Università degli studi di Napoli Parthenope
SCUOLA DI SCIENZE E TECNOLOGIE
Corso di Basi di Dati

IMPERO GALATTICO

Autori:

Stefano Emanuele Aldanese 0124003003

Donato D'Ambrosio 0124002977

Anno Accademico: 2025/2026

Data di consegna: 10/07/2025

Indice

Elenco delle tabelle

Elenco delle figure

Appendice

0.1 Elenco degli script

- **Seguire le instruzione all'interno dello script `possessoreDB.sql` per creare il possessore del Database**
  - `possessoreDB.sql`: Script per la creazione dell'utente *amministratore* del database.

  Le istruzioni dettagliate per la connessione sono presenti all'interno del file nei commenti.

- **Login come amministratore**: Eseguire il login da terminale con il comando:

  ```
  sqlplus amministratore/admin123@//localhost:[porta_libera]/[il_tuo_PDB]
  ```

  Sostituire `[il_tuo_PDB]` con il nome effettivo del proprio Pluggable Database (es. `XEPDB1`). Maggiori dettagli all'interno di `possessoreDB.sql`.

- **Script da eseguire in ordine obbligatorio:**
  - `creazioneDDL.sql`: Script per la creazione delle tabelle e delle viste del database.
  - `viste.sql`: Contiene le definizioni delle viste del database.
  - `funzioni.sql`: Contiene le funzioni PL/SQL (utilizzate all'interno delle procedure).
  - `procedure.sql`: Contiene le procedure PL/SQL (alcune dipendono dalle viste).

- **Script da eseguire in ordine arbitrario:**
  - `triggers.sql`: Contiene i trigger di controllo e sicurezza.
  - `popolamentoDML.sql`: Script per il popolamento iniziale del database.
  - `users.sql`: Script per la creazione degli utenti applicativi con relativi privilegi.

- **Script da eseguire se si vuole droppare l'intero database**
  - `distruzioneDDL.sql`: Contiene i comandi di `DROP` per tabelle, funzione, procedure, trigger e users!

- **Script da eseguire se si vuole aggiungere sinonimi *pubblici* alle tabelle**
  - `sinonimi.sql`: Contiene i comandi per creare sinonimi pubblici per tutte le tabelle, viste, funzioni e procedure
  - Se **NON** è stato eseguito lo script "sinonimi.sql", per fare le query dal punto di vista degli users mettere il suffisso "amministratore."

    ```
    SELECT * FROM amministratore.Corporation;
    ```

E' stato fornito anche un file README per ulteriori informazioni

1 Progettazione

1.1 Sintesi dei requisiti

Nel Lontanissimo 3000 l'Impero Galattico è **incontrastato** dopo aver sterminato i suoi nemici di maggior potenza militare. Rimangono ad opporsi solo pochi gruppi indipendenti di ribelli sparsi per la galassia.

In questa era di totale controllo, l'**IMPERATORE GALATTICO** ha come maggior ostacolo solo la difficoltà di gestire la terribile logistica del suo impero. Per rendere questo ruolo più facile, ha commissionato un database per tenere traccia di alcuni elementi fondamentali del suo dominio.

In particolare deve tenere traccia dei seguenti elementi:

- I *pianeti* e i *sistemi planetari* nella galassia, controllati o meno.
- Le *guerre di conquista* combattute in tutto l'impero contro ribelli e oppositori su pianeti ancora non occupati o contestati.
- i *gruppi ribelli* che si oppongono al volere imperiale

  Vengono classificati come gruppo ribelle tutte le organizzazioni che tentano di contrastare la conquista di un pianeta.
- Le *divisioni* militari dell'Impero e le relative *flotte*, controllate da generali chiamati *Soth*, in particolare a quali guerre sono assegnate nel tempo.
- Le *corporation*, che permettono all'impero di sostenersi e di espandersi, la loro locazione su più pianeti e le relative merci prodotte.

  Le corporation sono sollecitate ad investire economicamente nelle operazioni militari dell'impero, il controllo di un pianeta viene garantito ad un unica Corporazione alla volta, in base al quantitativo di finanziamenti in fase di conquista.
- Lo spostamento di *materiali* e *armi* in giro per i pianeti.

1.1.1 Glossario

- **Sistema Planetario**: Insieme di pianeti orbitanti attorno a una stella principale
- **Divisione Imperiale**: Unità militare dell'impero con effettivi e specializzazione (fanteria, artiglieria, trasporti, cavalleria)
- **Guerra di Conquista**: Conflitto per il controllo di un pianeta
- **Occupazione**: Controllo di un pianeta da parte di una corporation dopo una guerra
- **Soth**: Soldato cibernetico assegnato alla gestione di una flotta

1.2 Diagramma E/R

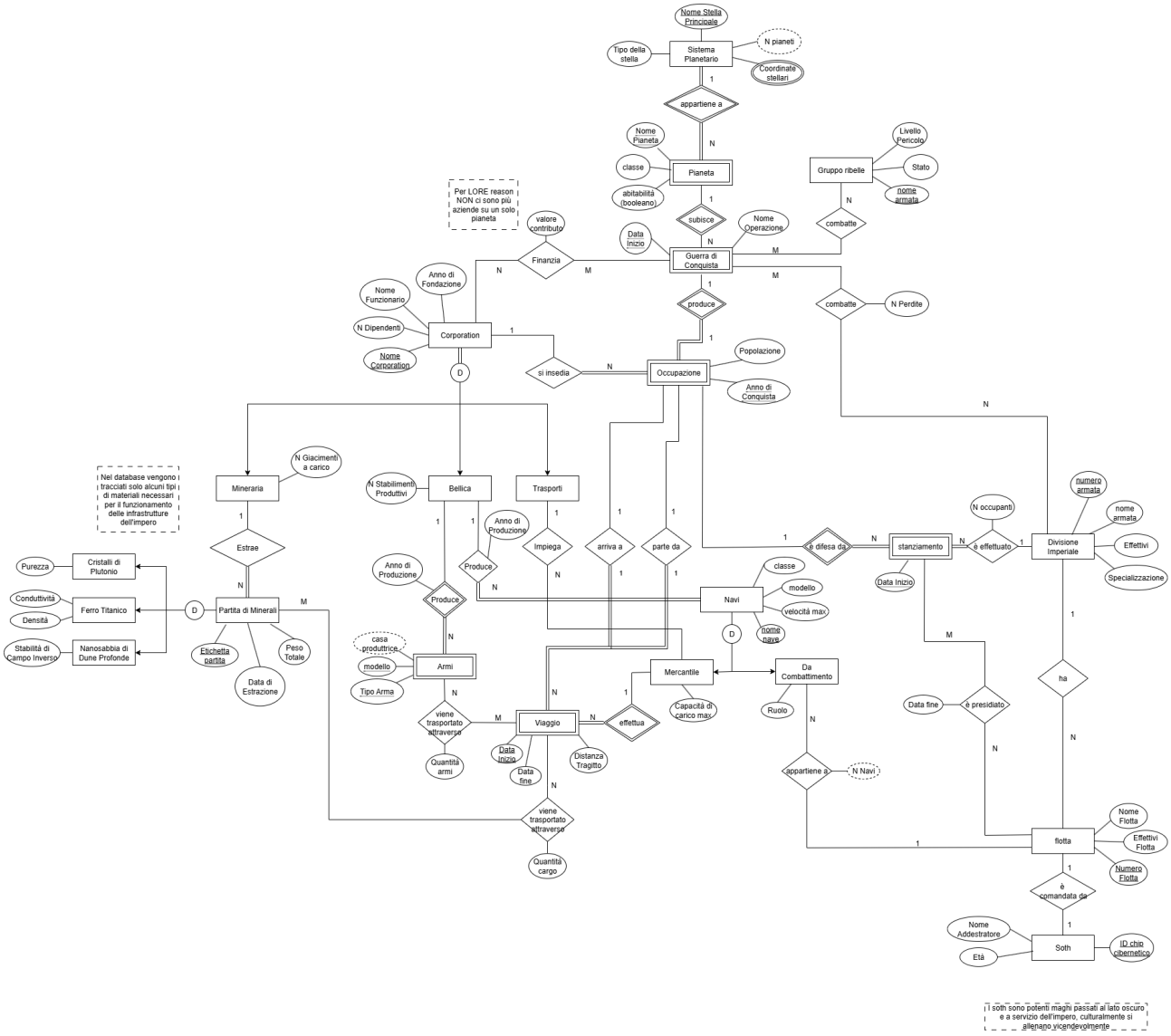

Figura 1: Diagramma Entità-Relazione

1.3 Diagramma relazionale

Figura 2: Diagramma Relazionale

1.4 Utenti e loro categorie

Utente	Tipo	Volume	Permessi
Admin	Amministratore	1	ALL
Imperatore	Comune	1	SELECT ANY EXECUTE ON fine_guerra EXECUTE ON dichiara_guerra EXECUTE ON assegna_divisione
Generale	Comune	700	SELECT ON vista_guerre_attive SELECT ON vista_guerre_finite SELECT, INSERT ON Stazionamento SELECT, INSERT ON e_presidiato SELECT, INSERT ON Soth EXECUTE ON fine_guerra EXECUTE ON assegna_divisione EXECUTE ON rinforza_divisione
Dirigente Corporation	Comune	7000	SELECT, INSERT ON Corporation SELECT, INSERT ON Corporation_Bellica SELECT, INSERT ON Corporation_Mineraria SELECT, INSERT ON Corporation_Trasporti SELECT, INSERT ON Viaggio EXECUTE EXECUTE ON finanzia_guerra

Tabella 1: Tavola degli utenti

1.5 Diagramma UML

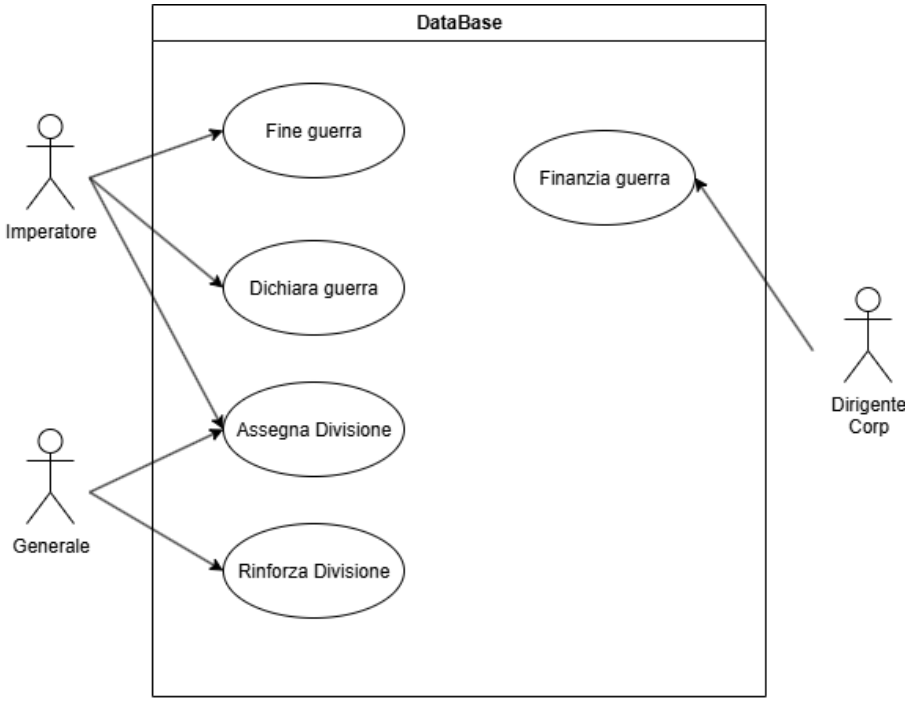

Figura 3: Diagramma UML dei Casi D'uso

1.5.1 Operazioni degli utenti

Seguono le schede descrittore operazione per le operazioni degli utenti:

OPERAZIONE	fine_guerra
SCOPO:	creare un occupazione che sancisce la fine di una guerra
ARGOMENTI:	chiavi della guerra, popolazione del pianeta, perdite totali
RISULTATO:	creazione occupazione, assegnamento di una corporation o errore
ERRORI:	No divisioni in guerra, data di occupazione errata
USA:	occupazione, combatte2, Divisione Imperiale, va_divisionia_coinvolte, guerre_di_conquista
MODIFICA:	occupazione, combatte2
PRIMA:	c'è una guerra attiva
POI:	c'è un occupazione relazionata a una guerra finita

Tabella 2: Operazione fine_guerra

OPERAZIONE	dichiara_guerra
SCOPO:	creare una guerra di conquista
ARGOMENTI:	chiavi del pianeta, nome operazione, data inizio, presenza(bool) di ribelli, livello pericolo di ribelli eventuali
RISULTATO:	creazione guerra, eventuale creazione di nuovo gruppo ribelle
ERRORI:	Guerra già attiva su pianeta, data di inizio guerra errata, il pianeta non è abitabile
USA:	occupazione, combatte2, Divisione_imperiale, guerre_di_conquista
MODIFICA:	guerre_di_conquista, Gruppo_Ribelle, combatte1
PRIMA:	c'è un pianeta non occupato o un occupazione
POI:	c'è una guerra ed un relativo gruppo di ribelli

Tabella 3: Operazione dichiara_guerra

OPERAZIONE	assegna_divisione
SCOPO:	assegna una divisione al combattimento in una guerra
ARGOMENTI:	chiavi della guerra, chiavi della divisione
RISULTATO:	creazione assegnamento in combatte2 ribelle
ERRORI:	la divisione ha troppi pochi effettivi, guerra già conclusa
USA:	vista_guerre_attive, combatte2, Divisione_imperiale, guerre_di_conquista
MODIFICA:	combatte2
PRIMA:	la divisione non è assegnata a una specifica guerra
POI:	la divisione è assegnata a una specifica guerra

Tabella 4: Operazione finanzia_guerra

OPERAZIONE	finanzia_guerra
SCOPO:	Registrare un finanziamento per una guerra da parte di una corporation
ARGOMENTI:	Nome della corporation finanziatrice, chiavi di guerra, Importo del finanziamento
RISULTATO:	Inserimento del finanziamento
ERRORI:	Contributo inferiore ai finanziamenti esistenti(lowballing)
USA:	Tabelle `finanzia`, `guerre_di_conquista`
MODIFICA:	Tabella `finanzia`
PRIMA:	La guerra esiste e può avere o meno finanziamenti precedenti
POI:	La guerra ha un nuovo finanziamento registrato

