Bortolussi Lorenzo SM3201370

Corso di Intelligenza Artificiale e Data Analitycs

PROGETTO DI BASI DI DATI

PRESENTAZIONE DEL PROGETTO

Si vuole realizzare una base dati di supporto per il videogioco Deep Rock Galactic, sparatutto cooperativo in prima persona in cui i giocatori partono da una base spaziale per affrontare missioni nelle caverne di un pianeta alieno.

In particolare, il progetto è incentrato sulla meccanica delle birre presenti nella stazione spaziale, oggetti consumabili che conferiscono potenziamenti temporanei ma soprattutto effetti stravaganti ai giocatori. Questi oggetti richiedono materiali non comuni per pagarne la licenza e per gli acquisti successivi.

Il progetto seguente ha l'obiettivo di consigliare ai giocatori le missioni da intraprendere per ottenere la maggior quantità di ognuno dei quattro materiali utilizzati come valuta, basandosi su uno storico di partite passate.

REQUISITI DELLA BASE DATI

Deve essere presente l'insieme delle birre disponibili con relativi costi di licenza. Ogni giocatore deve avere una lista delle licenze in suo possesso.

Per ogni giocatore è presente l'elenco di partite da lui giocate, ognuna contenente classe utilizzata, incarico intrapreso, anomalia se presente, bioma e materiali

ottenuti. Ogni partita ha associata la versione del gioco in cui è stata svolta ed un ID crescente basato sull'ordine di inserimento.

A partire dalle informazioni sulle partite giocate, è possibile ricavare i consigli sulle missioni da intraprendere, composte di classe, incarico, anomalia, bioma. L'utente può inserire un determinato materiale e ottenere la combinazione che ha portato ad ottenere la maggiore quantità del materiale richiesto.

Per calcolare i suggerimenti le partite vengono considerate solo se appartenenti all'ultima versione di gioco. Ogni giocatore può inoltre visualizzare tale suggerimento considerando tutte le partite presenti nel database oppure solo le partite che ha giocato lui stesso.

Gli utenti possono inserire i materiali presenti nel loro inventario e richiedere una lista delle licenze acquistabili con i materiali in loro possesso con descrizione dell'effetto delle birre. Si assume che ogni utente abbia un unico profilo associato.

CONSIDERAZIONI SUI VOLUMI E OPERAZIONI DI INTERESSE

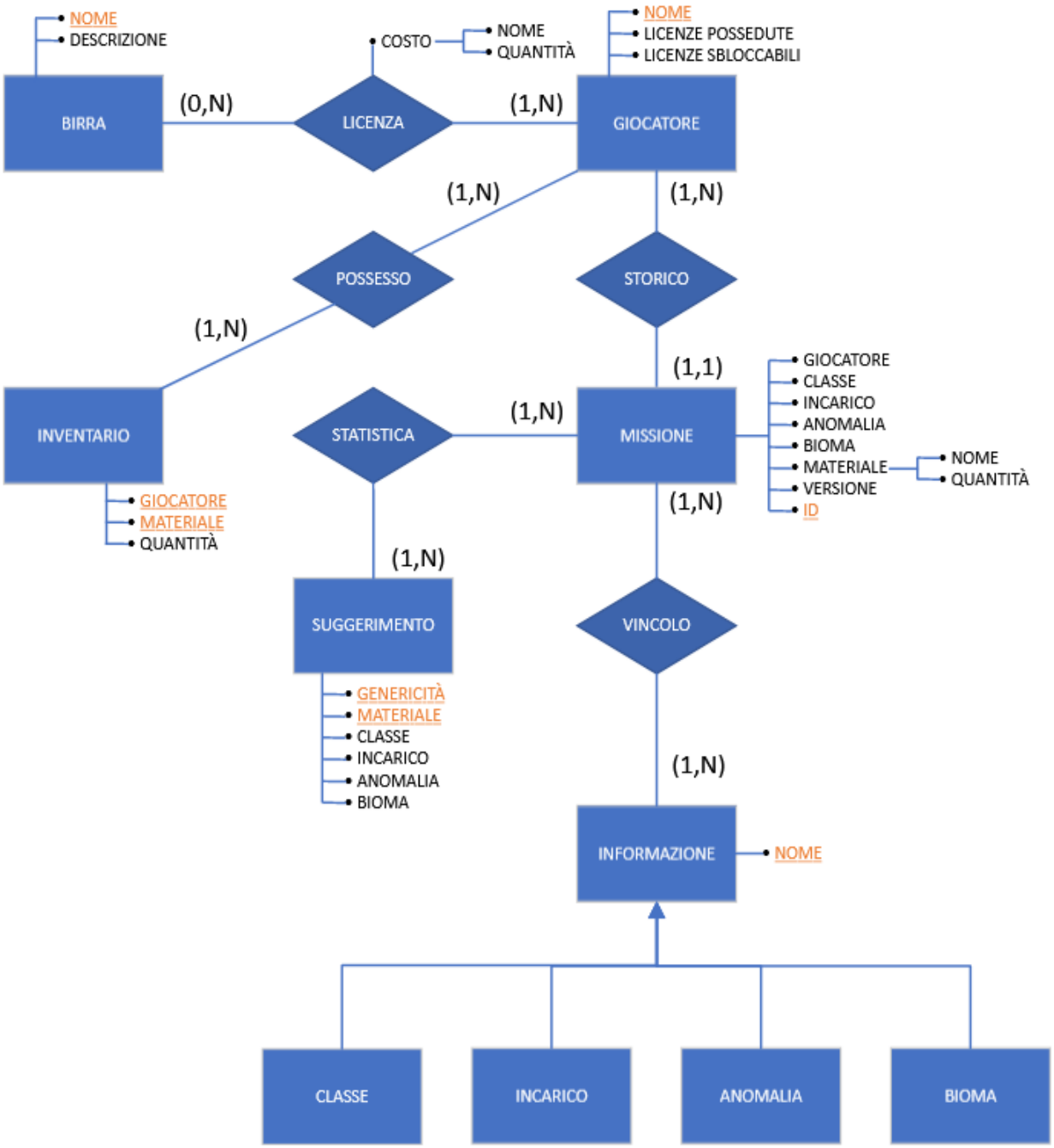
Si decide di permettere l'accesso al database in via sperimentale a non più di 500 giocatori. Si suppone che questi utenti giochino in media un'ora al giorno, quindi che ognuno di essi possa:

