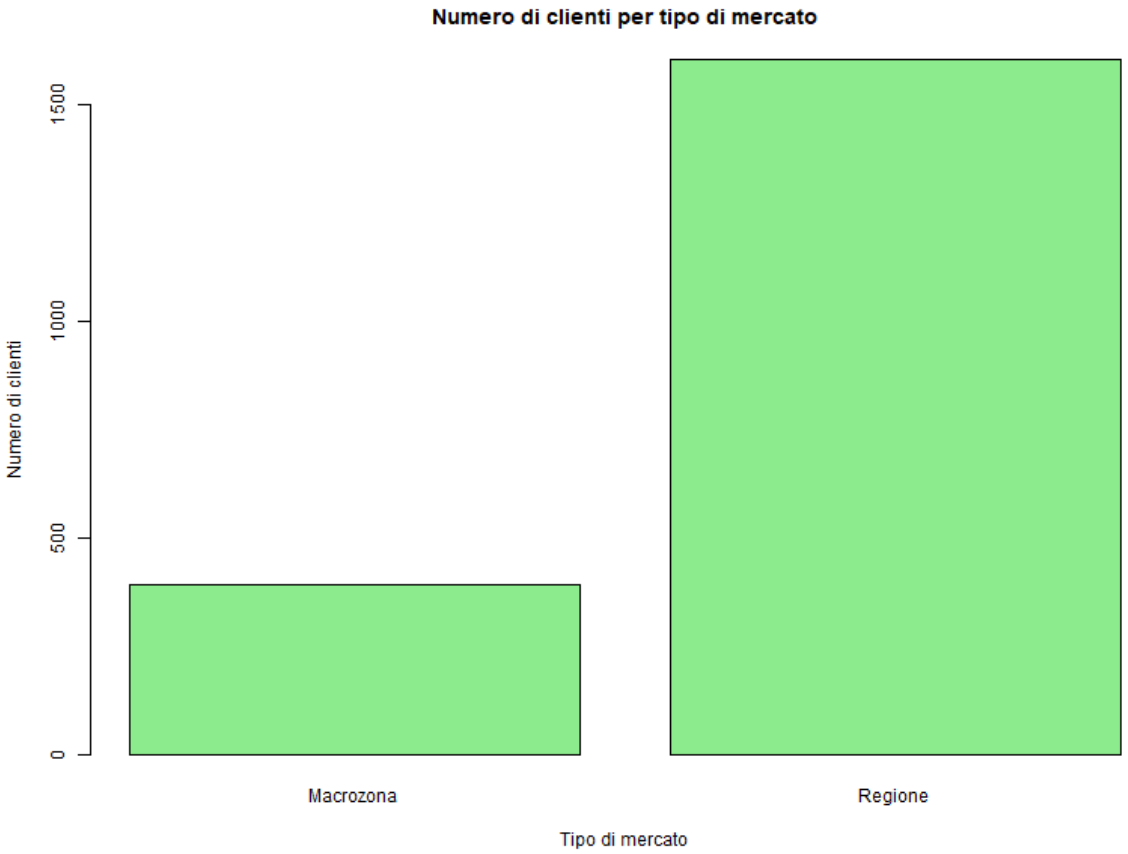
2. Studio delle fasce di prezzo degli ordini

```
query2 <- "SELECT prezzototale FROM ordine;"
ordini_tot <- dbGetQuery(con, query2)

png("grafici/2_prezzototale_istogramma.png", width = 800, height = 600)
hist(ordini_tot$prezzototale, breaks = 30, col = "lightblue", border = "black",
     main = "Prezzi degli ordini", xlab = "Prezzo Totale", ylab = "N. ordini")
dev.off()
```



