Progetto Basi Di Dati Lego

Introduzione

Lo scopo del progetto è di creare un servizio che permetta ai clienti di svolgere delle indagini su che set Lego possono fare partendo da un quantitativo di set Lego in loro possesso. Inoltre i clienti possono utilizzare un negozio per poter effettuare degli ordini e seguire delle mailinglist.

Requisiti della base di dati

Ogni set Lego è composto da dei pezzi, questi pezzi sono identificati per tipo e colore da un codice (5 numeri per il tipo, 3 per il colore) . Del set Lego si sa qual è il codice che lo rappresenta, l'anno di uscita, a che collezione appartiene (city, technic, family, architecture...) e qual è il suo valore di mercato.

L'utente è definito da un username e può essere un cliente o un admin. Dell'utente sappiamo anche password, nome, cognome ed email. Il cliente può essere iscritto a una, nessuna o più delle mailing list che sono rappresentate da un nome e possono avere una descrizione.

Il cliente può definire un numero di set Lego che possiede e, in base ai set indicati, può chiedere se ha le possibilità di poter svolgere un altro set. Inoltre il cliente può decidere di ordinare dei set lego dal negozio.

Il cliente può vedere lo stato dei suoi ordini e i dettagli.

L'admin può aggiungere, modificare o rendere non disponibile all'acquisto dei set Lego , può aggiungere dei pezzi e può modificare lo sato degli ordini da in lavorazione a spedito e da spedito a consegnato.

Del negozio si tiene conto degli ordini avvenuti. Dell'ordine si sa il suo numero, data in cui è stato effettuato, stato, data in cui è stato spedito e la composizione dell'ordine.

Degli ordini inoltre si tiene presente le informazioni relative ai pagamenti con un numero che identifica il pagamento, la data del pagamento e il quantitativo.

Suddivisione del testo in frasi omogenee

Frasi sui set Lego

"Ogni set Lego è composto da dei pezzi,..."

"Del set Lego si sa qual è il codice che lo rappresenta, l'anno di uscita, a che collezione appartiene (city, technic, family, architecture...) e qual è il suo valore di mercato."

Frasi sui pezzi

"...questi pezzi sono identificati per tipo e colore da un codice."

Frasi sugli utenti

“L’utente è definito da un username e può essere un cliente o un admin. Dell’utente sappiamo anche password, nome, cognome ed email.”

Frasi sui clienti

“ Il cliente può essere iscritto a una, nessuna o più delle mailing list che sono rappresentate da un nome e possono avere una descrizione.”

“Il cliente può definire un numero di set Lego che possiede e, in base ai set indicati, può chiedere se ha le possibilità di poter svolgere un altro set. Inoltre il cliente può decidere di ordinare dei set lego dal negozio.”

“Il cliente può vedere lo stato dei suoi ordini e i dettagli.”

Frasi sugli admin

“L’admin può aggiungere, modificare o rendere non disponibile all’acquisto dei set Lego , può aggiungere dei pezzi e può modificare lo sato degli ordini da in lavorazione a spedito e da spedito a consegnato.”

Frasi sulla mailing list

“...essere iscritto a una o più delle mailing list che sono rappresentate da un nome e possono avere una descrizione.”

Frasi sugli ordini

“...si tiene conto degli ordini avvenuti. Dell’ordine si sa il suo numero, data in cui è stato effettuato, stato, data in cui è stato spedito e la composizione dell’ordine.”