- inserire due partite al giorno nell'entità Missioni;
- richiedere una lista delle birre non ancora sbloccate con relativi costi e descrizioni una volta ogni settimana;
- fare otto richieste di statistiche per i materiali ogni settimana;
- aggiornare una volta al giorno il proprio inventario;
- modificare una volta alla settimana la lista delle licenze che possiede;
- richiedere una lista delle licenze sbloccabili con i propri materiali;

Si vuole inoltre fornire la possibilità agli utenti di inserire personalmente il proprio profilo nel database.

Operazione	Tipo	Frequenza
Inserimento di un nuovo utente	Interattiva	Trascurabile rispetto alle altre operazioni
Inserimento delle missioni completate con relative informazioni	Interattiva	1000 / giorno
Aggiornamento dei suggerimenti	Batch	8016 / settimana
Richiesta licenze bloccate con relativi costi e descrizioni	Interattiva	500 / settimana
Richiesta di consigli per i materiali	Interattiva	4000 / settimana
Richiesta licenze sbloccabili	Interattiva	500 / settimana
Aggiornamento dell'inventario	Interattiva	500 / giorno
Aggiornamento delle licenze possedute	Interattiva	500 / settimana
Aggiornamento delle licenze sbloccabili	Batch	4000 / settimana

SCHEMA CONCETTUALE E-R



DIZIONARIO DEI DATI: ENTITÀ

Entità	Descrizione	Attributi	Identificatore
Birra	Oggetto acquistabile	Nome, Descrizione	Nome
Giocatore	Utente della piattaforma	Nome, Licenze Possedute, Licenze Sbloccabili	Nome
Inventario	Materiali posseduti dai giocatori	Giocatore, Materiale, Quantità	Giocatore, Materiale
Missione	Elenco di tutte le partite d+ei giocatori	Giocatore, Classe, Incarico, Anomalia, Bioma, Materiale, Versione, ID	ID
Suggerimento	Entità che immagazzina i metodi per ottenere le statistiche di interesse	Genericità, Materiale, Classe, Incarico, Anomalia, Bioma	Genericità, Materiale
Informazione	Vincoli per i campi delle altre tabelle	Nome	Nome
Classe	Elenco di tutte le classi presenti	Nome	Nome
Incarico	Elenco di tutti gli incarichi possibili	Nome	Nome
Anomalia	Elenco delle anomalie disponibili	Nome	Nome
Bioma	Elenco dei biomi esistenti	Nome	Nome