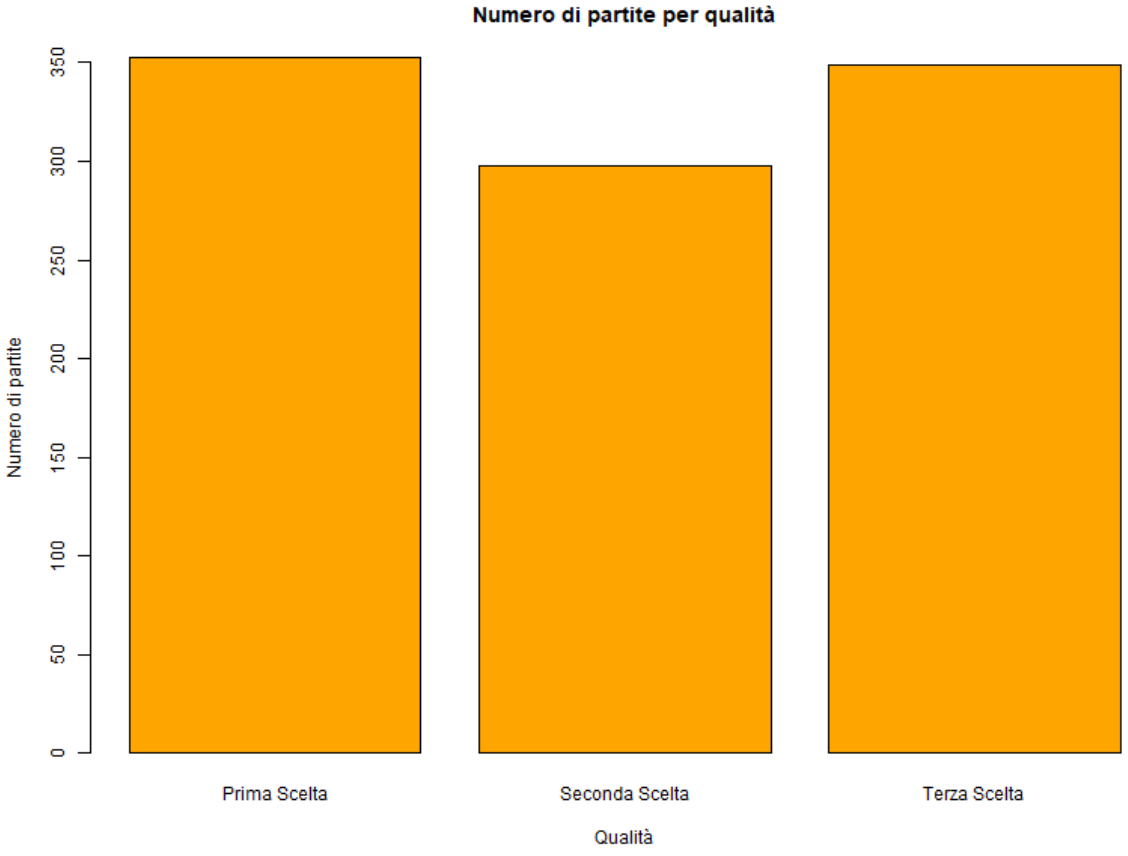
3. Numero di clienti per tipo di mercato

```
query3 <- "SELECT m.tipo, COUNT(*) AS n_clienti FROM cliente c JOIN mercato m ON
c.id_mercato = m.id_mercato GROUP BY m.tipo;"
clienti_mercato <- dbGetQuery(con, query3)

# Converti la colonna in numerico
clienti_mercato$n_clienti <- as.numeric(clienti_mercato$n_clienti)
clienti_mercato <- na.omit(clienti_mercato)

png("grafici/3_clienti_per_mercato.png", width = 800, height = 600)
barplot(clienti_mercato$n_clienti, names.arg = clienti_mercato$tipo, col = "lightgreen",
        main = "Numero di clienti per tipo di mercato", xlab = "Tipo di mercato", ylab = "Numero
di clienti")
dev.off()
```