Tabella 5: Operazione finanzia_guerra

OPERAZIONE	rinforza_divisione
SCOPO:	aggiunge agli effettivi della divisione un numero di soldati
ARGOMENTI:	chiavi della divisione, effettivi da aggiungere
RISULTATO:	numero di effettivi della divisione aumentato
ERRORI:	numero effettivi aggiunti negativo
USA:	Divisione_imperiale
MODIFICA:	Divisione_imperiale
PRIMA:	La divisione ha un numero di effettivi Totali N
POI:	La divisione ha un numero di effettivi Totali N+m con m valore in input

Tabella 6: Operazione rinforza_divisione

OPERAZIONI	TIPO	Volume	PERIODO
Fine guerra	B	50	anno
Dichiara guerra	B	50	anno
Assegna guerra	B	80	anno
Finanzia Guerra	B	400	anno
Rinforza divisione	B	5	anno

Tabella 7: Tabella delle Operazioni

1.5.2 Volumi

Sistema_Planetario	E	10,000	20	Anno
Pianeta	ED	50,000	100	Anno
Guerra_Conquista	ED	5,000	50	Anno
Gruppo_Ribelle	E	1,000	10	Anno
Divisione_Imperiale	E	500	5	Anno
Corporation	E	200	5	Anno
Corporation_Bellica	ES	50	1	Anno
Armi	ED	5,000	100	Anno
Corporation_Trasporti	ES	50	1	Anno
Corporation_Mineraria	ES	100	2	Anno
Partita_di_minerali	ED	1,000,000	2,000	Mese
Ferro_Titanico	ES	500,000	1,000	Mese
Nanosabbia_di_dune_profonde	ES	400,000	800	Mese
Cristalli_di_plutonio	ES	100,000	200	Mese
Navi	E	10,000	200	Anno
Navi_mercantili	ES	6,000	120	Anno
finanzia	A	30,000	600	Anno
Flotta	E	100,000	10	Anno
Navi_da_combattimento	ES	4,000	80	Anno
Occupazione	ED	15,000	300	Anno
Viaggio	ED	300,000	6,000	Settimana
viene_trasportato_attraverso_minerali	A	250,000	5,000	Settimana
viene_trasportato_attraverso_armi	A	50,000	1,000	Settimana
Stazionamento	ED	50,000	1,000	Mese
e_presidiato	A	100,000	2,000	Mese
Soth	E	100,000	20,000	Anno
combatte1	A	7,500	200	Anno
combatte2	A	40,000	600	Anno
vista_effettivi_attuali	V	-	-	-
vista_guerre_attive	V	-	-	-
vista_guerre_finite	V	-	-	-
vista_pianeti_non_assegnati	V	-	-	-

Tabella 8: Tavola dei volumi.

E	Entità
ED	Entità Debole
ES	Entità Specializzata
A	Associazioni (tabelle di transizione)
V	Viste

1.6 Vincoli di integrità

1.6.1 Vincoli statici

I vincoli statici sono quelli definiti direttamente nello schema della base di dati durante la creazione delle tabelle. Nella nostra base di dati sono presenti i seguenti vincoli statici:

- **Pianeta.abitabilita**: può assumere solo valori 0 o 1
- **Gruppo_Ribelle.livello_pericolo**: deve essere un valore compreso tra 1 e 10
- **Gruppo_Ribelle.stato**: può essere solo 'Attivo' o 'Distrutto' (default 'Attivo')
- **Divisione_Imperiale.numero_armata**: deve seguire il formato DIV00-000
- **Divisione_Imperiale.specializzazione**: valori: 'Cavalleria', 'Trasporti', 'Fanteria', 'Artiglieria', 'Fanteria pesante', 'Ricognizione'
- **Flotta.numero_flotta**: deve seguire il formato FLOT00-001
- **Navi_da_combattimento.ruolo**: valori ammessi: 'ricognizione', 'supporto', 'assalto', 'bombardamento'

1.6.2 Vincoli dinamici

I vincoli dinamici sono stati implementati tramite trigger e controlli nelle procedure di inserimento

- **check_perdite_NO_negative**: Impedisce l'inserimento di un numero di perdite superiore agli effettivi disponibili per una divisione
- **check_gioviano_abitabile**: Impedisce l'inserimento di pianeti gioviani abitabili
- **check_capacita_carico_max_armi/minerali**: Impedisce il superamento della capacità di carico massima delle navi mercantili
- **check_guerra_attiva**: Impedisce l'inserimento di una nuova guerra su un pianeta che ha già una guerra attiva
- **check_data_guerra**: Verifica che la data di inizio guerra sia successiva all'ultima occupazione registrata
- **check_data_occupazione**: Verifica che la data di occupazione sia successiva alla data di inizio della guerra corrispondente
- **check_guerra_prima_di_occupazione**: Impedisce l'inserimento di un'occupazione senza una corrispondente guerra
- **check_viaggio_guerra_attiva**: Impedisce viaggi da o verso pianeti con guerre attive
- **check_viaggio_stessa_destinazione**: Impedisce viaggi con stessa partenza e destinazione
- **check_guerra_su_pianeta_non_abitabile**: Impedisce quando si lancia la procedura[1] `dichiara_guerra` di dichiarare guerra su un pianeta disabitato

[1] E' implementato dentro la procedura stessa e NON come trigger!

2 Verifica di normalità

2.1 Prima Forma Normale

Il database rispetta la *prima forma normale* in quanto:

1. **Atomicità degli attributi**: Tutti gli attributi sono atomici e non divisibili. Ad esempio:
   - `nome_pianeta`, `nome_stella_principale` sono stringhe atomiche
   - `data_inizio`, `anno_conquista` sono date considerate atomiche per convenzione
   - I valori numerici come `effettivi`, `numero_perdite` sono singoli
2. **Assenza di gruppi ripetuti**: Non ci sono colonne che contengono valori multipli o composti (es: numero di telefono)
3. **Identificatori univoci**: Ogni tabella ha una chiave primaria ben definita:
   - Chiavi semplici (es: `Sistema_Planetario.nome_stella_principale`)
   - Chiavi composte dove necessario (es: `Pianeta(nome_pianeta, nome_stella_principale)`)

ATTENZIONE: ci sono diverse tabelle che utilizzano campi `DATE` come parte di chiavi primarie o chiavi esterne, il che potrebbe potenzialmente causare anomalie di inserimento

Esempio: Se `anno_conquista` in `Occupazione` è `"2424-02-01"` ma nel `Viaggio` si inserisce `"2424-02-01 00:00:01"`, il riferimento fallirà. → Bisogna assicurarsi che tutti gli utenti che accedono al database usino lo stesso formato di data.

2.2 Seconda Forma Normale

Il database rispetta la *seconda forma normale*:

1. **Tutte le tabelle sono in *prima forma normale***
2. **Nessuna dipendenza parziale**: Per le tabelle con chiavi composte, tutti gli attributi non chiave dipendono dall'intera chiave

2.3 Terza Forma Normale e BCNF

Il database rispetta la *terza forma normale* in quanto:

1. **Tutte le tabelle sono in Seconda Forma Normale**: Come già stabilito, il database rispetta la *seconda forma normale*. Le tabelle con chiavi composte[2] hanno tutti i loro attributi non-chiave che *dipendono* dall'intera chiave primaria composita, senza alcuna dipendenza parziale.
2. **Non ci sono dipendenze transitive**: Ogni attributo non-chiave in ogni tabella dipende direttamente dalla chiave primaria e da nessun altro attributo non-chiave

2.4 Possibili estensioni

- Sistema di tracciamento del percorso del cargo (in questo momento si traccia solo la partenza e la destinazione)
- Politiche sanzionatorie per Corporation che non finanziano abbastanza guerre di conquista

[2] `Pianeta`, `Guerra_Conquista`, `finanzia`, `Viaggio`, `viene_trasportato_attraverso_minerali`, `viene_trasportato_attraverso_armi`, `Stazionamento`, `e_presidiato`, `combatte1`, `combatte2`, `Occupazione`

3 Implementazione

Il codice si riferisce al DBMS Oracle 10g XE e il linguaggio adottato è il PL/SQL.

L'ordine[3] con cui lanciare gli script **NON** è quello con cui sono presentati nel capitolo, che riflette piuttosto l'organizzazione logica dell'argomento.

Tutti i nomi arbitrari sono in minuscolo, mentre le parole riservate in maiuscolo,opportunamente evidenziate in blu.

3.1 Creazione utenti

Il primo passo da compiere è accedere al DBMS come utente `SYSDBA` e creare l'utente amministratore[4] della base di dati. È possibile creare contestualmente anche altri utenti, sebbene essi non possano ancora ricevere alcun privilegio di oggetto, poiché lo schema non è stato ancora definito.

```
1 CREATE USER amministratore IDENTIFIED BY admin123;
2 GRANT ALL PRIVILEGES TO amministratore;
```

Listing 1: Script: possessoreDB.sql

```
1 CREATE USER imperatore IDENTIFIED BY impero4ever;
2 CREATE USER generale IDENTIFIED BY hateyoda;
3 CREATE USER dirigente_corporation IDENTIFIED BY fatturato;
```

Listing 2: Script: users.sql

3.2 Data Definition Language

Il DDL riflette pedissequamente lo schema relazionale: le tabelle sono create mediante altrettante istruzioni `CREATE TABLE` che includono i vincoli di integrità esprimibili nel modello relazionale (l'obbligatorietà, il vincoli di dominio statico, l'univocità, l'integrità referenziale, etc...)

3.2.1 Sistema Planetario

La tabella `Sistema_Planetario` rappresenta l'entità radice del database galattico, contenente i sistemi stellari conosciuti. Ogni sistema è identificato univocamente dal nome della sua stella principale, che funge da chiave primaria. Il campo `tipo_stella` è obbligatorio (NOT NULL) e classifica la stella secondo la classificazione stellare standard (es. 'G' per stelle come il Sole).

Classificazione Stellare Il campo `tipo_stella`, obbligatorio (NOT NULL), segue la classificazione di Harvard-Yerkes:

- **O** (Blu): Stelle ipergiganti
- **B** (Azzurre): Stelle giovani e massicce
- **A** (Bianche): Stelle di sequenza principale
- **F** (Bianco-gialle): Subgiganti
- **G** (Gialle): Stelle simili al Sole
- **K** (Arancioni): Nane arancioni
- **M** (Rosse): Nane rosse

[3] Fare riferimento al `README` o all' `Elenco degli script` presente all'inizio della relazione
[4] Lanciare lo script `possessoreDB.sql`

Le coordinate tridimensionali (`x`, `y`, `z`) sono espresse in anni luce rispetto a un sistema di riferimento galattico centrale. Sebbene queste coordinate siano opzionali, la loro presenza è fondamentale per:

- Calcolare rotte di navigazione interstellari, Determinare distanze tra sistemi, Analizzare la distribuzione spaziale delle corporation

Note Implementative

- La scelta di usare `VARCHAR2` invece di `CHAR` per il nome della stella permette di risparmiare spazio senza sacrificare la flessibilità
- Le coordinate sono memorizzate come `NUMBER` per garantire precisione nei calcoli astrometrici
- L'assenza di vincoli sulle coordinate permette di rappresentare anche sistemi la cui posizione esatta non è ancora stata rilevata

```
CREATE TABLE Sistema_Planetario (
    nome_stella_principale    VARCHAR2(50)    PRIMARY KEY,
    tipo_stella               VARCHAR2(20)    NOT NULL,
    coordinate_stellari_x     NUMBER,
    coordinate_stellari_y     NUMBER,
    coordinate_stellari_z     NUMBER
);
```

Listing 3: Script: creazioneDDL.sql

3.2.2 Pianeti

La tabella `Pianeta` registra tutti i corpi celesti di tipo planetario conosciuti nell'Impero, con le loro caratteristiche fondamentali, dai pianeti(abitabili), partono tutte le attività economiche e produttive dell'impero[5].

Classificazione Planetaria Il campo `classe` utilizza un sistema di classificazione a due livelli:

- **T (Terrestre)**: Pianeti rocciosi con superficie solida
- **G (Gioviani)**: Giganti gassosi o ghiacciati

Abitabilità Il campo `abitabilita` è un booleano (0/1) che indica:

- 1: Mondi in Zona Abitabile con potenziale colonizzabile: Atmosfera respirabile, Presenza di acqua liquida, Temperature compatibili
- 0: Mondi non abitabili:Temperature estreme, Atmosfere tossiche, Assenza di magnetosfera

Vincoli e Relazioni

- Chiave primaria composta da `nome_pianeta` e `nome_stella_principale`
- Vincolo di foreign key che lega ogni pianeta al suo sistema stellare
- Cancellazione a cascata per mantenere l'integrità referenziale

[5]Il progettista del database è fortunatamente riuscito a convincere l'**IMPERATORE GALATTICO** a rinunciare alla produzione di armi di distruzione planetaria, visto e considerato che la scomparsa di uno di loro sarebbe **DISASTROSA** per l'integrità del database

```sql
CREATE TABLE Pianeta (
    nome_pianeta           VARCHAR2(50)    NOT NULL,
    nome_stella_principale VARCHAR2(50)    NOT NULL,
    classe                 VARCHAR2(20),
    abitabilita            NUMBER(1)       CHECK (abitabilita IN (0,1)),
    CONSTRAINT pk_pianeta PRIMARY KEY (nome_pianeta, nome_stella_principale),
    -- Vincolo di chiave esterna verso Sistema_Planetario
    CONSTRAINT fk_pianeta_sistema FOREIGN KEY (nome_stella_principale)
        REFERENCES Sistema_Planetario(nome_stella_principale) ON DELETE CASCADE
);
```

Listing 4: Script: creazioneDDL.sql

3.2.3 Guerra Conquista

La tabella `Guerra_Conquista` registra tutti i conflitti interstellari per il controllo di pianeti nell'Impero Galattico. Ogni guerra è univocamente identificata dalla tripla `(data_inizio, nome_stella_principale, nome_pianeta)`.