DIZIONARIO DEI DATI: RELAZIONI

Relazione	Descrizione	Componenti	Attributi
Licenza	Associa agli utenti le birre in loro possesso	Giocatore, Birra	Costo
Possesso	Associa agli utenti i materiali in loro possesso	Giocatore, Inventario	
Storico	Lega i giocatori alle partite da loro giocate	Giocatore, Missione	
Statistica	Fornisce ai consigli i dati presenti nelle missioni inserite dagli utenti	Suggerimento, Missione	
Vincolo	Vincola i dati in 'Missione' alle informazioni corrette del gioco	Informazione, Missione	

VINCOLI NON ESPRIMIBILI GRAFICAMENTE

Non possono esistere più giocatori con lo stesso nome.

Quando viene modificato l'inventario di un giocatore bisogna ricalcolare le licenze sbloccabili.

Quando un giocatore acquisisce una licenza vanno modificate le licenze sbloccabili e quelle sbloccate.

CONSIDERAZIONI GENERALI

Le entità Informazione e Birra contengono dati che non cambiano nel tempo, relativi a informazioni del gioco.

L'entità Informazione contiene, per ogni caratteristica di interesse per l'applicazione contenuta nel gioco, un elenco di tutti i possibili valori che quei campi possono assumere. Questo serve per vincolare i rispettivi campi presenti nelle altre tabelle, in particolare quelli che possono essere modificati dall'utente.

Si gestisce il caso in cui il campo "Anomalia" sia vuoto con la stringa vuota '', preferendo non inserire valori nulli.

L'entità Suggerimento utilizza le informazioni inserite dagli utenti nella tabella Missioni e consiglia le combinazioni migliori per ottenere ogni dato materiale. Il suggerimento può essere generale, utilizzando quindi tutti i dati presenti nel database, oppure vincolato alle partite dei singoli giocatori. In entrambi i casi si considerano solo le partite registrate con l'ultima versione del gioco. È sufficiente aggiornare queste statistiche settimanalmente.

Si osserva che la generalizzazione con entità padre l'entità Informazione è di tipo totale ed esclusiva poiché, nei riguardi della mia applicazione, tutte le informazioni di interesse del gioco sono rappresentate come entità figlie e sono ben distinte.

ANALISI DELLE RIDONDANZE

Osservata l'assenza di cicli, si pone attenzione agli attributi derivabili da altre entità.

Si osserva che l'attributo "Licenze Sbloccabili" dell'entità Giocatore è ridondante, poiché ricavabile dagli attributi "Licenze Possedute" di Giocatore e "Nome" appartenente all'entità Birre.

L’operazione di aggiornamento delle licenze sbloccabili modifica le licenze sbloccabili da un personaggio ogniqualvolta quest’ultimo modifica i materiali nel suo inventario (500 operazioni totali al giorno) o modifica le licenze sbloccate (500 operazioni alla settimana). Poiché non è necessario salvare questa informazione in memoria e confrontando la quantità di operazioni effettuate tra la soluzione con ridondanza (7*500/settimana + 500/settimana) e la soluzione senza ridondanza (4000/settimana) si decide di eliminare l’attributo dall’entità Giocatore.

Come conseguenza i vincoli non esprimibili legati all’attributo “Licenze Sbloccabili” non vengono più considerati.

Dei campi presenti nella tabella Suggerimento, quattro sono il risultato di un calcolo settimanale che elabora i dati presenti nella tabella Missioni (4 * 2004 / settimana). Considerando le operazioni settimanali di richiesta per i suggerimenti, generali e personali (4000 / settimana) si decide di mantenere l’entità invariata, calcolando un numero definito di volte i consigli e limitandosi a operazioni di lettura per fornire queste informazioni in maniera rapida ai giocatori.

ELIMIMINAZIONE DELLE GENERALIZZAZIONI

La generalizzazione dell’entità Informazione è stata inserita perché serve considerare separatamente i figli, i quali hanno dimensioni diverse tra di loro e non devono contenere valori nulli. Si decide quindi di accorpare gli attributi del genitore nei figli.

Come conseguenza ognuno dei figli entra in relazione con l’entità Missioni.

PARTIZIONAMENTO DI ENTITÀ E RELAZIONI

Gli attributi multivalore presenti nello schema contengono i quattro tipi di materiale e le rispettive quantità. Per non rallentare le operazioni di lettura, si decide di spezzarli in quattro attributi, ognuno rappresentante uno dei quattro materiali di interesse (abbreviati con BB, YC, MS, SN) e contenenti le quantità relative.