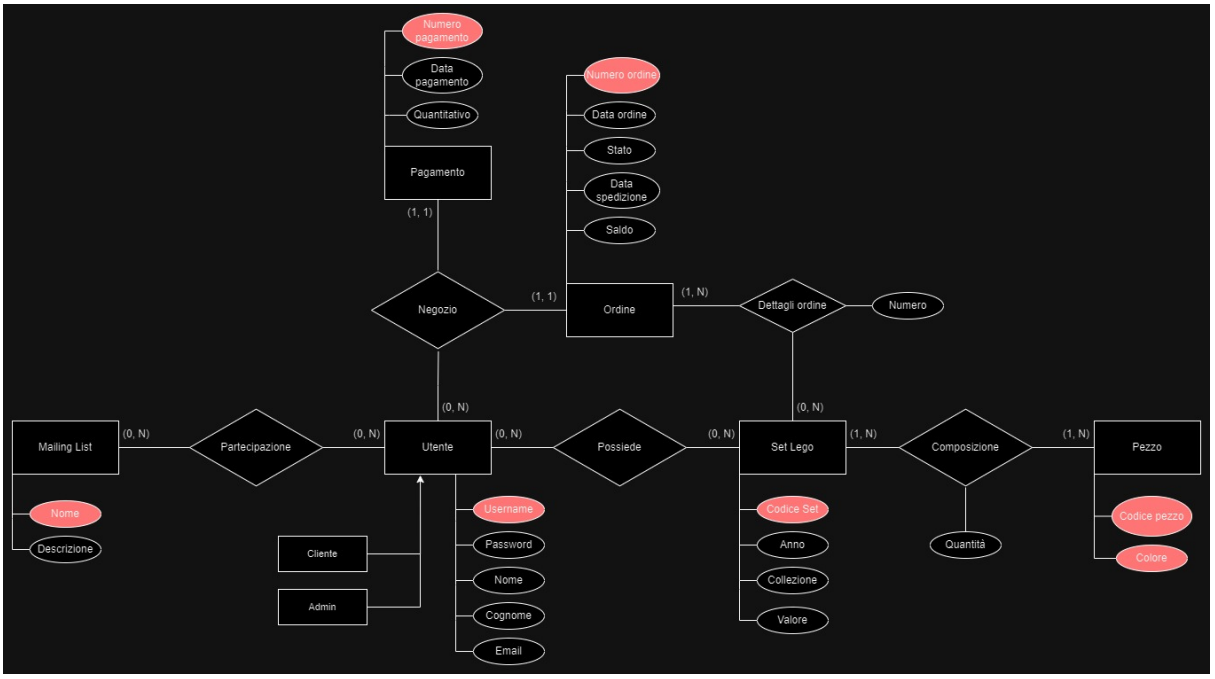
Frasi sui pagamenti

“...le informazioni relative ai pagamenti con un numero che identifica il pagamento, la data del pagamento e il quantitativo.”

Azioni eseguibili

- L’admin vuole poter aggiungere dei set Lego, cambiarne il valore e in caso non renderli disponibili all’acquisto; (20 volte all’anno)
- L’admin vuole poter aggiungere dei pezzi; (10 volte all’anno)
- L’admin vuole poter cambiare lo stato degli ordini; (5 volte al giorno)
- L’admin vuole poter aggiungere traccia di un pagamento effettuato; (5 volte al giorno)
- L’admin vuole poter creare o eliminare una mailinglist; (1 volta ogni anno)
- Il cliente vuole poter aggiungere dei set Lego che ha in possesso e vedere se può creare un set da quelli a lui disponibili; (5 volte al giorno)
- Il cliente vuole poter acquistare dei set Lego dal negozio; (1 volta al mese)
- Il cliente vuole poter vedere quali sono i suoi ordini; (3 volte al mese)
- Il cliente vuole poter iscriversi o disiscriversi da una mailing list; (2 volte ogni anno)

Diagramma Entity-Relationship



Glossario dei termini

Termine	Descrizione	Identificatore
Mailing list	Lista di indirizzi a cui inviare delle mail	Nome
Utente	Entità fisica che utilizza il servizio	Username
Cliente	Account di chi usufruisce del servizio	Username
Admin	Account di chi gestisce il servizio	Username
Set Lego	Collezione di componenti che formano un Lego	Codice set
Pezzo	Componente Lego	Codice pezzo
Pagamento	Informazioni relative a una transazione monetaria	Numero pagamento
Ordine	Oggetto di una transazione	Numero ordine