Nome Operazione Nome in codice dell'operazione militare (opzionale)

Vincoli e Relazioni

- Chiave primaria composta che garantisce l'unicità di ogni conflitto
- Vincolo di foreign key che lega la guerra al pianeta target
- Cancellazione a cascata per mantenere l'integrità referenziale
- Vincolo di unicità aggiuntivo per supportare le relazioni con `combatte1` e `combatte2`

Note Implementative La data di fine conflitto è derivabile dalla tabella `Occupazione` a cui corrisponde l'attributo `Anno_conquista`

- La logica implementativa è invertita: una guerra si dice conclusa (con data fine) solo quando viene generata un occupazione

Questa tabella costituisce il fulcro del sistema di intelligence militare dell'Impero, integrandosi con i registri delle flotte, delle divisioni e dei gruppi ribelli.

```sql
CREATE TABLE Guerra_Conquista (
    data_inizio           DATE           NOT NULL,
    nome_stella_principale VARCHAR2(50)  NOT NULL,
    nome_pianeta          VARCHAR2(50)   NOT NULL,
    nome_operazione       VARCHAR2(50),
    CONSTRAINT pk_guerra PRIMARY KEY (data_inizio, nome_stella_principale,
    nome_pianeta),
    -- Vincolo di chiave esterna verso Pianeta
    CONSTRAINT fk_guerra_pianeta FOREIGN KEY (nome_pianeta, nome_stella_principale)
        REFERENCES Pianeta(nome_pianeta, nome_stella_principale) ON DELETE CASCADE,
    -- Chiave unica necessaria per FK di combatte1/combatte2
    CONSTRAINT uq_guerra_datainizio UNIQUE (data_inizio, nome_pianeta,
    nome_stella_principale)
);
```

Listing 5: Script: creazioneDDL.sql

3.2.4 Gruppo Ribelle

La tabella `Gruppo_Ribelle` cataloga tutte le organizzazioni ribelli attive contro l'Impero Galattico. Rappresenta una minaccia prioritaria per l'intelligence imperiale.

Classificazione delle Forze Ribelli

- `nome_armata`: Identificativo univoco del gruppo (Primary Key)
  - L'impero usa la convenzione "Ribelli-`nome_stella_principale`",
- `livello_pericolo`: Valutazione strategica (1-10)
  - 1-3: "Oh no, hanno fatto un graffito con 'Viva la Ribellione' su un muro di Rhea." → i ribelli sono disorganizzati o spontanei e la loro presenza non è una minaccia per l'impero.
  - 4-6: "Aspetta... questi stanno parlando tra pianeti???" → Reti interplanetarie, i ribelli iniziano ad organizzarsi contro l'Impero, rappresentano una futura minaccia.
  - 7-9: "Abbiamo perso un incrociatore... e forse anche il controllo della situazione." → Minaccia sistemica: la ribellione inizia a richiedere una strategia specifica di neutralizzazione.
  - 10: "Vader svegliati, abbiamo un problema." → Emergenza galattica: vanno eliminati il prima possibile.
- `stato`: Stato operativo
  - DEFAULT 'Attivo' (presunto se non diversamente specificato)
  - 'Distrutto' dopo interventi militari conclusivi, l'utente generale fara un UPDATE sulla entry per segnalare il cambiamneto dello stato

Protocollo di Valutazione Imperiale L'Impero prima di dichiarare guerra conduce ricognizioni standardizzate per stabilire il livello di pericolo che verrà comunicato all'imperatore all'atto di dichiarazione della guerra

Vincoli Operativi

- CHECK su `livello_pericolo` garantisce valutazioni coerenti
- CHECK su `stato` limita agli stati ufficiali riconosciuti
- DEFAULT su `stato` ottimizza l'inserimento nuovi gruppi

```
CREATE TABLE Gruppo_Ribelle (
    nome_armata         VARCHAR2(50)    PRIMARY KEY,
    livello_pericolo    NUMBER(2) CHECK (livello_pericolo BETWEEN 1 AND 10),
    stato               VARCHAR2(20) DEFAULT 'Attivo' CHECK (stato IN ('Attivo', '
    Distrutto'))
);
```

Listing 6: Script: creazioneDDL.sql

3.2.5 Divisione Imperiale

La tabella `Divisione_Imperiale` registra tutte le unità militari operative dell'Impero Galattico. Ogni divisione rappresenta una forza autonoma con specifiche capacità strategiche.

Struttura Organizzativa

- `numero_armata`: Codice identificativo univoco (Primary Key), NON composto[6]
  - Formato standard: `DIV[settore]-[codice]` (es. `DIV05-101`)
  - Vincolo di formato garantito da espressione regolare

[6] Viene trattato come il codice fiscale nella realtà, quindi viene

- `nome_armata`: Denominazione ufficiale (UNIQUE)
- `effettivi`: Numero di unità militari
  - Range tipico: 150-1500 unità
  - NULL per divisioni in formazione
- `specializzazione`: Tipo di forze (CHECK constraint)
  - **Cavalleria**: Veicoli rapidi e assalti mobili
  - **Trasporti**: Logistica e rifornimenti
  - **Fanteria**: Truppe base
  - **Artiglieria**: Supporto pesante a distanza
  - **Fanteria pesante**: Truppe d'assalto corazzate
  - **Ricognizione**: Esplorazione e intelligence

Protocolli Militari

- Divisioni con stesso prefisso settoriale[7] operano in coordinamento
- Le divisioni con pressisso settoriale 00 iniziale [8] sono considerate d'élite
- La specializzazione determina l'equipaggiamento standard

Note Implementative

- Il vincolo UNIQUE su `nome_armata` previene duplicati
- Il formato del numero armata facilita ricerche per settore
- Viene utilizzata una REGREX per far si che venga immessa il `numero_armata` secondo i protocolli imperiali
- E' stato utilizzato `LOWER` nel `CHECK` della specializzazione per consentire di accettare valori come "ricognizione", "RICOGNIZIONE", ecc...

Questa struttura supporta la gerarchia militare imperiale e l'assegnazione strategica delle risorse nei teatri di guerra.

```
CREATE TABLE Divisione_Imperiale (
    numero_armata       VARCHAR2(20)    NOT NULL,
    nome_armata         VARCHAR2(50)    UNIQUE,
    effettivi           NUMBER,
    specializzazione    VARCHAR2(30),
    CONSTRAINT pk_divisione PRIMARY KEY (numero_armata)
    CONSTRAINT formato_numero_armata CHECK (
        REGEXP_LIKE(numero_armata, '^DIV\d{2}-\d{3}$')
    ),
    CONSTRAINT specializzazione_ammessa CHECK (
        LOWER(specializzazione) IN ('cavalleria', 'trasporti', 'fanteria', '
    artiglieria', 'fanteria pesante', 'ricognizione')
    )
);
```

Listing 7: Script: creazioneDDL.sql

[7] Si intende divisioni come DIV[prefisso1]-[numero1] e DIV[prefisso1]-[numero2]
[8] Si intende divisioni DIV00-[numero]

3.2.6 Corporation

La tabella `Corporation` rappresenta le principali entità commerciali e industriali operanti nell'Impero Galattico. Queste megacorporazioni controllano risorse chiave e settori strategici del pianeta che gli è stato affidato dall'Impero.

Questa struttura supporta il complesso sistema economico imperiale, integrandosi con i registri di produzione, trasporto e attività belliche.

Struttura Organizzativa

- `nome_corporation`: Identificativo univoco (Primary Key)
  - Convenzione di denominazione: Settore + Attività[9]
- `anno_fondazione`: Anno di costituzione (NOT NULL)
  - Formato YYYY (Era Imperiale)
- `nome_funzionario`: Rappresentante e fondatore dell'azienda(da non confondersi col dirigente) (NOT NULL)
- `numero_dipendenti`: Forza lavoro totale
  - Include sia personale organico che contractor
  - NULL per corporation neo-costituite

Classificazione Corporation per settore

- Minerarie
- Belliche
- Logistiche/Trasporti

Note Implementative Le corporation sono collegate alle tabelle specializzate (Minerarie/Belliche/-Trasporti)

Regolamentazione Imperiale

- Ci può essere una sola azienda per pianeta

```sql
CREATE TABLE Corporation (
    nome_corporation   VARCHAR2(50)    PRIMARY KEY,
    anno_fondazione    VARCHAR2(4)     NOT NULL,
    nome_funzionario   VARCHAR2(50)    NOT NULL,
    numero_dipendenti  NUMBER
);
```

Listing 8: Script: creazioneDDL.sql

3.2.7 Corporation Bellica

La tabella `Corporation_Bellica` registra le corporazioni autorizzate alla produzione di armamenti nell'Impero Galattico. Queste entità strategiche riforniscono le forze imperiali di tecnologie militari.

[9] es. "KRAXX Extractors"

Struttura Organizzativa

- `nome_corporation`: Identificativo univoco (Primary Key)
  - Deve corrispondere a una corporation registrata
- `numero_stabilimenti_produttivi`: Capacità industriale
  - Include sia fabbriche che centri R&S
  - Valore NULL indica strutture non dichiarate

Questa tabella è fondamentale per il complesso militare-industriale imperiale, integrandosi con i registri delle divisioni, delle flotte e dei sistemi di armamento.

```
CREATE TABLE Corporation_Bellica (
    nome_corporation               VARCHAR2(50)    PRIMARY KEY,
    numero_stabilimenti_produttivi NUMBER,
    -- Vincolo di chiave esterna verso Corporation
    CONSTRAINT fk_corpbellica_corp FOREIGN KEY (nome_corporation)
        REFERENCES Corporation(nome_corporation) ON DELETE CASCADE
);
```

Listing 9: Script: creazioneDDL.sql

3.2.8 Armi

La tabella `Armi` registra tutti i sistemi d'arma prodotti dalle corporazioni belliche autorizzate nell'Impero Galattico. Ogni arma è identificata da una combinazione univoca di produttore, modello e tipologia.

Struttura Tecnica

- `nome_corporation`: Produttore autorizzato (NOT NULL)
  - Deve essere registrato in `Corporation_Bellica`
- `modello`: modello dell'arma (NOT NULL)
- `tipo_arma`: Categoria tecnologica (laser, plasma, ecc..) (NOT NULL)
- `data_produzione`: Anno di sviluppo (opzionale)
  - Formato standard: YYYY-MM-DD
  - NULL per prototipi non certificati

```
CREATE TABLE Armi (
    nome_corporation   VARCHAR2(50)    NOT NULL,
    modello            VARCHAR2(50)    NOT NULL,
    tipo_arma          VARCHAR2(30)    NOT NULL,
    data_produzione    DATE,
    CONSTRAINT pk_armi PRIMARY KEY (nome_corporation, modello, tipo_arma),
    -- Vincolo di chiave esterna verso Corporation_Bellica
    CONSTRAINT fk_armi_corpbellica FOREIGN KEY (nome_corporation)
        REFERENCES Corporation_Bellica(nome_corporation) ON DELETE CASCADE
);
```

Listing 10: Script: creazioneDDL.sql

3.2.9 Corporation Trasporti

La tabella `Corporation_Trasporti` cataloga le entità autorizzate al trasporto interstellare nell'Impero Galattico. Queste corporazioni costituiscono la spina dorsale della rete logistica imperiale.

Questa tabella è integrata con i registri delle navi mercantili, dei viaggi e dei sistemi planetari, formando il sistema nervoso del commercio interstellare.

Caratteristiche Principali

- `nome_corporation`: Identificativo univoco (Primary Key)
  - Deve corrispondere a una corporation registrata

```
CREATE TABLE Corporation_Trasporti (
    nome_corporation   VARCHAR2(50)    PRIMARY KEY,
    -- Vincolo di chiave esterna verso Corporation
    CONSTRAINT fk_corptrasporti_corp FOREIGN KEY (nome_corporation)
        REFERENCES Corporation(nome_corporation) ON DELETE CASCADE
);
```

Listing 11: Script: creazioneDDL.sql

3.2.10 Corporation Mineraria

La tabella `Corporation Mineraria` registra le corporazioni autorizzate all'estrazione di risorse minerali nell'Impero Galattico. Queste entità controllano i giacimenti strategici che alimentano l'economia imperiale.

Struttura Organizzativa

- `nome_corporation`: Identificativo univoco (Primary Key)
  - Deve corrispondere a una corporation registrata

  text

- `numero_giacimenti_a_carico`: Siti estrattivi attivi
  - Include sia miniere planetarie che asteroidali
  - NULL indica attività estrattive non strutturate

Note Implementative

- La chiave esterna garantisce l'integrità referenziale
- Cancellazione a cascata per mantenere la coerenza dei dati

```
CREATE TABLE Corporation_Mineraria (
    nome_corporation               VARCHAR2(50)    PRIMARY KEY,
    numero_giacimenti_a_carico    NUMBER,
    -- Vincolo di chiave esterna verso Corporation
    CONSTRAINT fk_corpmineraria_corp FOREIGN KEY (nome_corporation)
        REFERENCES Corporation(nome_corporation) ON DELETE CASCADE
);
```

Listing 12: Script: creazioneDDL.sql

3.2.11 Partita di Minerali

La tabella `Partita di minerali` registra ogni spedizione di materiali estratti nell'Impero Galattico. Ogni partita rappresenta un lotto omogeneo di risorse pronte per la lavorazione o il trasporto.

Caratteristiche Tecniche

- `etichetta_partita`: Codice identificativo univoco (Primary Key)
  - Formato: [TIPO][SETTORE]-[NUMERO] (es. STE00-3003)
- `data_estrazione`: Data di produzione (NOT NULL)
- `peso_totale`: Massa in tonnellate imperiali
- `nome_corporation`: Corporazione responsabile (NOT NULL)

Classificazione Minerali

- Ferro Titanico
- Nanosabbia di dune profonde
- Cristalli di plutonio

Note Implementative

- Le tabelle specializzate ereditano la chiave primaria
- La foreign key garantisce la tracciabilità della provenienza

```
CREATE TABLE Partita_di_minerali (
    etichetta_partita  VARCHAR2(10)          NOT NULL,
    data_estrazione    DATE            NOT NULL,
    peso_totale        NUMBER,
    nome_corporation   VARCHAR2(50)    NOT NULL,
    CONSTRAINT pk_partita PRIMARY KEY (etichetta_partita),
    -- Vincolo di chiave esterna verso Corporation_Mineraria
    CONSTRAINT fk_partita_corpmineraria FOREIGN KEY (nome_corporation)
        REFERENCES Corporation_Mineraria(nome_corporation) ON DELETE CASCADE
);
```

Listing 13: Script: creazioneDDL.sql

3.2.12 Ferro Titanico

La tabella `Ferro Titanico` registra le caratteristiche specifiche delle partite di questo materiale strategico, fondamentale per la costruzione di strutture e veicoli militari.