Considerando che le letture sull'entità Missione vengono filtrate attraverso versione di gioco e giocatori, si decide di separare Missione in due entità, la prima contenente ID missione, versione e giocatore, la seconda contenente le informazioni sulle partite e l'ID relativo alle singole missioni.

ACCORPAMENTO DI ENTITÀ E RELAZIONI

Si decide di accorpare gli attributi della relazione Licenza all'interno dell'entità birra, con la stessa separazione dei materiali descritta nella sezione precedente.

L'entità Inventario viene incorporata nell'entità giocatore, poiché nel tenerle separate rallento gli accessi ai materiali posseduti dai giocatori senza ottenere benefici significativi.

Tenendo in considerazione il modello relazionale al quale ci si sta avvicinando, si decide di assegnare l'attributo "Possesso" alla relazione Licenza, il quale sostituirà dell'attributo "Licenze Possedute" dell'entità Giocatore.

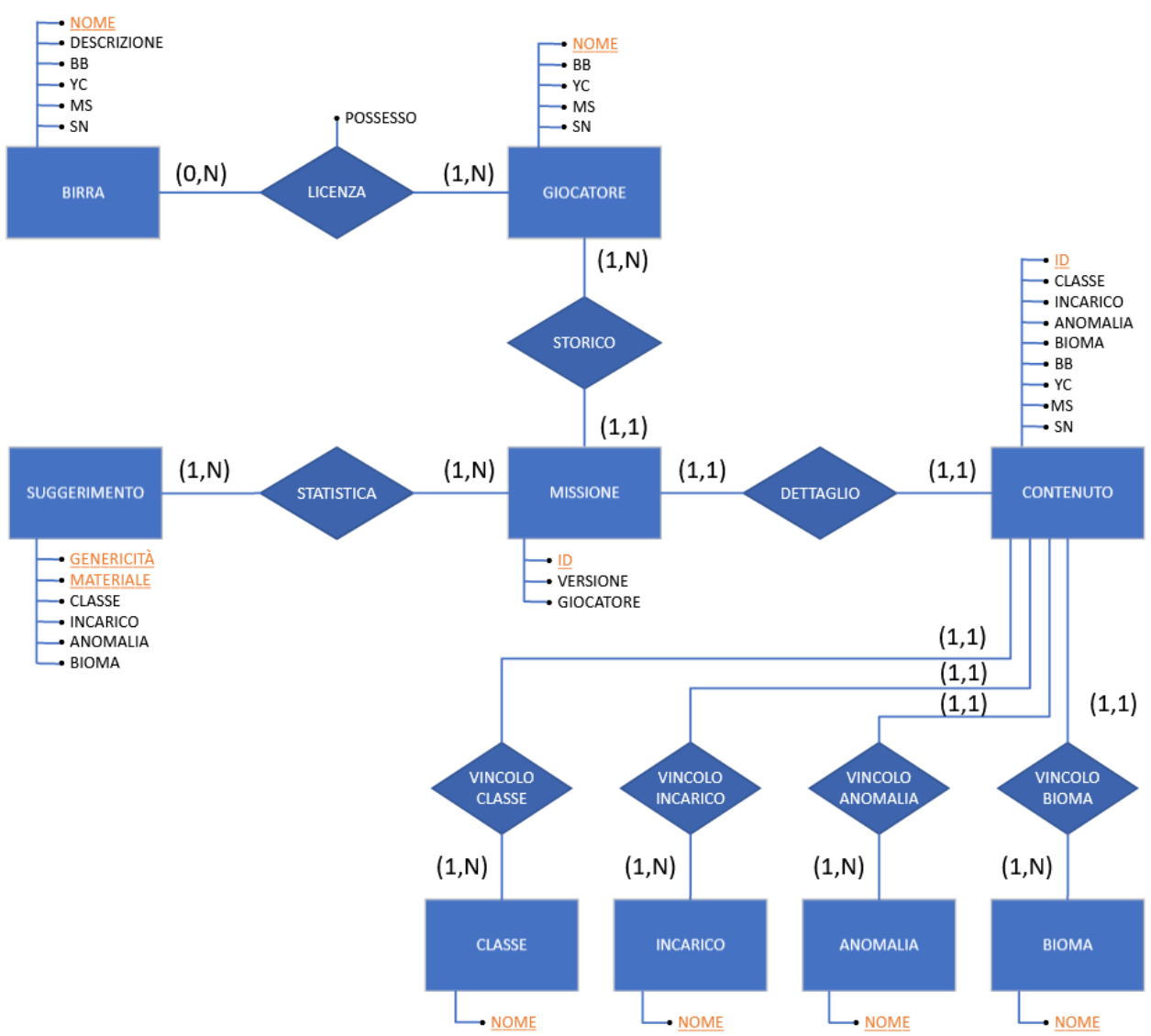
SCELTA DEGLI IDENTIFICATORI PRIMARI

Nello schema sono evidenziati gli attributi scelti come identificatori primari, riportati di seguito per chiarezza:

- Birra: Nome
- Giocatore: Nome
- Missione: ID
- Contenuto: ID
- Suggerimento: Genericità, Materiale
- Classe: Nome

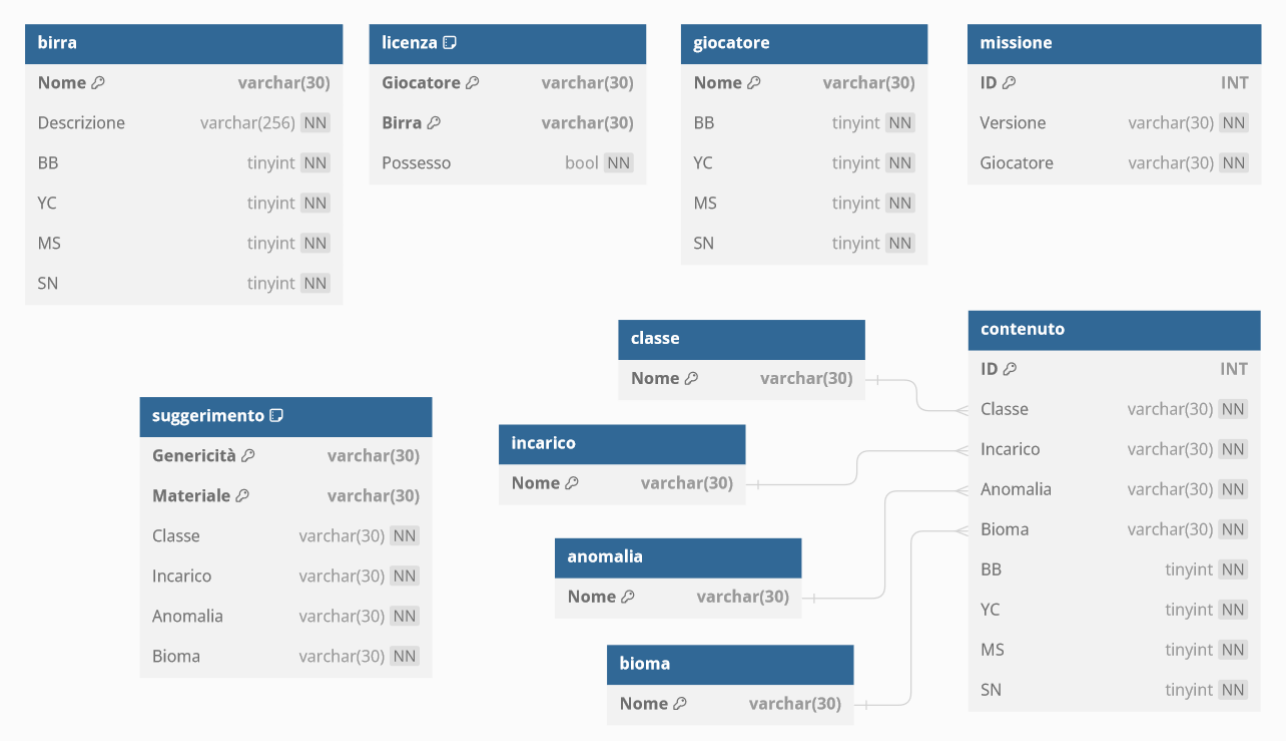
- Incarico: Nome
- Anomalia: Nome
- Bioma: Nome

DIAGRAMMA E-R RISTRUTTURATO



SCHEMA LOGICO

- Birra: Nome, Descrizione, BB, YC, MS, SN
- Licenza: Giocatore, Birra, Possesso
- Giocatore: Nome, BB, YC, MS, SN
- Missione: ID, Versione, Giocatore
- Contenuto: ID, Classe, Incarico, Anomalia, Bioma, BB, YC, MS, SN
- Suggerimento: Genericità, Materiale, Classe, Incarico, Anomalia, Bioma
- Classe: Nome
- Incarico: Nome
- Anomalia: Nome
- Bioma: Nome



NORMALIZZAZIONE

Il database è in prima forma normale, poiché tutti i campi sono atomici e non possono contenere più di un dato.