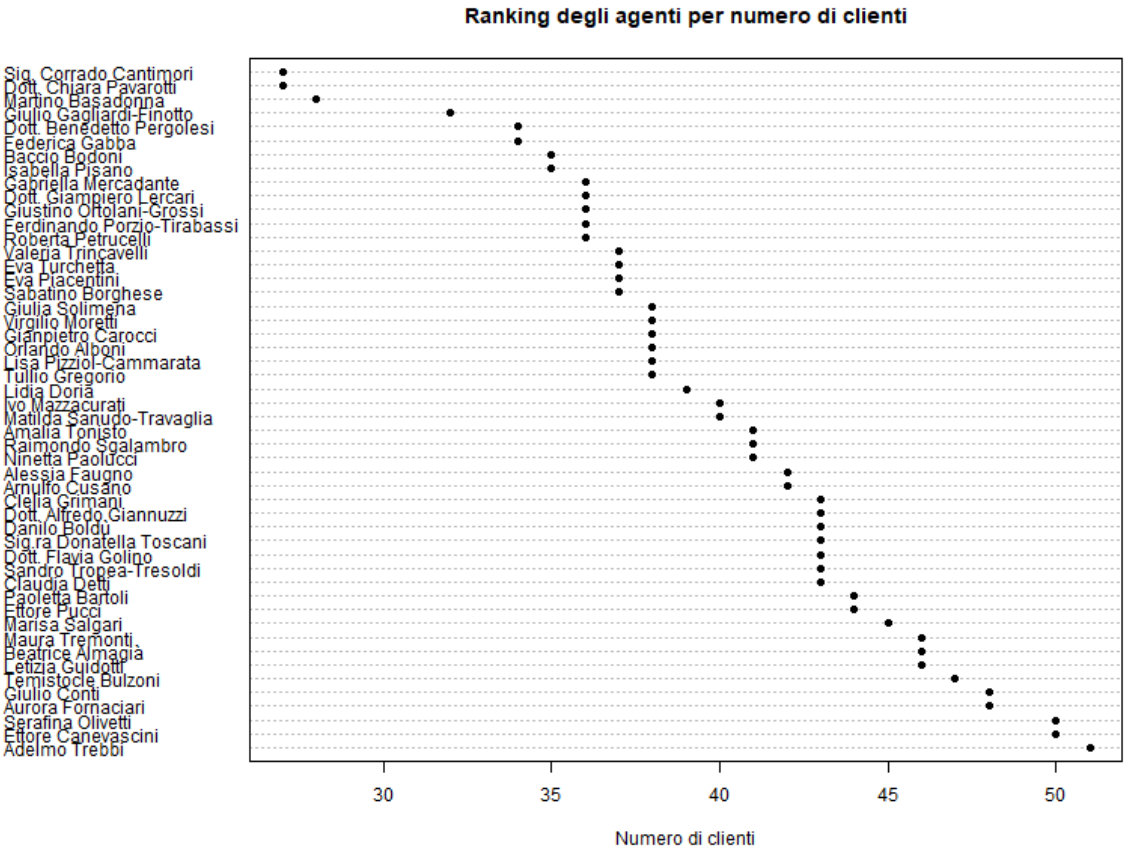
4. Numero di partite per qualità

```
query4 <- "SELECT qualita, COUNT(*) AS num_partite FROM partita GROUP BY qualita;"
partite_qualita <- dbGetQuery(con, query4)

partite_qualita$num_partite <- as.numeric(partite_qualita$num_partite)
partite_qualita <- na.omit(partite_qualita)

png("grafici/4_partite_qualita_bar.png", width = 800, height = 600)
barplot(partite_qualita$num_partite, names.arg = partite_qualita$qualita, col = "orange",
        main = "Numero di partite per qualità", xlab = "Qualità", ylab = "Numero di partite")
dev.off()
```



5. Ranking degli agenti per numero di clienti

```
query5 <- "SELECT a.nome AS agente, COUNT(c.id_cliente) AS num_clienti FROM agente
a JOIN cliente c ON a.id_agente = c.id_agente GROUP BY a.nome ORDER BY num_clienti
DESC;"
agenti_clienti <- dbGetQuery(con, query5)

png("grafici/5_agenti_ranking_dot.png", width = 800, height = 600)
dotchart(agenti_clienti$num_clienti, labels = agenti_clienti$agente, pch = 19,
        main = "Ranking degli agenti per numero di clienti", xlab = "Numero di clienti")
dev.off()
```



Codici e Risorse

È possibile visualizzare e scaricare i file relativi al progetto al seguente link:
https://github.com/AleRoncadin/LAB_BASI_DI_DATI

All'interno del repository sono disponibili:

- Database completo in PostgreSQL (**Lab_DB.sql**)
- Script per il popolamento del database in Python/R (**crea_dati.py** e **carica_dati_su_db.r**)
- Script per l'analisi dei dati in R (**analisi_dati.r**)
- Dati delle tabelle in formato CSV
- Grafici generati dalle analisi