Proprietà Fisiche

- `conduttività`: Misurata in % rispetto allo standard imperiale
- `densità`: Peso specifico in g/cm^3

Utilizzi Principali

- Scafi delle navi da guerra
- Strutture di difesa planetaria
- Componenti per reattori

```sql
CREATE TABLE Ferro_Titanico (
    etichetta_partita  VARCHAR2(10)          PRIMARY KEY,
    conduttivita       NUMBER,
    densita            NUMBER,
    -- Vincolo di chiave esterna verso Partita_di_minerali
    CONSTRAINT fk_ferro_partita FOREIGN KEY (etichetta_partita)
        REFERENCES Partita_di_minerali(etichetta_partita) ON DELETE CASCADE
);
```

Listing 14: Script: creazioneDDL.sql

3.2.13 Nanosabbia di Dune Profonde

La tabella `Nanosabbia di dune profonde` cataloga le proprietà uniche di questo materiale esotico, essenziale per la produzione di scudi energetici e sistemi di occultamento.

stabilita campo inverso Misura della coerenza quantistica (%)

- Valori superiori al 90% indicano qualità militare
- Valori inferiori al 60% sono considerati scarto

```sql
CREATE TABLE Nanosabbia_di_dune_profonde (
    etichetta_partita  VARCHAR2(10)          PRIMARY KEY,
    stabilita_campo_inverso NUMBER,
    -- Vincolo di chiave esterna verso Partita_di_minerali
    CONSTRAINT fk_nanosabbia_partita FOREIGN KEY (etichetta_partita)
        REFERENCES Partita_di_minerali(etichetta_partita) ON DELETE CASCADE
);
```

Listing 15: Script: creazioneDDL.sql

3.2.14 Cristalli di Plutonio

La tabella `Cristalli di plutonio` registra le caratteristiche di questo materiale altamente energetico, utilizzato per i reattori delle navi capitali e delle stazioni spaziali.

Purezza Grado di perfezione cristallina (%)

- Valori superiori al 95% sono richiesti per uso militare
- Valori inferiori al 30% sono considerati pericolosi

```sql
CREATE TABLE Cristalli_di_plutonio (
    etichetta_partita  VARCHAR2(10)          PRIMARY KEY,
    purezza            NUMBER,
    -- Vincolo di chiave esterna verso Partita_di_minerali
    CONSTRAINT fk_cristalli_partita FOREIGN KEY (etichetta_partita)
        REFERENCES Partita_di_minerali(etichetta_partita) ON DELETE CASCADE
);
```

Listing 16: Script: creazioneDDL.sql

3.2.15 Navi

La tabella `Navi` cataloga tutte le unità navali registrate nell'Impero Galattico, sia civili che militari. Costituisce il registro centrale del traffico spaziale imperiale.

Caratteristiche Tecniche

- `nome_nave`: Identificativo univoco (Primary Key)
- `modello`: Designazione del costruttore
- `velocita_max`: Velocità massima raggiungibile dal veicolo.
- `nome_corporation`: produttore, deve essere una corporation bellica registrata

Classificazione Navale in navi da Combattimento e Mercantili

```
CREATE TABLE Navi (
    nome_nave           VARCHAR2(50)    PRIMARY KEY,
    modello             VARCHAR2(50),
    velocita_max        NUMBER,
    nome_corporation    VARCHAR2(50)    NOT NULL,
    -- Vincolo di chiave esterna verso Corporation_Bellica
    CONSTRAINT fk_navi_corpbellica FOREIGN KEY (nome_corporation)
        REFERENCES Corporation_Bellica(nome_corporation) ON DELETE CASCADE
);
```

Listing 17: Script: creazioneDDL.sql

3.2.16 Navi Mercantili

La tabella `Navi_mercantili` registra le unità dedicate al trasporto civile e logistico nell'Impero Galattico. Queste navi costituiscono la flotta commerciale imperiale.

capacita_carico_max numero di unità di cargo che è possibile trasportare

Protocolli Commerciali Devono essere registrate a una corporation di trasporti

```
CREATE TABLE Navi_mercantili (
    nome_nave           VARCHAR2(50)    PRIMARY KEY,
    nome_corporation    VARCHAR2(50)    NOT NULL,
    capacita_carico_max NUMBER,
    -- Vincolo di chiave esterna verso Navi
    CONSTRAINT fk_navimerc_navi FOREIGN KEY (nome_nave)
        REFERENCES Navi(nome_nave) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Corporation_Trasporti
    CONSTRAINT fk_navimerc_corptrans FOREIGN KEY (nome_corporation)
        REFERENCES Corporation_Trasporti(nome_corporation) ON DELETE CASCADE
);
```

Listing 18: Script: creazioneDDL.sql

3.2.17 Finanzia

La tabella di transizione `finanzia` documenta i contributi economici delle corporation alle guerre di conquista imperiali. Rappresenta il legame tra interessi economici e espansione militare.

Parametri Finanziari `valore_contributo`: Importo in crediti imperiali

Regolamentazione

- Ogni corporation può finanziare più conflitti
- I contributi sono considerati donazioni per dimostrare la lealtà della Corporation all'Impero

```sql
CREATE TABLE finanzia (
    nome_corporation    VARCHAR2(50)    NOT NULL,
    data_inizio         DATE            NOT NULL,
    nome_stella_principale VARCHAR2(50) NOT NULL,
    nome_pianeta        VARCHAR2(50)    NOT NULL,
    valore_contributo   NUMBER,
    CONSTRAINT pk_finanzia PRIMARY KEY (nome_corporation, data_inizio,
    nome_stella_principale, nome_pianeta),
    -- Vincolo di chiave esterna verso Corporation
    CONSTRAINT fk_finanzia_corp FOREIGN KEY (nome_corporation)
        REFERENCES Corporation(nome_corporation) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Guerra_Conquista
    CONSTRAINT fk_finanzia_guerra FOREIGN KEY (data_inizio, nome_stella_principale,
    nome_pianeta)
        REFERENCES Guerra_Conquista(data_inizio, nome_stella_principale, nome_pianeta
    ) ON DELETE CASCADE
);
```

Listing 19: Script: creazioneDDL.sql

3.2.18 Flotta

La tabella `Flotta` organizza le forze navali imperiali in gruppi tattici. Ogni flotta rappresenta una formazione operativa autonoma.

Struttura Organizzativa

- `numero_flotta`: Codice identificativo (Primary Key)
  - Formato[10]: FLOT[settore]-[numero]
- `nome_flotta`: Designazione ufficiale (NOT NULL)
- `effettivi_flotta`: Numero di navi assegnate
- `numero_armata`: Divisione di riferimento (NOT NULL)

Gerarchia Militare Le flotte rispondono alla divisione imperiale di afferenza

```sql
CREATE TABLE Flotta (
    numero_flotta       VARCHAR2(20)    PRIMARY KEY,
    nome_flotta         VARCHAR2(50)    NOT NULL,
    effettivi_flotta    NUMBER,
    numero_armata       VARCHAR2(20)    NOT NULL,
    -- Vincolo di chiave esterna verso Divisione_Imperiale
    CONSTRAINT fk_flotta_divisione FOREIGN KEY (numero_armata)
        REFERENCES Divisione_Imperiale(numero_armata) ON DELETE CASCADE,
    -- Vincolo CHECK sul formato del numero_flotta
    CONSTRAINT formato_numero_flotta CHECK (
        REGEXP_LIKE(numero_flotta, '^FLOT\d{2}-\d{3}$')
    )
);
```

Listing 20: Script: creazioneDDL.sql

3.2.19 Navi da Combattimento

La tabella `Navi_da_combattimento` cataloga le unità militari operative nelle flotte imperiali. Queste navi costituiscono la forza di proiezione dell'Impero.

[10] es. FLOT02-003

Ruoli Tattici

- `ruolo`: Specializzazione operativa
  - Ricognizione: Sensori e esplorazione
  - Supporto: Logistica e riparazioni
  - Assalto: Combattimento diretto
  - Bombardamento: Attacco a distanza

Assegnazione Strategica Ogni nave deve essere assegnata a una flotta

Note Implementative

- E' stato utilizzato `LOWER` nel `CHECK` del ruolo per consentire di accettare valori come "ricognizione", "RICOGNIZIONE", ecc...
- Viene utilizzata una REGREX per far si che venga immessa il `numero_flotta` secondo i protocolli imperiali

```
CREATE TABLE Navi_da_combattimento (
    nome_nave           VARCHAR2(50)    PRIMARY KEY,
    numero_flotta       VARCHAR2(20)    NOT NULL,
    ruolo               VARCHAR2(30),

    -- Vincolo di chiave esterna verso Navi
    CONSTRAINT fk_navicomb_navi FOREIGN KEY (nome_nave)
        REFERENCES Navi(nome_nave) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Flotta (solo numero_flotta)
    CONSTRAINT fk_navicomb_flotta FOREIGN KEY (numero_flotta)
        REFERENCES Flotta(numero_flotta) ON DELETE CASCADE,
    -- Vincolo CHECK sul formato del numero_flotta (es. FLOT00-001)
    CONSTRAINT formato_numero_flotta_navi_comb CHECK (
        REGEXP_LIKE(numero_flotta, '^FLOT\d{2}-\d{3}$')
    ),
    CONSTRAINT check_ruolo CHECK (
        LOWER(ruolo) IN ('ricognizione', 'supporto', 'assalto', 'bombardamento')
    )
);
```

Listing 21: Script: creazioneDDL.sql

3.2.20 Occupazione

La tabella `Occupazione` documenta i pianeti conquistati e amministrati dall'Impero Galattico. Rappresenta l'estensione territoriale imperiale.

popolazione : Censimento in milioni di abitanti

Gestione Planetaria `nome_corporation`: Ente amministratore designato, responsabile dello sfruttamento delle risorse.

Note Storiche

- L'`anno_conquista` corrisponde alla data di fine della guerra di conquista associata
- La tripla chiave esterna lega alla guerra corrispondente

```sql
CREATE TABLE Occupazione (
    anno_conquista       DATE            NOT NULL,
    nome_stella_principale  VARCHAR2(50)    NOT NULL,
    nome_pianeta            VARCHAR2(50)    NOT NULL,
    data_inizio             DATE            NOT NULL,
    popolazione             NUMBER,
    nome_corporation        VARCHAR2(50)    NOT NULL,
    CONSTRAINT pk_occupazione PRIMARY KEY (anno_conquista, nome_stella_principale,
    nome_pianeta, data_inizio),
    -- Vincolo di chiave esterna verso Guerra_Conquista
    CONSTRAINT fk_occupazione_guerra FOREIGN KEY (data_inizio, nome_stella_principale
    , nome_pianeta)
        REFERENCES Guerra_Conquista(data_inizio, nome_stella_principale, nome_pianeta
    ) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Corporation
    CONSTRAINT fk_occupazione_corporation FOREIGN KEY (nome_corporation)
        REFERENCES Corporation(nome_corporation) ON DELETE CASCADE
);
```

Listing 22: Script: creazioneDDL.sql

3.2.21 Viaggio

La tabella `Viaggio` registra tutte le rotte commerciali autorizzate nell'Impero Galattico. Documenta il movimento di merci tra pianeti occupati fatte dalle singole navi.

Parametri di Navigazione

- `distanza_tragitto`: distanza misurata in anni luce
- `data_inizio_viaggio`: Partenza effettiva
- `data_fine_viaggio`: Arrivo registrato
  - NULL indica viaggio in corso o fallito

Attributi pianeta La tabella viaggio contiene chiavi esterne che puntano al pianeta di partenza e di arrivo

- `anno_conquista`
- `nome_stella`
- `nome_pianeta`
  - NULL indica viaggio in corso o fallito

Sicurezza

- Ogni viaggio deve essere effettuato da nave mercantile
- I punti di partenza/destinazione devono essere pianeti occupati
- I punti di partenza/destinazione **NON** possono essere uguali, l'Impero non traccia i movimenti intrapanetari

```sql
CREATE TABLE Viaggio (
    data_inizio_viaggio DATE NOT NULL,
    data_fine_viaggio DATE,
    distanza_tragitto NUMBER,
    nome_nave VARCHAR2(50) NOT NULL,
    -- Campi partenza
```

```
    anno_conquista_partenza DATE NOT NULL,
    nome_stella_principale_partenza VARCHAR2(50) NOT NULL,
    nome_pianeta_partenza VARCHAR2(50) NOT NULL,
    data_inizio_GC_partenza DATE NOT NULL,
    -- Campi destinazione
    nome_pianeta_destinazione VARCHAR2(50) NOT NULL,
    nome_stella_principale_destinazione VARCHAR2(50) NOT NULL,
    anno_conquista_destinazione DATE NOT NULL,
    data_inizio_GC_destinazione DATE NOT NULL,
    CONSTRAINT pk_viaggio PRIMARY KEY (data_inizio_viaggio, nome_nave),
    CONSTRAINT fk_viaggio_navimerc FOREIGN KEY (nome_nave)
        REFERENCES Navi_mercantili(nome_nave) ON DELETE CASCADE,
    CONSTRAINT fk_viaggio_occupazione_partenza FOREIGN KEY (anno_conquista_partenza,
    nome_stella_principale_partenza, nome_pianeta_partenza, data_inizio_GC_partenza)
        REFERENCES Occupazione(anno_conquista, nome_stella_principale, nome_pianeta,
    data_inizio) ON DELETE CASCADE,
    CONSTRAINT fk_viaggio_occupazione_destinazione FOREIGN KEY (
    anno_conquista_destinazione, nome_stella_principale_destinazione,
    nome_pianeta_destinazione, data_inizio_GC_destinazione)
        REFERENCES Occupazione(anno_conquista, nome_stella_principale, nome_pianeta,
    data_inizio) ON DELETE CASCADE
);
```

Listing 23: Script: creazioneDDL.sql

3.2.22 Trasporto Minerali

La tabella di transizione `viene_trasportato_attraverso_minerali` documenta il movimento di risorse estratte attraverso l'Impero. Garantisce la tracciabilità delle materie prime strategiche.

quantita_cargo unità di cargo trasportate

```
CREATE TABLE viene_trasportato_attraverso_minerali (
    etichetta_partita      VARCHAR2(10)    NOT NULL,
    data_inizio_viaggio    DATE            NOT NULL,
    nome_nave              VARCHAR2(50)    NOT NULL,
    quantita_cargo         NUMBER          NOT NULL,
    CONSTRAINT pk_trasporto PRIMARY KEY (etichetta_partita, data_inizio_viaggio,
    nome_nave),
    -- Vincolo di chiave esterna verso Partita_di_minerali
    CONSTRAINT fk_trasporto_partita FOREIGN KEY (etichetta_partita)
        REFERENCES Partita_di_minerali(etichetta_partita) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Viaggio
    CONSTRAINT fk_trasporto_viaggio FOREIGN KEY (data_inizio_viaggio, nome_nave)
        REFERENCES Viaggio(data_inizio_viaggio, nome_nave) ON DELETE CASCADE
);
```