Il database è anche in seconda forma normale, poiché ogni colonna dipende dalla chiave primaria.

La tabella Birra non è in terza forma normale, dato che i prezzi delle licenze possono essere ricondotti alle birre (chiave primaria) attraverso la descrizione delle stesse. Si decide di mantenerla in seconda forma normale perché aggiungerebbe complessità nell'accorpare nome, descrizione e costo delle birre nelle operazioni descritte in precedenza.

Nella tabella suggerimenti, i quattro campi Classe, Incarico, Anomalia, Bioma vengono riempiti attraverso un'operazione che viene svolta una volta alla settimana. Dato che non cambiano ogni volta che i campi su cui sono calcolati vengono modificati ma lo fanno con cadenza ben definita, non impediscono alla tabella Suggerimento di essere considerata in terza forma normale.

Le tabelle non esaminate in precedenza rispettano tutte la terza forma normale.

CREAZIONE DEL DATABASE

Nella creazione del database si decide di inserire preventivamente nella tabella Suggerimento le quattro entry relative ai consigli generici, poiché i successivi inserimenti vengono gestiti durante l'inserimento dei singoli giocatori. Si presta attenzione alla distinzione tra i consigli generali e quelli dei singoli giocatori, effettuata definendo il campo genericità dei quattro consigli generici come "0", mentre per ogni giocatore questo campo presenterà il nome del giocatore preceduto dal carattere "." .

La creazione delle restanti tabelle è relativamente semplice, pertanto verrà omessa.

Come si può notare dallo schema logico, la presenza di vincoli di integrità referenziale è ridotta, questo perché gli altri legami tra tabelle vengono gestiti all'interno delle operazioni descritte in seguito.

Nella creazione del database si decide di inserire i quattro campi per consigli generici nella tabella Suggerimento e si popolano le tabelle Classe, Incarico, Anomalia, Bioma, Birra, dato che contengono informazioni di gioco che rimarranno invariate durante l'utilizzo del database.