Vincoli non esprimibili

- Una email non può essere associata a due username diversi;
- Un ordine futuro non può contenere un set Lego non più disponibile;
- Lo stato di un ordine non può essere spedito se non vi è un numero pagamento associato;
- Un ordine non può avere un quantitativo pari a zero di un set;
- Un set non può avere un numero pari a zero di un determinato componente;

Tabella volumi

Concetto	Tipo	Volume
Mailing list	E	4
Utente	E	290
Cliente	E	300
Admin	E	10
Set Lego	E	20000 circa
Pezzo	E	4000 circa
Pagamento	E	200
Ordine	E	200
Partecipazione	R	400
Possiede	R	1000
Composizione	R	30000
Negozio	R	500
Dettagli ordine	R	600

Ristrutturazione

Eliminazione delle generalizzazioni

Si può notare la presenza di una generalizzazione totale ed esclusiva avente come entità padre utente e figlie cliente, admin. Questa è stata eliminata andando ad accorpare i figli al genitore con l'aggiunta di un attributo "Permessi" che può assumere il valore "Cliente" o "Admin" .

Eliminazione relationship n-aria

Si può notare la presenza di una relationship n-aria tra utente, ordine e pagamento. Per non dover creare un altra tabella "Negozio" si è preferito andare a spostare l'entità "Pagamento" e aggiungere nella schema logico una FK "Username".

Valutazione ridondanza

Non vi sono cicli che possano generare ridondanze.

Si può notare la presenza di una ridondanza dovuta da un campo calcolato: "Saldo". Questo campo può essere calcolato sommando il valore dei set ordinati durante l'ordine.

Questo campo è coinvolto soltanto durante una operazione, ossia quando il cliente "vede i suoi ordini".

Considerando la presenza di circa 20000 set Lego e che ogni cliente vada a vedere i suoi ordini 3 volte al mese avremo:

Senza ridondanza

Concetto	Costrutto	Accessi	Tipo
Ordine	Entità	200	L
Dettagli ordine	Relazione	600	L
Set Lego	Entità	20000	L

Con ridondanza

Concetto	Costrutto	Accessi	Tipo
Ordine	Entità	200	L

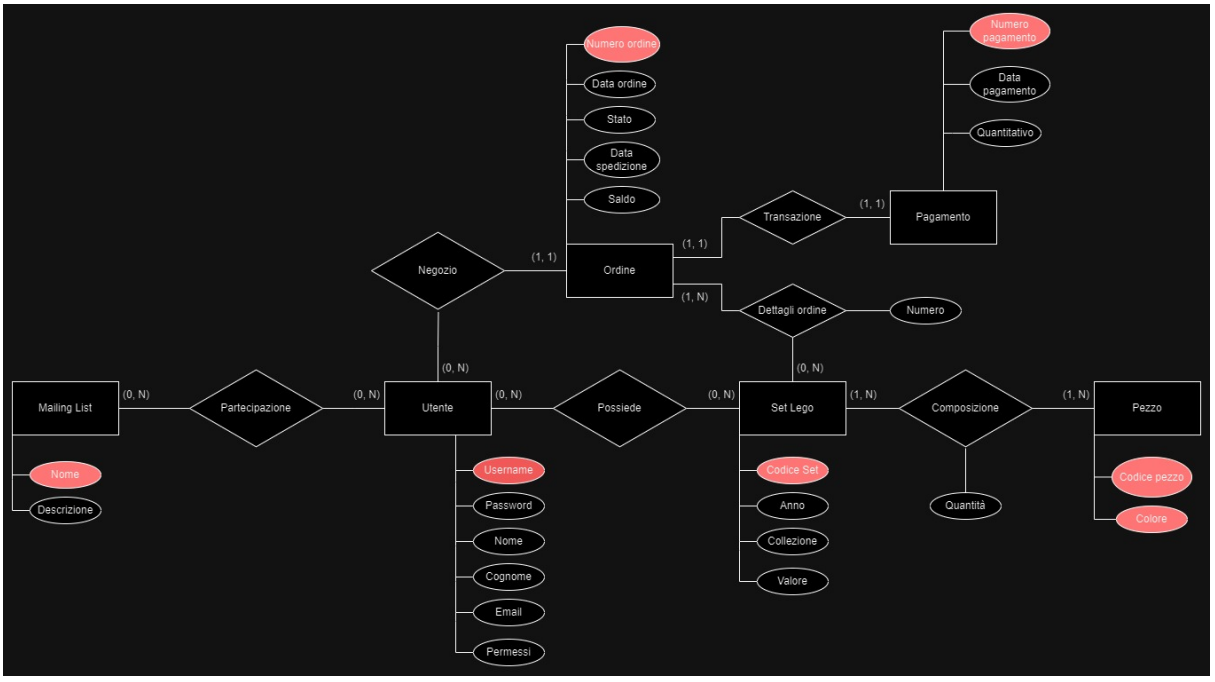
Conclusione:

Con la ridondanza si avrebbero prestazioni più elevate ma questo comporterebbe un appesantimento in fase di scrittura e non verrebbe rispettata la terza forma normale.

Si è deciso di tenere la ridondanza.

"Quantitativo" in pagamenti rappresenta l'importo pagato per un ordine specifico e viene inserito al momento della verifica della transazione da parte di un admin per poter cambiare lo stato dell'ordine corrispettivo, non è un campo calcolato.

Diagramma E-R ristrutturato



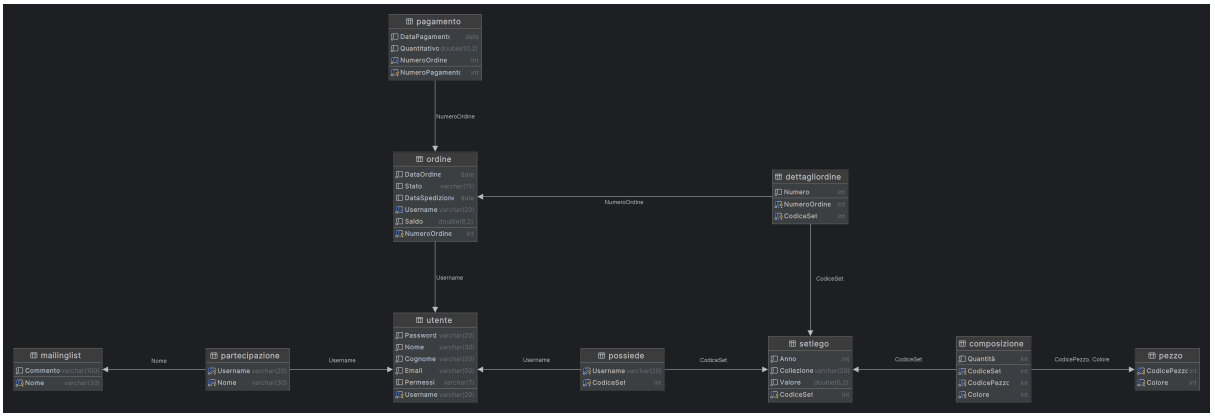
Scelta degli identificatori primari

Per ogni entità è stato scelto un identificatore primario i quali sono indicati dal riempimento della cella.

Gli identificatori primari sono i seguenti:

- Mailing list: “nome” della mailing list;
- Utente: “username” dell’utente;
- Set Lego: il “codice del set” Lego;
- Pezzo: il “codice del pezzo Lego e del suo colore”;
- Ordine: il “numero dell’ordine”;
- Pagamento: il “numero del pagamento”;

Schema logico



sottolineatura = chiave primaria

italizzato = chiave secondaria

- MailingList (Nome, Commento)
- Partecipazione (*Username*, *Nome*)
- Utente (Username, Password, Nome, Cognome, Email, Permessi)
- Possiede (*Username*, *CodiceSet*)
- SetLego (CodiceSet, Anno, Collezione, Valore)
- Composizione (*CodiceSet*, *CodicePezzo*, *CodiceColore*,Quantità)
- Pezzo (CodicePezzo, Colore)
- Ordine (NumeroOrdine, DataOrdine, Stato, DataSpedizione, *Username*)
- Pagamento (NumeroPagamento, DataPagamento, Quantitativo, *NumeroOrdine*)
- DettagliOrdine (*NumeroOrdine*, *CodiceSet*, Numero)

Normalizzazione

Prima forma normale

Tutte le tabelle sono in prima forma normale, tutte le colonne sono atomiche.