Listing 24: Script: creazioneDDL.sql

3.2.23 Trasporto Armi

La tabella di transizione `viene_trasportato_attraverso_armi` registra i movimenti di armamenti tra i sistemi imperiali. Soggetta a rigorosi protocolli di sicurezza.

quantita_cargo_armi unità di cargo trasportate

```
CREATE TABLE viene_trasportato_attraverso_armi (
    nome_corporation        VARCHAR2(50)    NOT NULL,
    modello                 VARCHAR2(50)    NOT NULL,
    tipo_arma               VARCHAR2(30)    NOT NULL,
    data_inizio_viaggio     DATE            NOT NULL,
    nome_nave               VARCHAR2(50)    NOT NULL,
    quantita_cargo_armi     NUMBER          NOT NULL,
```

```
    CONSTRAINT pk_trasporto_armi PRIMARY KEY (nome_corporation, modello, tipo_arma,
    data_inizio_viaggio, nome_nave),
    -- Vincolo di chiave esterna verso Armi
    CONSTRAINT fk_trasporto_armi FOREIGN KEY (nome_corporation, modello, tipo_arma)
        REFERENCES Armi(nome_corporation, modello, tipo_arma) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Viaggio
    CONSTRAINT fk_trasporto_armi_viaggio FOREIGN KEY (data_inizio_viaggio, nome_nave)
        REFERENCES Viaggio(data_inizio_viaggio, nome_nave) ON DELETE CASCADE
);
```

Listing 25: Script: creazioneDDL.sql

3.2.24 Stazionamento

La tabella `Stazionamento` documenta la presenza militare imperiale sui pianeti occupati. Rappresenta l'apparato di controllo territoriale.

numero_occupanti Effettivi dispiegati

Note Strategiche Ogni stazionamento è legato a una specifica occupazione

```
CREATE TABLE Stazionamento (
    data_inizio_stazionamento DATE            NOT NULL,
    numero_armata             VARCHAR2(20)    NOT NULL,
    anno_conquista            DATE            NOT NULL,
    nome_stella_principale    VARCHAR2(50)    NOT NULL,
    nome_pianeta              VARCHAR2(50)    NOT NULL,
    data_inizio               DATE            NOT NULL,
    numero_occupanti          NUMBER,
    CONSTRAINT pk_stazionamento PRIMARY KEY (data_inizio_stazionamento, numero_armata
    , anno_conquista, nome_stella_principale, nome_pianeta, data_inizio),
    -- Vincolo di chiave esterna verso Divisione_Imperiale
    CONSTRAINT fk_stazionamento_divisione FOREIGN KEY (numero_armata)
        REFERENCES Divisione_Imperiale(numero_armata) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Occupazione
    CONSTRAINT fk_stazionamento_occupazione FOREIGN KEY (anno_conquista,
    nome_stella_principale, nome_pianeta, data_inizio)
        REFERENCES Occupazione(anno_conquista, nome_stella_principale, nome_pianeta,
    data_inizio) ON DELETE CASCADE
);
```

Listing 26: Script: creazioneDDL.sql

3.2.25 Presidio

La tabella di transizione `e_presidiato` coordina la difesa integrata dei territori imperiali, unendo forze terrestri e spaziali.

Ciclo Operativo

- `data_fine_stazionamento`: Conclusione missione
  - NULL indica presidio attivo

```
CREATE TABLE e_presidiato (
    data_inizio_stazionamento DATE            NOT NULL,
    numero_armata             VARCHAR2(20)    NOT NULL,
    anno_conquista            DATE            NOT NULL,
    nome_stella_principale    VARCHAR2(50)    NOT NULL,
    nome_pianeta              VARCHAR2(50)    NOT NULL,
    data_inizio               DATE            NOT NULL,
    numero_flotta             VARCHAR2(20)    NOT NULL,
```

```
    data_fine_stazionamento   DATE,
    CONSTRAINT pk_presidiato PRIMARY KEY (data_inizio_stazionamento, numero_armata,
    anno_conquista, nome_stella_principale, nome_pianeta, data_inizio, numero_flotta)
    ,
    -- Vincolo di chiave esterna verso Stazionamento
    CONSTRAINT fk_presidiato_stazionamento FOREIGN KEY (data_inizio_stazionamento,
    numero_armata, anno_conquista, nome_stella_principale, nome_pianeta, data_inizio)
        REFERENCES Stazionamento(data_inizio_stazionamento, numero_armata,
    anno_conquista, nome_stella_principale, nome_pianeta, data_inizio) ON DELETE
    CASCADE,
    -- Vincolo di chiave esterna verso Flotta
    CONSTRAINT fk_presidiato_flotta FOREIGN KEY (numero_flotta)
        REFERENCES Flotta(numero_flotta) ON DELETE CASCADE
);
```

Listing 27: Script: creazioneDDL.sql

3.2.26 Soth

La tabella `Soth` registra gli ufficiali cibernetici dell'Impero, elementi chiave nella gestione delle flotte spaziali.

Caratteristiche Individuali

- `ID_chip_cibernetico`: Identificativo univoco (Primary Key)
- `data_conversione`: Data in cui vengono impiegati a livello effettivo come Soth, il più grande degli onori militari.

Assegnazione Operativa Ogni Soth è assegnato a una flotta

```
CREATE TABLE Soth (
    ID_chip_cibernetico NUMBER          PRIMARY KEY,
    nome                VARCHAR2(50),
    data_conversione    DATE,
    numero_flotta       VARCHAR2(20)     NOT NULL,
    -- Vincolo di chiave esterna verso Flotta
    CONSTRAINT fk_soth_flotta FOREIGN KEY (numero_flotta)
        REFERENCES Flotta(numero_flotta) ON DELETE CASCADE
);
```

Listing 28: Script: creazioneDDL.sql

3.2.27 Combattimento Ribelle

La tabella di transizione `combatte1` documenta la presenza di gruppi ribelli come ostili nelle varie guerre di conquista.

```
CREATE TABLE combatte1 (
    nome_armata_ribelle    VARCHAR2(50)    NOT NULL,
    data_inizio            DATE            NOT NULL,
    nome_stella_principale VARCHAR2(50)    NOT NULL,
    nome_pianeta           VARCHAR2(50)    NOT NULL,
    CONSTRAINT pk_combatte1 PRIMARY KEY (nome_armata_ribelle, data_inizio,
    nome_stella_principale, nome_pianeta),
    -- Vincolo di chiave esterna verso Gruppo_Ribelle
    CONSTRAINT fk_combatte1_ribelle FOREIGN KEY (nome_armata_ribelle)
        REFERENCES Gruppo_Ribelle(nome_armata) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Guerra_Conquista
    CONSTRAINT fk_combatte1_guerra FOREIGN KEY (data_inizio, nome_stella_principale,
    nome_pianeta)
        REFERENCES Guerra_Conquista(data_inizio, nome_stella_principale, nome_pianeta
    ) ON DELETE CASCADE
```

```
);
```

Listing 29: Script: creazioneDDL.sql

3.2.28 Combattimento Imperiale

La tabella di transizione `combatte2` registra l'impiego delle forze imperiali nelle guerre di conquista. Documenta lo sforzo militare dell'Impero.

Parametri Operativi `numero_perdite`: Vittime imperiali

Note Strategiche

- Una divisione può essere impiegata in un solo conflitto per volta
- Le perdite includono sia morti che dispersi

```
CREATE TABLE combatte2 (
    numero_armata          VARCHAR2(20)    NOT NULL,
    data_inizio            DATE            NOT NULL,
    nome_stella_principale VARCHAR2(50)    NOT NULL,
    nome_pianeta           VARCHAR2(50)    NOT NULL,
    numero_perdite         NUMBER,
    CONSTRAINT pk_combatte2 PRIMARY KEY (numero_armata, data_inizio,
    nome_stella_principale, nome_pianeta),
    -- Vincolo di chiave esterna verso Divisione_Imperiale
    CONSTRAINT fk_combatte2_divisione FOREIGN KEY (numero_armata)
        REFERENCES Divisione_Imperiale(numero_armata) ON DELETE CASCADE,
    -- Vincolo di chiave esterna verso Guerra_Conquista
    CONSTRAINT fk_combatte2_guerra FOREIGN KEY (data_inizio, nome_stella_principale,
    nome_pianeta)
        REFERENCES Guerra_Conquista(data_inizio, nome_stella_principale, nome_pianeta
    ) ON DELETE CASCADE
);
```

Listing 30: Script: creazioneDDL.sql

3.3 Data Manipulation Language

Per motivi di semplicità, essendo il Database a scopo dimostrativo la maggior parte delle tuple vengono inserite tramite insert nonostante sarebbe banaale creare procedure apposite e garantire gli accessi ai relativi utenti. È il caso di: pianeti, Corporation, Viaggio, Partita di minerali, armi etc.

Le Entità fondamentali, che essere inserite solo tramite procedure sono:

- Guerra_di_conquista
- occupazione
- Finanziamento
- Combatte2

Nonostante nel caso reale queste entità andrebbero popolate dagli utenti in base alle necessità, abbiamo comunque aggiunto un popolamento iniziale tramite insert per semplificare la fase di testing.

Di seguito alcuni esempi di Insert di tabelle

3.3.1 Pianeta

```
INSERT INTO Pianeta VALUES ('Terra',     'Helios', 'Terrestre', 1);
INSERT INTO Pianeta VALUES ('Marte',     'Helios', 'Gioviani',  0);
INSERT INTO Pianeta VALUES ('Nova',      'Alpha',  'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Venus',     'Helios', 'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Jupiter',   'Helios', 'Gioviani',  0);
INSERT INTO Pianeta VALUES ('Mercury',   'Helios', 'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Saturn',    'Helios', 'Gioviani',  0);
INSERT INTO Pianeta VALUES ('Uranus',    'Helios', 'Gioviani',  0);
INSERT INTO Pianeta VALUES ('Neptune',   'Helios', 'Gioviani',  0);
INSERT INTO Pianeta VALUES ('Pluto',     'Helios', 'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Ares',      'Alpha',  'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Gaia',      'Alpha',  'Terrestre', 1);
INSERT INTO Pianeta VALUES ('Chronos',   'Beta',   'Gioviani',  0);
INSERT INTO Pianeta VALUES ('Rhea',      'Beta',   'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Hyperion',  'Gamma',  'Gioviani',  0);
INSERT INTO Pianeta VALUES ('Selene',    'Gamma',  'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Ares',      'Epsilon', 'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Gaia',      'Zeta',    'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Venus',     'Eta',     'Terrestre', 0);
INSERT INTO Pianeta VALUES ('Jupiter',   'Theta',   'Gioviani',  0);
INSERT INTO Pianeta VALUES ('Mercury',   'Iota',    'Terrestre', 0);
```

Listing 31: Script: popolamentoDML.sql

3.3.2 Corporation

```
INSERT INTO Corporation VALUES ('KRAXX Extractors',    '2100', 'Joe Biden',
    500);
INSERT INTO Corporation VALUES ('TARKON Armaments',    '2090', 'Jessika Caldouron',
    800);
INSERT INTO Corporation VALUES ('VORST Dynamics',      '2110', 'Alan Smithee',
    300);
INSERT INTO Corporation VALUES ('SEKIGUCHI Armaments', '2053', 'Miura Oda',
    600);
INSERT INTO Corporation VALUES ('ASTRA Mining',        '2105', 'Lara Croft',
    700);
INSERT INTO Corporation VALUES ('NEBULA Tech',         '2115', 'Rick Sanchez',
    900);
INSERT INTO Corporation VALUES ('ORION Logistics',     '2125', 'Morty Smith',
    400);
INSERT INTO Corporation VALUES ('GALAXY Arms',         '2130', 'Sarah Connor',
    850);
INSERT INTO Corporation VALUES ('COSMOS Energy',       '2140', 'John Connor',
    950);
INSERT INTO Corporation VALUES ('SOLARIS Ventures',    '2150', 'Ellen Ripley',
    600);
INSERT INTO Corporation VALUES ('QUANTUM Dynamics',    '2160', 'Peter Parker',
    750);
INSERT INTO Corporation VALUES ('INFINITY Corp',       '2170', 'Tony Spark',
    1000);
INSERT INTO Corporation VALUES ('STELLAR Industries',  '2180', 'Bruce Wyn',
    650);
INSERT INTO Corporation VALUES ('LUNAR Holdings',      '2190', 'Clark Kento',
    550);
INSERT INTO Corporation VALUES ('PULSAR Systems',      '2200', 'Diana Prince',
    800);
```

Listing 32: Script: popolamentoDML.sql

4 Trigger

4.1 Controllo Perdite Negative

Questo trigger impedisce l'inserimento[11] di un numero di perdite superiore agli effettivi disponibili per una divisione militare, evitando così valori negativi. Utilizza una vista ottimizzata per ottenere gli effettivi attuali.

```
CREATE OR REPLACE TRIGGER check_perdite_NO_negative
BEFORE INSERT ON combatte2
FOR EACH ROW
DECLARE
    v_effettivi_disponibili NUMBER;
BEGIN
    -- Ottiene gli effettivi disponibili direttamente dalla vista
    SELECT Effettivi_Attuali INTO v_effettivi_disponibili
    FROM vista_effettivi_attuali
    WHERE numero_armata = :NEW.numero_armata;

    -- Verifica se le perdite superano gli effettivi disponibili
    IF :NEW.numero_perdite > v_effettivi_disponibili THEN
        RAISE_APPLICATION_ERROR(-20001,
            'Errore: Perdite (' || :NEW.numero_perdite || ') superiori agli effettivi
    disponibili (' ||
            v_effettivi_disponibili || ') per la divisione ' || :NEW.numero_armata);
    END IF;

```

[11] Non viene gestito l'update ne viene consentito di farlo agli utenti per problemai di Moving Table: cioè il trigger entra in race condition dato che ci tenta di fare l'update sulla tabella che sta controllando

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20002,
            'Divisione ' || :NEW.numero_armata || ' non trovata nei registri imperiali');
END;
```

Listing 33: triggers.sql

4.2 Controllo Pianeti Gioviani Abitabili

Impedisce l'inserimento o modifica di pianeti gioviani marcati come abitabili, in quanto fisicamente impossibile.