REALIZZAZIONE DELLE OPERAZIONI

Come prima cosa genero una procedura che tornerà utile per rendere più leggibile l'aggiornamento della tabella contenente i suggerimenti; non controllo l'input perché mi assicuro l'integrità nella funzione per l'aggiornamento dei suggerimenti

```
-- per comodità, definisco una funzione che prende un materiale e la genericità e aggiorna il suggerimento relativo

DELIMITER //

CREATE PROCEDURE CalcoloSuggerimento (IN TargetMat VARCHAR(30), IN TargetGen VARCHAR(30))

BEGIN

   DECLARE Ver VARCHAR(30);

-- ricavo l'ultima verisone di gioco

   SELECT versione INTO Ver FROM missione ORDER BY ID DESC LIMIT 1;


   -- costruisco un prepared statement filtrando per genericità richiesta; in caso non ci fossero dati da utilizzare ritorno un suggerimento qualsiasi

   IF STRCMP(TargetGen, '0') = 0 THEN

      -- calcolo i suggerimenti per i quattro campi generali

      SET @Query1 = CONCAT("SELECT IFNULL((SELECT LOWER(c.classe) FROM missione m INNER JOIN contenuto c ON m.ID = c.ID WHERE m.versione = '", Ver, "' GROUP BY LOWER(c.classe) ORDER BY SUM(", TargetMat, ")/COUNT(", TargetMat, ") LIMIT 1), 'Driller')");

      SET @Query2 = CONCAT("SELECT IFNULL((SELECT LOWER(c.incarico) FROM missione m INNER JOIN contenuto c ON m.ID = c.ID WHERE m.versione = '", Ver, "' GROUP BY LOWER(c.incarico) ORDER BY SUM(", TargetMat, ")/COUNT(", TargetMat, ") LIMIT 1), 'Point Extraction')");

      SET @Query3 = CONCAT("SELECT IFNULL((SELECT LOWER(c.anomalia) FROM missione m INNER JOIN contenuto c ON m.ID = c.ID WHERE m.versione = '", Ver, "' GROUP BY LOWER(c.anomalia) ORDER BY SUM(", TargetMat, ")/COUNT(", TargetMat, ") LIMIT 1), 'Critical Weakness')");

      SET @Query4 = CONCAT("SELECT IFNULL((SELECT LOWER(c.bioma) FROM missione m INNER JOIN contenuto c ON m.ID = c.ID WHERE m.versione = '", Ver, "' GROUP BY LOWER(c.bioma) ORDER BY SUM(", TargetMat, ")/COUNT(", TargetMat, ") LIMIT 1), 'Crystalline Caverns')");

   ELSE

      -- calcolo i suggerimenti per i campi specifici

      SET @Query1 = CONCAT("SELECT IFNULL((SELECT LOWER(c.classe) FROM missione m INNER JOIN contenuto c ON m.ID = c.ID WHERE m.versione = '", Ver, "' AND m.giocatore = '", TargetGen, "' GROUP BY LOWER(c.classe) ORDER BY SUM(", TargetMat, ")/COUNT(", TargetMat, ") LIMIT 1), 'Driller')");
```

```
    SET @Query2 = CONCAT("SELECT IFNULL((SELECT LOWER(c.incarico) FROM missione m INNER JOIN contenuto c ON m.ID = c.ID WHERE m.versione = '", Ver, "' AND m.giocatore = '", TargetGen, "' GROUP BY LOWER(c.incarico) ORDER BY SUM(", TargetMat, ")/COUNT(", TargetMat, ") LIMIT 1), 'Point Extraction')");

    SET @Query3 = CONCAT("SELECT IFNULL((SELECT LOWER(c.anomalia) FROM missione m INNER JOIN contenuto c ON m.ID = c.ID WHERE m.versione = '", Ver, "' AND m.giocatore = '", TargetGen, "' GROUP BY LOWER(c.anomalia) ORDER BY SUM(", TargetMat, ")/COUNT(", TargetMat, ") LIMIT 1), 'Critical Weakness')");

    SET @Query4 = CONCAT("SELECT IFNULL((SELECT LOWER(c.bioma) FROM missione m INNER JOIN contenuto c ON m.ID = c.ID WHERE m.versione = '", Ver, "' AND m.giocatore = '", TargetGen, "' GROUP BY LOWER(c.bioma) ORDER BY SUM(", TargetMat, ")/COUNT(", TargetMat, ") LIMIT 1), 'Crystalline Caverns')");

  END IF;


  -- compongo ed eseguo le query

  SET @Query1 = CONCAT("UPDATE suggerimento SET classe = (", @Query1, ") WHERE Genericità = '", TargetGen, "' AND Materiale = '", TargetMat, "'");

  SET @Query2 = CONCAT("UPDATE suggerimento SET incarico = (", @Query2, ") WHERE Genericità = '", TargetGen, "' AND Materiale = '", TargetMat, "'");

  SET @Query3 = CONCAT("UPDATE suggerimento SET anomalia = (", @Query3, ") WHERE Genericità = '", TargetGen, "' AND Materiale = '", TargetMat, "'");

  SET @Query4 = CONCAT("UPDATE suggerimento SET bioma = (", @Query4, ") WHERE Genericità = '", TargetGen, "' AND Materiale = '", TargetMat, "'");

  PREPARE stmt1 FROM @Query1;

  PREPARE stmt2 FROM @Query2;

  PREPARE stmt3 FROM @Query3;

  PREPARE stmt4 FROM @Query4;

  EXECUTE stmt1;

  EXECUTE stmt2;

  EXECUTE stmt3;

  EXECUTE stmt4;

  DEALLOCATE PREPARE stmt1;

  DEALLOCATE PREPARE stmt2;

  DEALLOCATE PREPARE stmt3;

  DEALLOCATE PREPARE stmt4;

END //

DELIMITER ;
```