Seconda forma normale

Tutte le tabelle sono in seconda forma normale, ciascuna colonna dipende dalla primary key.

Per l'entità "Ordine" vi è un campo calcolato, "Saldo", il quale dipende dalla struttura dell'ordine pertanto dipenderà comunque dalla chiave primaria. Di conseguenza rispetta la seconda forma normale.

Terza forma normale

La tabella "Ordine", avente il campo calcolato "Saldo" non rispetta le condizioni per essere in terza forma normale.

Le restanti tabelle invece rispetta le condizioni per la terza forma normale.

SQL

Trigger

```
/* TRIGGER: due utenti con username diversi non possono avere la stessa email */
DELIMITER $$
CREATE TRIGGER trg_2usernameDifferenteEmail
    BEFORE INSERT ON Utente
    FOR EACH ROW
    BEGIN
        IF EXISTS (SELECT*FROM Utente WHERE Email = NEW.Email) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Questa mail è già usata
            da un altro utente';
        end if;
    END $$
DELIMITER ;


/* TRIGGER: un ordine futuro non può contenre un set lego con valore zero */
DELIMITER $$
CREATE TRIGGER trg_nuovoOrdineSetNonDisponibile
    BEFORE INSERT ON dettagliordine
    FOR EACH ROW
    BEGIN
        IF EXISTS (SELECT*FROM SetLego
        WHERE CodiceSet = NEW.CodiceSet AND Valore = 0) THEN
            SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = "Questo set Lego non
            è disponibile per l'acquisto";
        end if;
    END $$
DELIMITER ;
```

```
/* TRIGGER: non può cambiare lo stato di un ordine da 'in lavorazione' a
'spedito' se non vi è un numero pagamento associato */
DELIMITER $$
CREATE TRIGGER trg_cambioStatoOrdineSenzaVerificaPagamento
    BEFORE UPDATE ON ordine
    FOR EACH ROW
    BEGIN
        IF (NOT EXISTS(SELECT DataPagamento FROM pagamento
        WHERE NumeroOrdine = OLD.NumeroOrdine) && NEW.Stato = 'Spedito') THEN
            SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'Questo ordine non è
            ancora stato pagato';
        end if;
    END $$
DELIMITER ;


/* TRIGGER: un ordine non può avere zero di un set */
DELIMITER $$
CREATE TRIGGER trg_nuovoOrdineQuantitàPerSetZero
    BEFORE INSERT ON dettagliordine
    FOR EACH ROW
    BEGIN
        IF NEW.Numero <= 0 THEN
            SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'Devi acquistare almeno
            un numero pari a 1 di questo set';
        end if;
    END $$
DELIMITER ;


/* TRIGGER: un set non può avere un numero pari a zero di
componenti in componenti */
DELIMITER $$
CREATE TRIGGER trg_nuovoSetQuantitàPerComponenteZero
    BEFORE INSERT ON composizione
    FOR EACH ROW
    BEGIN
        IF NEW.Quantità = 0 THEN
            SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Un set non può avere
            zero di un componente';
        end if;
    END $$
DELIMITER ;
```