```
CREATE OR REPLACE TRIGGER check_gioviano_abitabile
BEFORE INSERT OR UPDATE ON Pianeta
FOR EACH ROW
BEGIN
    IF :NEW.classe = 'Gioviani' AND :NEW.abitabilita = 1 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Errore: I pianeti gioviani non possono essere abitabili, fisicamente non puoi viviere sul gas!');
    END IF;
END;
```

Listing 34: triggers.sql

4.3 Controllo Capacità di Carico Navi Mercantili

Questi due trigger lavorano in tandem per verificare che il carico totale (armi + minerali) non superi la capacità massima della nave mercantile. Oracle non supporta trigger multi-tabella, quindi sono stati implementati due trigger separati che si controllano a vicenda.

```
CREATE OR REPLACE TRIGGER check_capacita_carico_max_armi
BEFORE INSERT ON viene_trasportato_attraverso_armi
FOR EACH ROW
DECLARE
    v_capacita_max NUMBER;
    v_carico_minerali NUMBER;
    v_carico_armi NUMBER;
    v_carico_totale NUMBER;
BEGIN
    SELECT nm.capacita_carico_max INTO v_capacita_max
    FROM Navi_mercantili nm
    WHERE nm.nome_nave = :NEW.nome_nave;

    SELECT NVL(SUM(vm.quantita_cargo), 0) INTO v_carico_minerali
    FROM viene_trasportato_attraverso_minerali vm
    WHERE vm.data_inizio_viaggio = :NEW.data_inizio_viaggio
      AND vm.nome_nave = :NEW.nome_nave;

    SELECT NVL(SUM(va.quantita_cargo_armi), 0) INTO v_carico_armi
    FROM viene_trasportato_attraverso_armi va
    WHERE va.data_inizio_viaggio = :NEW.data_inizio_viaggio
      AND va.nome_nave = :NEW.nome_nave;

    -- Includi il nuovo carico in inserimento
    v_carico_totale := v_carico_minerali + v_carico_armi + :NEW.quantita_cargo_armi;

    IF v_carico_totale > v_capacita_max THEN
        RAISE_APPLICATION_ERROR(-20004,
            'Errore: Carico totale (' || v_carico_totale || ') supera capacita' nave (' || v_capacita_max || ')');
    END IF;
END;
```

Listing 35: triggers.sql

```sql
CREATE OR REPLACE TRIGGER check_capacita_carico_max_minerali
BEFORE INSERT ON viene_trasportato_attraverso_minerali
FOR EACH ROW
DECLARE
    v_capacita_max NUMBER;
    v_carico_minerali NUMBER;
    v_carico_armi NUMBER;
    v_carico_totale NUMBER;
BEGIN
    SELECT nm.capacita_carico_max INTO v_capacita_max
    FROM Navi_mercantili nm
    WHERE nm.nome_nave = :NEW.nome_nave;

    SELECT NVL(SUM(vm.quantita_cargo), 0) INTO v_carico_minerali
    FROM viene_trasportato_attraverso_minerali vm
    WHERE vm.data_inizio_viaggio = :NEW.data_inizio_viaggio
      AND vm.nome_nave = :NEW.nome_nave;

    SELECT NVL(SUM(va.quantita_cargo_armi), 0) INTO v_carico_armi
    FROM viene_trasportato_attraverso_armi va
    WHERE va.data_inizio_viaggio = :NEW.data_inizio_viaggio
      AND va.nome_nave = :NEW.nome_nave;

    -- Includi il nuovo carico in inserimento
    v_carico_totale := v_carico_minerali + v_carico_armi + :NEW.quantita_cargo;

    IF v_carico_totale > v_capacita_max THEN
        RAISE_APPLICATION_ERROR(-20004,
            'Errore: Carico totale (' || v_carico_totale || ') supera capacita' nave
    (' || v_capacita_max || ')');
    END IF;
END;
```

Listing 36: triggers.sql

4.4 Controllo Guerra Attiva

Impedisce l'inserimento di una nuova guerra su un pianeta che ha già una guerra attiva (senza data di fine).

```sql
CREATE OR REPLACE TRIGGER check_guerra_attiva
BEFORE INSERT ON Guerra_Conquista
FOR EACH ROW
DECLARE
    guerra_attiva NUMBER;
BEGIN
    -- Controlla se esiste gia' una guerra attiva per questo pianeta (contiamo le
    guerre che non hanno ancora una data_fine/Non hanno l'occupazione associata)
    SELECT COUNT(*) INTO guerra_attiva
    FROM Guerra_Conquista gc
    LEFT JOIN Occupazione o ON gc.data_inizio = o.data_inizio
                           AND gc.nome_stella_principale = o.nome_stella_principale
                           AND gc.nome_pianeta = o.nome_pianeta
    WHERE gc.nome_stella_principale = :NEW.nome_stella_principale
    AND gc.nome_pianeta = :NEW.nome_pianeta
    AND o.anno_conquista IS NULL;

    IF guerra_attiva > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Esiste gia' una guerra attiva per il pianeta
     ' || :NEW.nome_pianeta || ' nel sistema ' || :NEW.nome_stella_principale);
    END IF;
END;
```

Listing 37: triggers.sql

4.5 Controllo Data Guerra

Verifica che la data di inizio di una nuova guerra non sia precedente all'ultima occupazione registrata per quel pianeta.

```
CREATE OR REPLACE TRIGGER check_data_guerra
BEFORE INSERT ON Guerra_Conquista
FOR EACH ROW
DECLARE
    v_ultima_occupazione DATE;
BEGIN
    SELECT MAX(anno_conquista)
    INTO v_ultima_occupazione
    FROM Occupazione
    WHERE nome_stella_principale = :NEW.nome_stella_principale
    AND nome_pianeta = :NEW.nome_pianeta;

    -- Se esiste un'occupazione precedente e la nuova guerra inizia prima
    IF :NEW.data_inizio < v_ultima_occupazione AND v_ultima_occupazione IS NOT NULL
    THEN
        RAISE_APPLICATION_ERROR(-20002, 'Data guerra (' || TO_CHAR(:NEW.data_inizio,
    'YYYY-MM-DD') ||
                            ') precedente all''ultima occupazione (' ||
                            TO_CHAR(v_ultima_occupazione, 'YYYY-MM-DD') || ')');
    END IF;
END;
/
```

Listing 38: triggers.sql

4.6 Controllo Data Occupazione

Assicura che la data di occupazione sia successiva alla data di inizio della guerra corrispondente.

```
CREATE OR REPLACE TRIGGER check_data_occupazione
BEFORE INSERT ON Occupazione
FOR EACH ROW
DECLARE
    data_inizio_guerra DATE;
BEGIN
    -- Ottiene la data di inizio dell'ultima guerra per questo pianeta
    SELECT MAX(data_inizio) INTO data_inizio_guerra
    FROM Guerra_Conquista
    WHERE nome_stella_principale = :NEW.nome_stella_principale
    AND nome_pianeta = :NEW.nome_pianeta;

    -- Verifica che la data di occupazione sia successiva alla data di inizio guerra
    IF :NEW.anno_conquista < data_inizio_guerra THEN
        RAISE_APPLICATION_ERROR(-20002, 'La data di occupazione deve essere
    successiva alla data di inizio della guerra');
    END IF;
END;
```

Listing 39: triggers.sql

4.7 Controllo Guerra Prima di Occupazione

Impedisce l'inserimento di un'occupazione senza una corrispondente guerra registrata.

```
CREATE OR REPLACE TRIGGER check_guerra_prima_di_occupazione
BEFORE INSERT ON Occupazione
FOR EACH ROW
DECLARE
    v_guerra_esiste NUMBER;
BEGIN
    -- Verifica se esiste una guerra corrispondente
```

```
    SELECT COUNT(*) INTO v_guerra_esiste
    FROM Guerra_Conquista
    WHERE data_inizio = :NEW.data_inizio
    AND nome_stella_principale = :NEW.nome_stella_principale
    AND nome_pianeta = :NEW.nome_pianeta;

    -- Se non esiste la guerra corrispondente
    IF v_guerra_esiste = 0 THEN
        RAISE_APPLICATION_ERROR(-20003,
            'Non e' possibile inserire un''occupazione senza una corrispondente guerra. ' ||
            'Prima crea una guerra per ' || :NEW.nome_pianeta ||
            ' nel sistema ' || :NEW.nome_stella_principale);
    END IF;
END;
```

Listing 40: triggers.sql

4.8 Controllo Viaggio con Guerra Attiva

Blocca i viaggi che partono o arrivano su pianeti con guerre attive.

```
CREATE OR REPLACE TRIGGER check_viaggio_guerra_attiva
BEFORE INSERT ON Viaggio
FOR EACH ROW
DECLARE
    guerra_attiva_partenza NUMBER;
    guerra_attiva_destinazione NUMBER;
BEGIN
    -- Controlla guerra attiva al pianeta di partenza
    SELECT COUNT(*) INTO guerra_attiva_partenza
    FROM Guerra_Conquista gc
    LEFT JOIN Occupazione o ON gc.data_inizio = o.data_inizio
                            AND gc.nome_stella_principale = o.nome_stella_principale
                            AND gc.nome_pianeta = o.nome_pianeta
    WHERE gc.nome_stella_principale = :NEW.nome_stella_principale_partenza
    AND gc.nome_pianeta = :NEW.nome_pianeta_partenza
    AND o.anno_conquista IS NULL;

    -- Controlla guerra attiva al pianeta di destinazione
    SELECT COUNT(*) INTO guerra_attiva_destinazione
    FROM Guerra_Conquista gc
    LEFT JOIN Occupazione o ON gc.data_inizio = o.data_inizio
                            AND gc.nome_stella_principale = o.nome_stella_principale
                            AND gc.nome_pianeta = o.nome_pianeta
    WHERE gc.nome_stella_principale = :NEW.nome_stella_principale_destinazione
    AND gc.nome_pianeta = :NEW.nome_pianeta_destinazione
    AND o.anno_conquista IS NULL;

    IF guerra_attiva_partenza > 0 THEN
        RAISE_APPLICATION_ERROR(-20003,
            'Viaggio bloccato: guerra attiva sul pianeta di partenza (' ||
            :NEW.nome_pianeta_partenza || ')');
    END IF;

    -- Controllo separato per destinazione
    IF guerra_attiva_destinazione > 0 THEN
        RAISE_APPLICATION_ERROR(-20004,
            'Viaggio bloccato: guerra attiva sul pianeta di destinazione (' ||
            :NEW.nome_pianeta_destinazione || ')');
    END IF;
END;
```

Listing 41: triggers.sql

4.9 Controllo Viaggio con Stessa Destinazione

Impedisce l'inserimento di viaggi con partenza e destinazione uguali.

```
CREATE OR REPLACE TRIGGER check_viaggio_stessa_destinazione
BEFORE INSERT ON Viaggio
FOR EACH ROW
BEGIN
    -- Verifica se il pianeta di partenza e destinazione sono uguali
    IF :NEW.nome_pianeta_partenza = :NEW.nome_pianeta_destinazione AND
       :NEW.nome_stella_principale_partenza = :NEW.
    nome_stella_principale_destinazione THEN

        RAISE_APPLICATION_ERROR(-20005,
            'Il viaggio non puo' avere lo stesso pianeta di partenza e destinazione:
    ' ||
            :NEW.nome_pianeta_partenza || ' nel sistema ' ||
            :NEW.nome_stella_principale_partenza);
    END IF;
END;
```

Listing 42: triggers.sql

5 Procedure e funzioni

5.1 Procedure dell'Imperatore

1. `fine_guerra` La procedura `fine_guerra` permette di concludere una guerra di conquista, assegnando il controllo del pianeta conquistato a una `Corporation` selezionata mediante la funzione `determina_corporation`. Inoltre, distribuisce equamente eventuali perdite tra le divisioni imperiali coinvolte nella guerra e registra l'occupazione nella tabella `Occupazione` con `anno_conquista` uguale alla data in cui viene lanciata la procedura[12]. Se le perdite cumulative superano gli effettivi originali della divisione, viene notificata la distruzione della stessa. In caso di errore, viene effettuato il rollback della transazione.

```
CREATE OR REPLACE PROCEDURE fine_guerra (
    p_data_inizio_guerra IN DATE,
    p_nome_stella_principale IN VARCHAR2,
    p_nome_pianeta IN VARCHAR2,
    p_popolazione IN NUMBER, -- la popolazione e' una variabile messa all'inserimento
     per la creazione della nuova tupla Occupazione
    p_perdite_totali IN NUMBER DEFAULT 0 -- Default: 0, nessuna perdita
)
IS
    v_corporation_scelta VARCHAR2(50);
    v_anno_conquista DATE := SYSDATE + 100*365; -- Usa la data corrente (+ 100 anni)
    di quando viene lanciata la procedura
    v_divisioni_coinvolte NUMBER;
    v_perdite_per_divisione NUMBER;
BEGIN

   -- Calcola e distribuisce le perdite tra le divisioni coinvolte
    IF p_perdite_totali > 0 THEN
        -- Conta quante divisioni hanno partecipato alla guerra
        SELECT COUNT(*) INTO v_divisioni_coinvolte
        FROM combatte2
        WHERE data_inizio = p_data_inizio_guerra
          AND nome_stella_principale = p_nome_stella_principale
          AND nome_pianeta = p_nome_pianeta;

        IF v_divisioni_coinvolte = 0 THEN
```

[12] Nella procedura viene aggiunta a SYSDATE 100 anni (365*100 perché Oracole conta le date in giorni) per coerenza interna del database, dato che le date gia inserite sono 100 anni nel futuro al minimo

```
25          DBMS_OUTPUT.PUT_LINE('Attenzione: Perdite totali specificate (' ||
    p_perdite_totali ||
26                               ') ma nessuna divisione imperiale risulta coinvolta
    nella guerra.');
27      ELSE
28          -- Calcola perdite per divisione (distribuzione equa -> potremmo
    assegnare le perdite in modo casuale, ma    cos  e' pi  semplice da
    implementare)
29          v_perdite_per_divisione := FLOOR(p_perdite_totali / v_divisioni_coinvolte
    );
30
31          -- Aggiorna le perdite per ogni divisione e verifica se e' stata
    distrutta
32          FOR divisione IN (
33              SELECT d.numero_armata, d.nome_armata, d.effettivi AS
    Effettivi_Originali,
34                     NVL(SUM(c.numero_perdite), 0) AS Perdite_Totali
35              FROM Divisione_Imperiale d
36              LEFT JOIN combatte2 c ON d.numero_armata = c.numero_armata
37              WHERE c.data_inizio = p_data_inizio_guerra
38                AND c.nome_stella_principale = p_nome_stella_principale
39                AND c.nome_pianeta = p_nome_pianeta
40              GROUP BY d.numero_armata, d.nome_armata, d.effettivi
41          ) LOOP
42              UPDATE combatte2
43              SET numero_perdite = NVL(numero_perdite, 0) + v_perdite_per_divisione
44              WHERE data_inizio = p_data_inizio_guerra
45                AND nome_stella_principale = p_nome_stella_principale
46                AND nome_pianeta = p_nome_pianeta
47                AND numero_armata = divisione.numero_armata;
48
49              -- Verifica se la divisione    stata distrutta e manda il messaggio
    di distruzione
50              -- Abbiamo fatto in modo che le perdite vengono calcolate solo alla
    fine della guerra, quindi non ci sono perdite parziali (mid-guerra)
51              IF (divisione.Perdite_Totali + v_perdite_per_divisione) >= divisione.
    Effettivi_Originali THEN
52                  DBMS_OUTPUT.PUT_LINE('Divisione distrutta: ' || divisione.
    nome_armata ||
53                                       ' (Codice: ' || divisione.numero_armata ||
54                                       '), Perdite: ' || (divisione.Perdite_Totali +
     v_perdite_per_divisione) ||
55                                       '/' || divisione.Effettivi_Originali);
56              ELSE
57                  DBMS_OUTPUT.PUT_LINE('Aggiornate ' || v_perdite_per_divisione ||
58                                       ' perdite per la divisione ' || divisione.
    numero_armata);
59              END IF;
60          END LOOP;
61      END IF;
62  END IF;
63
64  -- Usa la funzione determina_corporation per assegnare una corporation al pianeta
     conquistato
65  v_corporation_scelta := determina_corporation(
66      p_data_inizio => p_data_inizio_guerra,
67      p_stella_principale => p_nome_stella_principale,
68      p_pianeta => p_nome_pianeta
69  );
70
71  INSERT INTO Occupazione (
72      anno_conquista,
73      nome_stella_principale,
74      nome_pianeta,
75      data_inizio,
76      popolazione,
```

```
        nome_corporation
    ) VALUES (
        v_anno_conquista,   -- Usa SYSDATE qui
        p_nome_stella_principale,
        p_nome_pianeta,
        p_data_inizio_guerra,
        p_popolazione,
        v_corporation_scelta
    );

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Guerra conclusa con successo. Corporation ' ||
    v_corporation_scelta ||
                         ' assegnata al pianeta ' || p_nome_pianeta || ' nel sistema
    ' || p_nome_stella_principale ||
                         ' in data ' || TO_CHAR(v_anno_conquista, 'DD/MM/YYYY') ||
                         '. Perdite totali: ' || p_perdite_totali);
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Errore durante la conclusione della guerra: ' ||
    SQLERRM);
        RAISE;
END fine_guerra;
```

Listing 43: Script: procedure.sql

2. `dichiara_guerra` La procedura `dichiara_guerra` avvia formalmente una guerra di conquista su un pianeta abitabile con `data_inizio` uguale alla data in cui viene lanciata la procedura[13]. Inserisce una nuova tupla nella tabella `Guerra_Conquista`, e se il pianeta è abitato da ribelli, associa (o crea, se necessario) un `Gruppo_Ribelle` al conflitto tramite la tabella `combatte1`. I parametri consentono di specificare anche il livello di pericolo dei ribelli. In caso di errore, viene effettuato il rollback della transazione.

```
CREATE OR REPLACE PROCEDURE dichiara_guerra(
    p_nome_stella_principale IN VARCHAR2,
    p_nome_pianeta IN VARCHAR2,
    p_nome_operazione IN VARCHAR2,
    p_ha_ribelli IN NUMBER DEFAULT 0, -- 0 = no ribelli, 1 = con ribelli (DEFAULT:
    pianeta disabitato)
    p_livello_pericolo IN NUMBER DEFAULT 0 -- l'Impero prima di attaccare fa una
    ricognizione per stabilire il livello di pericolo (DEFAULT: 0, pianeta disabitato
    )
)
IS
    v_esiste_ribelli NUMBER;
    v_nome_armata_ribelle VARCHAR2(50);
    v_e_abitabile NUMBER;
    v_data_inizio DATE := SYSDATE + 100*365; -- Usa la data corrente (+ 100 anni) di
    quando viene lanciata la procedura
BEGIN
    -- Verifica che il pianeta sia abitabile, dato che non facciamo guerre su pianeti
     inospitali
    SELECT abitabilita INTO v_e_abitabile
    FROM pianeta PI
    WHERE PI.nome_pianeta = p_nome_pianeta
        AND PI.nome_stella_principale = p_nome_stella_principale;

    IF v_e_abitabile = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Il pianeta ' || p_nome_pianeta ||
                                ' non    abitabile');
    END IF;
```

[13]Nella procedura viene aggiunta a SYSDATE 100 anni (365*100 perché Oracole conta le date in giorni) per coerenza interna del database, dato che le date gia inserite sono 100 anni nel futuro al minimo

```
24
25      -- Inserisce la nuova guerra
26      INSERT INTO Guerra_Conquista (
27          data_inizio,
28          nome_stella_principale,
29          nome_pianeta,
30          nome_operazione
31      ) VALUES (
32          v_data_inizio,
33          p_nome_stella_principale,
34          p_nome_pianeta,
35          p_nome_operazione
36      );
37
38      -- Se ci sono ribelli, crea o associa il gruppo ribelle
39      IF p_ha_ribelli = 1 THEN
40          v_nome_armata_ribelle := 'Ribelli ' || p_nome_stella_principale;
41
42          -- Verifica se esiste gi  un gruppo ribelle per questo sistema
43          SELECT COUNT(*) INTO v_esiste_ribelli
44          FROM Gruppo_Ribelle
45          WHERE nome_armata = v_nome_armata_ribelle;
46
47          -- Se non esiste, lo crea
48          IF v_esiste_ribelli = 0 THEN
49              INSERT INTO Gruppo_Ribelle (
50                  nome_armata,
51                  livello_pericolo,
52                  stato
53              ) VALUES (
54                  v_nome_armata_ribelle,
55                  p_livello_pericolo,
56                  'Attivo'
57              );
58          END IF;
59
60          -- Associa il gruppo ribelle alla guerra
61          INSERT INTO combatte1 (
62              nome_armata_ribelle,
63              data_inizio,
64              nome_stella_principale,
65              nome_pianeta
66          ) VALUES (
67              v_nome_armata_ribelle,
68              v_data_inizio,
69              p_nome_stella_principale,
70              p_nome_pianeta
71          );
72      END IF;
73
74      COMMIT;
75      DBMS_OUTPUT.PUT_LINE('Guerra dichiarata con successo sul pianeta ' ||
        p_nome_pianeta ||
76                           ' nel sistema ' || p_nome_stella_principale ||
77                           ' con data inizio: ' || TO_CHAR(v_data_inizio, 'DD/MM/YYYY'))
        ;
78  EXCEPTION
79      WHEN OTHERS THEN
80          ROLLBACK;
81          DBMS_OUTPUT.PUT_LINE('Errore durante la dichiarazione di guerra: ' || SQLERRM
        );
82          RAISE;
83  END dichiara_guerra;
```

Listing 44: Script: procedure.sql

5.2 Procedure Generali

1. `assegna_divisione` La procedura `assegna_divisione` consente di inviare una divisione imperiale in rinforzo a una guerra attiva. Prima dell'inserimento nella tabella `combatte2`, vengono effettuati controlli per assicurarsi che:

- la divisione abbia un numero minimo di effettivi disponibili (tramite la vista `vista_effettivi_attuali`);
- la guerra esista ed è ancora attiva (tramite `vista_guerre_attive`);
- la divisione non sia già assegnata a quella guerra.

```
CREATE OR REPLACE PROCEDURE assegna_divisione(
    p_data_inizio_guerra IN DATE,
    p_nome_stella_principale IN VARCHAR2,
    p_nome_pianeta IN VARCHAR2,
    p_numero_armata IN VARCHAR2
)
IS
    v_guerra_attiva NUMBER;
    v_divisione_esiste NUMBER;
    v_divisione_gia_inviata NUMBER;
    v_effettivi_attuali NUMBER;
    v_min_effettivi NUMBER := 100; -- Soglia minima di effettivi
BEGIN
    -- Verifica che la divisione abbia abbastanza effettivi
    SELECT Effettivi_Attuali INTO v_effettivi_attuali
    FROM vista_effettivi_attuali
    WHERE numero_armata = p_numero_armata;

    IF v_effettivi_attuali < v_min_effettivi THEN
        RAISE_APPLICATION_ERROR(-20001, 'La divisione ' || p_numero_armata ||
                                ' ha solo ' || v_effettivi_attuali ||
                                ' effettivi disponibili (minimo richiesto: ' ||
    v_min_effettivi || ')');
    END IF;

    -- Verifica che la guerra esista ed    attiva
    SELECT COUNT(*) INTO v_guerra_attiva
    FROM vista_guerre_attive
    WHERE data_inizio = p_data_inizio_guerra
      AND nome_stella_principale = p_nome_stella_principale
      AND nome_pianeta = p_nome_pianeta;

    IF v_guerra_attiva = 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'La guerra specificata non esiste o    gia
    conclusa');
    END IF;

    -- Verifica che la divisione non sia gia stata assegnata a questa guerra
    SELECT COUNT(*) INTO v_divisione_gia_inviata
    FROM combatte2
    WHERE data_inizio = p_data_inizio_guerra
      AND nome_stella_principale = p_nome_stella_principale
      AND nome_pianeta = p_nome_pianeta
      AND numero_armata = p_numero_armata;

    IF v_divisione_gia_inviata > 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'La divisione ' || p_numero_armata ||
                                '    gia stata assegnata a questa guerra');
    END IF;

    -- Inserisce la divisione nella guerra
    INSERT INTO combatte2 (
        numero_armata,
        data_inizio,
```

```
        nome_stella_principale,
        nome_pianeta,
        numero_perdite
    ) VALUES (
        p_numero_armata,
        p_data_inizio_guerra,
        p_nome_stella_principale,
        p_nome_pianeta,
        0 -- Inizialmente le perdite sono 0
    );

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Rinforzi inviati con successo. Divisione ' ||
    p_numero_armata ||
                         ' assegnata alla guerra su ' || p_nome_pianeta ||
                         ' nel sistema ' || p_nome_stella_principale ||
                         ' con ' || v_effettivi_attuali || ' effettivi disponibili');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Errore: Divisione ' || p_numero_armata || ' non trovata
    ');
        RAISE;
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Errore durante l''invio dei rinforzi: ' || SQLERRM);
        RAISE;
END assegna_divisione;
```

Listing 45: Script: procedure.sql

2. `rinforza_divisione` La procedura `rinforza_divisione` consente di aumentare il numero totale di effettivi di una divisione imperiale esistente, aggiornando il valore della colonna `effettivi` nella tabella `Divisione_Imperiale`. L'operazione è consentita solo se il numero di effettivi da aggiungere è strettamente positivo. In caso contrario, la procedura solleva un'eccezione con codice d'errore personalizzato.

```
CREATE OR REPLACE PROCEDURE rinforza_divisione(
    p_numero_armata IN VARCHAR2,
    p_effettivi_da_aggiungere IN NUMBER
) AS
    v_effettivi_attuali NUMBER;
    v_effettivi_nuovi NUMBER;
BEGIN
    -- Verifica che gli effettivi da aggiungere siano positivi
    IF p_effettivi_da_aggiungere <= 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Il numero di effettivi da aggiungere deve
    essere positivo');
    END IF;

    -- Recupera gli effettivi attuali della divisione
    SELECT effettivi INTO v_effettivi_attuali
    FROM Divisione_Imperiale
    WHERE numero_armata = p_numero_armata;

    -- Calcola i nuovi effettivi
    v_effettivi_nuovi := v_effettivi_attuali + p_effettivi_da_aggiungere;

    -- Esegui l'aggiornamento
    UPDATE Divisione_Imperiale
    SET effettivi = v_effettivi_nuovi
    WHERE numero_armata = p_numero_armata;

    -- Output di conferma
    DBMS_OUTPUT.PUT_LINE('Divisione ' || p_numero_armata || ' aggiornata:');
```

```
28      DBMS_OUTPUT.PUT_LINE('Effettivi precedenti: ' || v_effettivi_attuali);
29      DBMS_OUTPUT.PUT_LINE('Effettivi incrementare : ' || v_effettivi_nuovi);
30
31  EXCEPTION
32      WHEN NO_DATA_FOUND THEN
33          RAISE_APPLICATION_ERROR(-20003, 'Divisione con numero armata ' ||
        p_numero_armata || ' non trovata');
34      WHEN OTHERS THEN
35          RAISE_APPLICATION_ERROR(-20004, 'Errore durante l''aggiornamento: ' ||
        SQLERRM);
36  END rinforza_divisione;
```

Listing 46: Script: procedure.sql

5.3 Procedure del Dirigente di Corporation

1. `finanzia_guerra` La procedura `finanzia_guerra` consente a una `Corporation` di finanziare una guerra di conquista in corso, specificando l'importo del contributo. Vengono effettuati i seguenti controlli:

- Se esiste già almeno un finanziamento per la guerra indicata, il contributo proposto deve essere maggiore o uguale al minimo già versato; in caso contrario, viene sollevata un'eccezione con messaggio informativo, impedendo il cosiddetto *lowballing*.
- Se non esiste alcun finanziamento registrato per la guerra, il contributo viene accettato senza vincoli.