Stored procedure

```
/* SP: cliente vuole vedere a che mailing list è iscritto */
DELIMITER $$
CREATE PROCEDURE sp_aCheMailingListSonoIscritto(IN inputUsername VARCHAR(20))
BEGIN
    SELECT Nome FROM partecipazione WHERE Username = inputUsername;
END $$
DELIMITER ;


/* SP: cliente vuole iscriversi a una mailing list */
DELIMITER $$
CREATE PROCEDURE sp_aggiungiInMailingList(IN inputUsername VARCHAR(20),
IN inputNome VARCHAR(30))
BEGIN
    INSERT INTO partecipazione (Username, Nome)
    VALUE (inputUsername, inputNome);
END $$
DELIMITER ;


/* SP: cliente vuole disiscriversi a una mailing list */
DELIMITER $$
CREATE PROCEDURE sp_rimuoviDaMailingList(IN inputUsername VARCHAR(20),
IN inputNome VARCHAR(30))
BEGIN
    DELETE FROM partecipazione
    WHERE Username = inputUsername AND Nome = inputNome;
END $$
DELIMITER ;


/* SP: cliente vuole vedere che set lego ha */
DELIMITER $$
CREATE PROCEDURE sp_cheSetLegoHo(IN inputUsername VARCHAR(20))
BEGIN
    SELECT CodiceSet FROM possiede WHERE Username = inputUsername;
END $$
DELIMITER ;


/* SP: cliente vuole aggiungere un set lego che possiede */
DELIMITER $$
CREATE PROCEDURE sp_aggiungiSetLegoMiei(IN inputUsername VARCHAR(20),
IN inputCodiceSet INT(6))
BEGIN
    INSERT INTO possiede (Username, CodiceSet)
    VALUE (inputUsername, inputCodiceSet);
END $$
DELIMITER ;
```

```
/* SP: cliente vuole rimuovere un set lego tra quelli che ha */
DELIMITER $$
CREATE PROCEDURE sp_rimuoviSetLegoMiei(IN inputUsername VARCHAR(20),
IN inputCodiceSet INT(6))
BEGIN
    DELETE FROM possiede
    WHERE Username = inputUsername AND CodiceSet = inputCodiceSet;
END $$
DELIMITER ;


/* SP: il cliente vuole vedere se può fare un determinato set
da quelli che possiede */
DELIMITER $$
CREATE PROCEDURE sp_verificaPossoFareQuestoSet(IN inputUsername VARCHAR(20),
IN inputCodiceSet INT(6))
BEGIN
    SELECT MIN(somma) FROM(
        SELECT SUM(Quantità) as somma FROM (
            SELECT CodicePezzo, Quantità FROM (
                (SELECT codiceSet FROM possiede
                INNER JOIN setLego USING(codiceSet)
                WHERE Username = inputUsername) AS setCheHa
            INNER JOIN composizione USING(codiceSet))
        UNION ALL
        SELECT CodicePezzo, Quantità*-1 FROM composizione
        WHERE CodiceSet = inputCodiceSet) AS confronto
    GROUP BY CodicePezzo) AS verifica;
END $$
DELIMITER ;


/* SP: cliente vuole creare un ordine */
DELIMITER $$
CREATE PROCEDURE sp_inserimentoOrdineNuovo(IN inputNumeroOrdine INT(8),
IN inputUsername VARCHAR(20),IN inputCodiceSet INT(6), IN inputNumero INT(3))
BEGIN
    START transaction;
       SET @dataOrdine = CURDATE();
       SELECT Valore*inputNumero INTO @salDoPrimoSet FROM setlego
       WHERE CodiceSet = inputCodiceSet;
       INSERT INTO Ordine (NumeroOrdine, DataOrdine, Stato, Username, Saldo)
       VALUES (inputNumeroOrdine,@dataOrdine,'In lavorazione',
       inputUsername, @salDoPrimoSet);
       INSERT INTO dettagliordine (NumeroOrdine, CodiceSet, Numero)
       VALUES (inputNumeroOrdine,inputCodiceSet,inputNumero);
    commit;
END $$
DELIMITER ;
```

```
DELIMITER $$
CREATE PROCEDURE sp_inserimentoOrdineEsistente(IN inputNumeroOrdine INT(8),
IN inputCodiceSet INT(6), IN inputNumero INT(3))
BEGIN
    START transaction;
    SELECT Saldo INTO @saldoAttuale FROM ordine
    WHERE NumeroOrdine = inputNumeroOrdine;
    SELECT Valore INTO @valoreNuovoSet FROM setLego
    WHERE CodiceSet = inputCodiceSet;
    SET @nuovoSaldo = @saldoAttuale + @valoreNuovoSet*inputNumero;
    INSERT INTO dettagliordine (NumeroOrdine, CodiceSet, Numero)
    VALUES (inputNumeroOrdine,inputCodiceSet,inputNumero);
    UPDATE ordine
        SET Saldo =  @nuovoSaldo WHERE NumeroOrdine = inputNumeroOrdine;
    commit;
END $$
DELIMITER ;


/* SP: cliente vuole vedere che ordini ha fatto*/
DELIMITER $$
CREATE PROCEDURE sp_vediMieiOrdini(IN inputUsername VARCHAR(20))
BEGIN
    SELECT ordine.NumeroOrdine, ordine.DataOrdine, ordine.Stato,
    ordine.Saldo, ordine.DataSpedizione FROM ordine
    INNER JOIN dettagliordine USING(NumeroOrdine)
    INNER JOIN setlego USING(CodiceSet)
    WHERE Username = inputUsername
    GROUP BY NumeroOrdine;
END $$
DELIMITER ;


/* SP: cliente vuole vedere i dettagli di un ordine */
DELIMITER $$
CREATE PROCEDURE sp_vediMieiOrdiniNelDettaglio(IN inputNumeroOrdine INT(8))
BEGIN
    SELECT DettagliOrdine.CodiceSet, DettagliOrdine.Numero,
    SetLego.Collezione, SetLego.Anno FROM dettagliordine
    INNER JOIN setlego USING(CodiceSet)
    WHERE NumeroOrdine = inputNumeroOrdine;
END $$
DELIMITER ;
```

```
/* SP: admin vuole poter aggiungere un set lego */
DELIMITER $$
CREATE PROCEDURE sp_aggiungiNuovoSetLego(IN inputCodiceSet INT(6),
IN inputAnno INT(4), IN inputCollezione VARCHAR(20), IN inputValore DOUBLE,
IN inputCodicePezzo INT(8), IN inputColore INT(3), IN inputQuantità INT(3))
BEGIN
    START transaction;
       INSERT INTO setlego (CodiceSet, Anno, Collezione, Valore)
       VALUES (inputCodiceSet,inputAnno,inputCollezione,inputValore);
       INSERT INTO composizione (CodiceSet, CodicePezzo, Colore, Quantità)
       VALUES (inputCodiceSet,inputCodicePezzo,inputColore,inputQuantità);
    commit;
END $$
DELIMITER ;

DELIMITER $$
CREATE PROCEDURE sp_aggiungiComponentiSetLegoEsistente(IN inputCodiceSet INT(6),
IN inputCodicePezzo INT(8), IN inputColore INT(3), IN inputQuantità INT(3))
BEGIN
    INSERT INTO composizione (CodiceSet, CodicePezzo,Colore, Quantità)
    VALUE (inputCodiceSet,inputCodicePezzo,inputColore,inputQuantità);
END $$
DELIMITER ;


/* SP: admin vuole poter aggiungere dei nuovi pezzi */
DELIMITER $$
CREATE PROCEDURE sp_aggiungiNuovoPezzo(IN inputCodicePezzo INT(5),
IN inputColorePezzo INT(3))
BEGIN
    INSERT INTO Pezzo (CodicePezzo, Colore)
    VALUES (inputCodicePezzo,inputColorePezzo);
END $$
DELIMITER ;


/* SP: admin vuole cambiare il valore di un SetLego */
DELIMITER $$
CREATE PROCEDURE sp_valore0SetLego(IN inputCodiceSet INT(6),
IN inputValore DOUBLE)
BEGIN
    UPDATE setlego
        SET Valore = inputValore WHERE CodiceSet = inputCodiceSet;
END $$
DELIMITER ;
```

```
/* SP: admin vuole poter vedere quali ordini non risultano pagati
(pertatno avranno come stato 'In lavorazione') */
DELIMITER $$
CREATE PROCEDURE sp_ordineNonPagato()
BEGIN
    SELECT*FROM ordine WHERE STATO = 'In lavorazione';
END $$
DELIMITER ;


/* SP: admin vuole poter aggiungere un pagamento */
DELIMITER $$
CREATE PROCEDURE sp_aggiungiPagamento(IN inputQuantitativo DOUBLE,
IN inputNumeroOrdine INT(8))
BEGIN
    INSERT INTO Pagamento (DataPagamento, Quantitativo, NumeroOrdine)
    VALUES (CURDATE(),inputQuantitativo,inputNumeroOrdine);
END $$
DELIMITER ;


/* SP: admin vuole poter cambiare lo stato di un ordine da
In lavorazione a Spedito*/
DELIMITER $$
CREATE PROCEDURE sp_cambiaStatoOrdineInSpedito
(IN inputNumeroOrdine INT(8))
BEGIN
    UPDATE ordine
        SET Stato = 'Spedito', DataSpedizione = CURDATE()
        WHERE NumeroOrdine = inputNumeroOrdine;
END $$
DELIMITER ;


/* SP: admin vuole poter cambiare lo stato di un ordine da
Spedito a Consegnato*/
DELIMITER $$
CREATE PROCEDURE sp_cambiaStatoOrdineInConsegnato
(IN inputNumeroOrdine INT(8))
BEGIN
    UPDATE ordine
        SET Stato = 'Consegnato', DataSpedizione = CURDATE()
        WHERE NumeroOrdine = inputNumeroOrdine;
END $$
DELIMITER ;
```

```
/* SP: admin vuole poter creare una nuova mailinglist */
DELIMITER $$
CREATE PROCEDURE sp_creaMailingList(IN inputNome VARCHAR(30),
IN inputCommento VARCHAR(100))
BEGIN
    INSERT INTO mailinglist (Nome, Commento)
    VALUES (inputNome, inputCommento);
END $$
DELIMITER ;


/* SP: admin vuole eliminare una mailinglist e chi vi fa parte */
DELIMITER $$
CREATE PROCEDURE sp_eliminaMailingList(IN inputNome VARCHAR(30))
BEGIN
    START transaction;
        DELETE FROM partecipazione WHERE Nome = inputNome;
        DELETE FROM mailinglist WHERE Nome = inputNome;
    commit;
END $$
DELIMITER ;
```