```
1   CREATE OR REPLACE PROCEDURE finanzia_guerra(
2       p_nome_corporation IN VARCHAR2,
3       p_data_inizio_guerra IN DATE,
4       p_nome_stella_principale IN VARCHAR2,
5       p_nome_pianeta IN VARCHAR2,
6       p_valore_contributo IN NUMBER
7   )
8   IS
9       v_min_contributo NUMBER;
10  BEGIN
11      -- Verifica se esiste gia un finanziamento per questa guerra
12      SELECT MIN(valore_contributo)
13      INTO v_min_contributo
14      FROM finanzia
15      WHERE data_inizio = p_data_inizio_guerra
16        AND nome_stella_principale = p_nome_stella_principale
17        AND nome_pianeta = p_nome_pianeta;
18
19      -- Se esiste un finanziamento precedente e il nuovo    minore, dai errore
20      IF v_min_contributo IS NOT NULL AND p_valore_contributo < v_min_contributo THEN
21          RAISE_APPLICATION_ERROR(-20001, 'Il finanziamento non pu  essere inferiore a
        ' || v_min_contributo || ' a quelli gia fatti per questa guerra.
22          L impero non accetta lowballing');
23      END IF;
24
25      -- Inserisce il finanziamento
26      INSERT INTO finanzia (
27          nome_corporation,
28          data_inizio,
29          nome_stella_principale,
30          nome_pianeta,
31          valore_contributo
32      ) VALUES (
33          p_nome_corporation,
34          p_data_inizio_guerra,
35          p_nome_stella_principale,
36          p_nome_pianeta,
```

```
        p_valore_contributo
    );

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Finanziamento registrato con successo. ' ||
                         p_nome_corporation || ' ha contribuito con ' ||
                         p_valore_contributo || ' alla guerra su ' ||
                         p_nome_pianeta || ' nel sistema ' ||
    p_nome_stella_principale);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Nessun finanziamento esistente, procedi con l'inserimento
        INSERT INTO finanzia (
            nome_corporation,
            data_inizio,
            nome_stella_principale,
            nome_pianeta,
            valore_contributo
        ) VALUES (
            p_nome_corporation,
            p_data_inizio_guerra,
            p_nome_stella_principale,
            p_nome_pianeta,
            p_valore_contributo
        );
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Finanziamento registrato con successo. ' ||
                             p_nome_corporation || ' ha contribuito con ' ||
                             p_valore_contributo || ' alla guerra su ' ||
                             p_nome_pianeta || ' nel sistema ' ||
    p_nome_stella_principale);

    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Errore durante il finanziamento della guerra: ' ||
    SQLERRM);
        RAISE;
END finanzia_guerra;
```

Listing 47: Script: procedure.sql

La funzione `determina_corporation` si occupa di determinare quale `Corporation` debba ricevere il controllo di un pianeta specifico in seguito a una guerra di conquista, in base a un criterio di priorità e riequilibrio del potere economico e territoriale.

1. **Corporation primaria**: se la guerra è stata finanziata da una o più corporation, si seleziona quella che ha versato il contributo economico maggiore specificamente per quella guerra.

2. **Corporation secondaria**: se la corporation primaria possiede già oltre il 50% dei pianeti dello stesso sistema stellare (ossia con la stessa `stella_principale`), allora si valuta di assegnare il pianeta alla seconda corporation con il maggior finanziamento.

3. **Corporation assoluta**: nel caso in cui non esista alcun finanziatore per la guerra, oppure né la corporation primaria né la secondaria siano valide per l'assegnazione (ad esempio per superamento della soglia del 50% senza un secondo candidato valido), il pianeta viene assegnato alla corporation con la somma totale più alta di finanziamenti in assoluto (cioè considerando tutte le guerre finanziate, indipendentemente dal pianeta o sistema).

Eccezioni gestite

- Se non esistono corporation finanziatrici per la guerra specifica, viene eseguita direttamente la selezione assoluta.

- In caso di assenza di dati (nessuna corporation valida per qualsiasi criterio), la funzione restituisce `NULL`.

```
CREATE OR REPLACE FUNCTION determina_corporation(
    p_data_inizio DATE,
    p_stella_principale VARCHAR2,
    p_pianeta VARCHAR2
) RETURN VARCHAR2
IS
    v_corporation_primaria VARCHAR2(50);
    v_corporation_secondaria VARCHAR2(50);
    v_corporation_assoluto VARCHAR2(50); -- Corporation con finanziamenti totali pi
    alti
    v_totale_pianeti_sistema NUMBER;
    v_pianeti_corporation NUMBER;
    v_percentuale NUMBER;
    v_ha_finanziatori NUMBER;
BEGIN
    -- 1. Controlla se ci sono finanziatori per questa guerra specifica
    SELECT COUNT(*) INTO v_ha_finanziatori
    FROM finanzia f
    WHERE f.data_inizio = p_data_inizio
      AND f.nome_stella_principale = p_stella_principale
      AND f.nome_pianeta = p_pianeta;

    -- 2. Se ci sono finanziatori per questa guerra
    IF v_ha_finanziatori > 0 THEN
        -- Trova la corporation con il finanziamento maggiore per questa guerra
        BEGIN
            SELECT nome_corporation INTO v_corporation_primaria
            FROM (
                SELECT f.nome_corporation, SUM(f.valore_contributo) as
   totale_finanziamenti
                FROM finanzia f
                WHERE f.data_inizio = p_data_inizio
                  AND f.nome_stella_principale = p_stella_principale
                  AND f.nome_pianeta = p_pianeta
                GROUP BY f.nome_corporation
                HAVING SUM(f.valore_contributo) = (
                    SELECT MAX(SUM(f2.valore_contributo))
                    FROM finanzia f2
                    WHERE f2.data_inizio = p_data_inizio
                      AND f2.nome_stella_principale = p_stella_principale
                      AND f2.nome_pianeta = p_pianeta
                    GROUP BY f2.nome_corporation
                )
            );
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                v_corporation_primaria := NULL;
        END;

        -- Trova la seconda corporation per finanziamenti per questa guerra
        BEGIN
            SELECT nome_corporation INTO v_corporation_secondaria
            FROM (
                SELECT f.nome_corporation
                FROM finanzia f
                WHERE f.data_inizio = p_data_inizio
                  AND f.nome_stella_principale = p_stella_principale
                  AND f.nome_pianeta = p_pianeta
                  AND f.nome_corporation != v_corporation_primaria
                GROUP BY f.nome_corporation
                HAVING SUM(f.valore_contributo) = (
                    SELECT MAX(SUM(f2.valore_contributo))
                    FROM finanzia f2
```

```
                    WHERE f2.data_inizio = p_data_inizio
                      AND f2.nome_stella_principale = p_stella_principale
                      AND f2.nome_pianeta = p_pianeta
                      AND f2.nome_corporation != v_corporation_primaria
                    GROUP BY f2.nome_corporation
                )
            );
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                v_corporation_secondaria := NULL;
        END;

        -- Conta quanti pianeti ci sono nel sistema
        SELECT COUNT(*) INTO v_totale_pianeti_sistema
        FROM Pianeta
        WHERE nome_stella_principale = p_stella_principale;

        -- Conta quanti pianeti la corporation primaria ha gi  nel sistema (cio attorno alla stessa stella)
        IF v_corporation_primaria IS NOT NULL THEN
            SELECT COUNT(*) INTO v_pianeti_corporation
            FROM Occupazione o
            JOIN Guerra_Conquista g ON o.data_inizio = g.data_inizio
                                    AND o.nome_stella_principale = g.nome_stella_principale
                                    AND o.nome_pianeta = g.nome_pianeta
            WHERE o.nome_corporation = v_corporation_primaria
              AND g.nome_stella_principale = p_stella_principale;

            -- Calcola la percentuale
            v_percentuale := (v_pianeti_corporation / v_totale_pianeti_sistema) * 100;

            -- Se supera il 50%, usa la secondaria (se esiste)
            IF v_percentuale > 50 AND v_corporation_secondaria IS NOT NULL THEN
                RETURN v_corporation_secondaria;
            ELSIF v_percentuale > 50 AND v_corporation_secondaria IS NULL THEN
                -- Se non c'  una seconda corporation, passa al assoluto
                NULL; -- Continua alla corporation assoluta
            ELSE
                RETURN v_corporation_primaria;
            END IF;
        END IF;
    END IF;

    -- 3. Se non ci sono finanziatori per questa guerra o non si pu  assegnare la corporation primaria/secondaria
    -- Trova la corporation con il finanziamento totale pi  alto in assoluto
    BEGIN
        SELECT nome_corporation INTO v_corporation_assoluto
        FROM (
            SELECT f.nome_corporation, SUM(f.valore_contributo) as totale_finanziamenti
            FROM finanzia f
            GROUP BY f.nome_corporation
            HAVING SUM(f.valore_contributo) = (
                SELECT MAX(SUM(valore_contributo))
                FROM finanzia
                GROUP BY nome_corporation
            )
        );

        RETURN v_corporation_assoluto;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
```

```
123     END;
124 END;
```

Listing 48: Script: funzioni.sql

6 Viste

6.1 vista_effettivi_attuali

La vista effettivi attuali ci permette di derivare il numero degli effettivi "veri", ovvero quelli attualmente attivi (quindi non deceduti in battaglia), per fare questo trova nella tabella di transizione combatte2 tutte le guerre combattute da una determinata divisione, e sottrae al numero di effettivi "globali" tutti i numeri delle perdite della data divisione.

- Il numero degli effettivi attuali varia quindi ogni volta che viene terminata una guerra con un numero di perdite maggiore di zero

Questa vista è necessaria al verificare alcuni vincoli dinamici(check_perdite_NO_negative) e ad informare il generale e l'imperatore della situazione attuale delle divisioni. Quando questo numero di effettivi di una divisione è troppo piccolo, diventa impossibile inviare la stessa in guerra e deve essere lanciata la procedura rinforza_divisione.

```
1  CREATE OR REPLACE VIEW vista_effettivi_attuali AS
2  SELECT
3      d.numero_armata,
4      d.nome_armata,
5      NVL(SUM(c.numero_perdite), 0) AS Perdite_Totali,
6      d.effettivi - NVL(SUM(c.numero_perdite), 0) AS Effettivi_Attuali
7  FROM
8      Divisione_Imperiale d
9  LEFT JOIN
10     combatte2 c ON d.numero_armata = c.numero_armata
11 GROUP BY
12     d.numero_armata, d.nome_armata, d.effettivi
13 ORDER BY
14     d.numero_armata;
```

Listing 49: Script: viste.sql

6.2 vista_guerre_attive e vista_guerre_finite

Queste due viste riportano rispettivamente tutte le guerre ancora attive(dove quindi ancora non è stata generata una relativa occupazione) e tutte le guerre finite(dove esiste una relazione con occupazione).

- Entrambe le viste vengono usate nei trigger per verificare, ad esempio che non venga avviata una guerra su un pianeta con già una guerra esistente.

ATTENZIONE: La seconda vista è derivabile dalla prima(visto che tutti i pianeti non presenti in vista_guerre_attive fanno parte invece della seconda), ma viene comunque esplicitata per comodità e facilità di lettura

```
1  CREATE VIEW vista_guerre_attive AS
2  SELECT gc.*
3  FROM Guerra_Conquista gc
4  WHERE NOT EXISTS (
5      SELECT 1
6      FROM Occupazione o
7      WHERE o.data_inizio = gc.data_inizio
8        AND o.nome_pianeta = gc.nome_pianeta
9        AND o.nome_stella_principale = gc.nome_stella_principale
10 );
```

```sql

CREATE VIEW vista_guerre_finite AS
SELECT
    gc.*,
    o.anno_conquista AS data_fine
FROM Guerra_Conquista gc
JOIN Occupazione o
  ON o.data_inizio = gc.data_inizio
 AND o.nome_pianeta = gc.nome_pianeta
 AND o.nome_stella_principale = gc.nome_stella_principale;
```

Listing 50: Script: viste.sql

6.3 Data Control Language

L'imperatore, figura centrale e autoritaria del sistema imperiale, è un maniaco del controllo e detiene il massimo livello di accesso all'interno del database. Può:

- Visualizzare qualsiasi tabella del database dell'Impero
- Eseguire tramite procedura tutte le scelte strategiche fondamentali, come la dichiarazione e la conclusione delle guerre
- Assegnare le divisioni militari alle guerre secondo il proprio giudizio
- Creare nuove Corporation, specificando per ciascuna il dirigente responsabile

```sql
GRANT CREATE SESSION TO imperatore;
GRANT CONNECT TO imperatore;
GRANT SELECT ANY TABLE TO imperatore;
GRANT INSERT ON Corporation TO imperatore;
GRANT EXECUTE ON fine_guerra TO imperatore;
GRANT EXECUTE ON dichiara_guerra TO imperatore;
GRANT EXECUTE ON assegna_divisione TO imperatore;
```

Listing 51: Script: users.sql

Il generale rappresenta il comando militare sul campo e dispone di un accesso limitato ma operativo. Le sue funzioni sono:

- Visualizzare le guerre attive e concluse attraverso apposite viste
- Consultare la forza attuale delle divisioni mediante la vista `vista_effettivi_attuali`
- Aggiornare lo stato dei gruppi ribelli (ad esempio, da attivo a distrutto) una volta verificata la loro eliminazione
- Inserire nuovi stazionamenti e collegarli alle divisioni presidiate;
- Promuovere soldati a Soth, ovvero comandanti di flotta
- Rinforzare le divisioni tramite l'apposita procedura, aumentando il numero di effettivi
- Partecipare all'assegnazione delle divisioni e alla conclusione dei conflitti

```sql
GRANT CREATE SESSION TO generale;
GRANT CONNECT TO generale;
GRANT SELECT ON vista_effettivi_attuali TO generale;
GRANT SELECT ON vista_guerre_attive TO generale;
GRANT SELECT ON vista_guerre_finite TO generale;
GRANT UPDATE ON Gruppo_Ribelle TO generale;
GRANT SELECT, INSERT ON Stazionamento TO generale;
GRANT SELECT, INSERT ON e_presidiato TO generale;
GRANT SELECT, INSERT ON Soth TO generale;
```

```
GRANT EXECUTE ON fine_guerra TO generale;
GRANT EXECUTE ON assegna_divisione TO generale;
GRANT EXECUTE ON rinforza_divisione TO generale;
```

Listing 52: Script: users.sql

Il dirigente di Corporation gestisce gli aspetti economici e logistici. I suoi poteri includono:

- La creazione di nuovi viaggi commerciali o militari (carico, trasporto, spostamento)
- La gestione e inserimento di nuove corporation, suddivise per tipologia (bellica, mineraria, trasporti)
- La possibilità di finanziare una nuova guerra di conquista mediante l'apposita procedura

```
GRANT CREATE SESSION TO dirigente_corporation;
GRANT CONNECT TO dirigente_corporation;
GRANT SELECT ON Corporation TO dirigente_corporation;
GRANT SELECT, INSERT ON Corporation_Bellica TO dirigente_corporation;
GRANT SELECT, INSERT ON Corporation_Mineraria TO dirigente_corporation;
GRANT SELECT, INSERT ON Corporation_Trasporti TO dirigente_corporation;
GRANT SELECT, INSERT ON Viaggio TO dirigente_corporation;
GRANT EXECUTE ON finanzia_guerra TO dirigente_corporation;
```

Listing 53: Script: users.sql