Demo in java

E’ stata scritta un demo in java la quale usa un database MySQL e il Micro-Orm Norm.

Le azioni eseguibili nella demo sono le seguenti:

- Il cliente può aggiungere un set Lego alla sua lista;
- Il cliente può rimuovere un set Lego dalla sua lista;
- Il cliente può vedere che set Lego sono nella sua lista;
- Il cliente può fare una ricerca se può fare un set Lego;
- Il cliente può fare un ordine Lego;
- Il cliente può vedere lo stato dei suoi ordini;
- Il cliente può unirsi a una delle mailinglist;
- Il cliente può uscire da una delle mailinglist;
- Il cliente può vedere a che mailinglist è iscritto;
- L’admin può aggiungere un set Lego;
- L’admin può aggiungere dei nuovi pezzi Lego;
- L’admin può cambiare il valore di un set Lego;
- L’admin può vedere quali ordini non risultano pagati;
- L’admin può cambiare lo stato di un ordine da “In lavorazione” a “Spedito”;
- L’admin può cambiare lo stato di un ordine da “Spedito” a “Consegnato”;
- L’admin può creare una nuova mailinglist;
- L’admin può eliminare una delle mailinglist;

Il seguente link punta alla repository di GitHUB del progetto svolto:

https://github.com/andrenada/Progetto-Basi-Di-Dati-Nadalutti-Andrea-Pier-Maria