Stored procedure che prende in input il nome dell'utente, controlla che non sia già presente nel database e lo inserisce, preparando i campi relativi nelle tabelle giocatore, licenza, suggerimento

```
-- creazione di un utente (interattiva) e controllo unicità del nome
DELIMITER //
CREATE PROCEDURE InserimentoGiocatore (IN NomeUtente VARCHAR(30))
BEGIN
   DECLARE unicità INT DEFAULT 0;
   -- controllo l'unicità del nome
   SELECT COUNT(*) INTO unicità FROM giocatore g WHERE NomeUtente = g.Nome;

   -- se il nome esiste termino la procedura
   IF unicità > 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Nome già utilizzato';
          END IF;

   START TRANSACTION;
   -- modifico la tabella giocatore
      INSERT INTO giocatore VALUES (NomeUtente, 0, 0, 0, 0);

   -- modifico la tabella licenza
      INSERT INTO licenza SELECT NomeUtente, b.Nome, FALSE FROM birra b;

          -- ora modifico la tabella suggerimento, inserendo temporaneamente suggerimenti qualsiasi in attesa di avere dati da elaborare
   -- inserisco il punto prima del nome per garantire il riconoscimento del campo generico
      INSERT INTO suggerimento Values (CONCAT('.',NomeUtente),'BB','Driller','Mining Expedition','Low Oxigen','Azure Weald');
      INSERT INTO suggerimento Values (CONCAT('.',NomeUtente),'YC','Engineer','On-Site Refining','Litophage Outbreak','Fungus Bogs');
      INSERT INTO suggerimento Values (CONCAT('.',NomeUtente),'MS','Gunner','Salvage Operation','Lethal Enemies','Salt Pits');
```

```
    INSERT INTO suggerimento Values (CONCAT('.',NomeUtente),'SN','Scout','Industrial Sabotage','Low Gravity','Magma Core');
        COMMIT;
END //
DELIMITER ;
```

Procedura che permette all'utente di inserire le informazioni relative a una missione che ha giocato nel database

```
-- inserire una nuova missione (interattiva)
DELIMITER //
CREATE PROCEDURE InserimentoPartita (IN Versione VARCHAR(30), IN Giocatore VARCHAR(30), IN Classe VARCHAR(30), IN Incarico VARCHAR(30),
                    IN Anomalia VARCHAR(30), IN Bioma VARCHAR(30),IN BB TINYINT, IN YC TINYINT, IN MS TINYINT, IN SN TINYINT)
BEGIN
  DECLARE missioneID INT;
  DECLARE EsistenzaUsr INT DEFAULT 0;
  DECLARE InClasse varchar(30);
  DECLARE InIncarico varchar(30);
  DECLARE InAnomalia varchar(30);
  DECLARE InBioma varchar(30);

  -- controllo l'esistenza dell'utente e dei campi inseriti
  SELECT COUNT(*) INTO EsistenzaUsr FROM giocatore g WHERE STRCMP(LOWER(Giocatore), LOWER(g.Nome)) = 0;
      SELECT COUNT(*) INTO InClasse FROM classe c WHERE STRCMP(LOWER(c.nome),LOWER(Classe)) = 0;
  SELECT COUNT(*) INTO InIncarico FROM incarico i WHERE STRCMP(LOWER(i.nome),LOWER(Incarico)) = 0;
  SELECT COUNT(*) INTO InAnomalia FROM anomalia a WHERE STRCMP(LOWER(a.nome),LOWER(Anomalia)) = 0;
  SELECT COUNT(*) INTO InBioma FROM bioma b WHERE STRCMP(LOWER(b.nome),LOWER(Bioma)) = 0;
```

```
    -- se uno di questi campi non esiste termino la procedura
    IF EsistenzaUsr = 0 OR InClasse = 0 OR InIncarico = 0 OR InAnomalia = 0 OR InBioma = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Errore nell'inserimento della partita";
    END IF;

-- per inserire la entry nelle due tabelle relative, uso una transazione
    START TRANSACTION;
        INSERT INTO missione(Versione, Giocatore) VALUES(Versione, Giocatore);
        -- mi salvo l'ultimo ID inserito e lo uso nel prossimo inserimento
        SET missioneID = LAST_INSERT_ID();
        INSERT INTO contenuto VALUES(missioneID, Classe, Incarico, Anomalia, Bioma, BB, YC, MS, SN);
    COMMIT;
END //
DELIMITER ;
```

Procedura utilizzata per aggiornare tutti i valori della tabella suggerimento

```
-- aggiornamento settimanale dei suggerimenti (batch)
DELIMITER //
CREATE PROCEDURE AggiornamentoSuggerimenti ()
BEGIN
    -- voglio ciclare su tutte le righe della tabella suggerimento, lo faccio con un cursore
    DECLARE flag INT DEFAULT FALSE;
    DECLARE TargetGen VARCHAR(30);
    DECLARE TargetMat VARCHAR(30);

    -- dichiaro il cursore
    DECLARE CursoreGiocatori CURSOR FOR SELECT Genericità, Materiale FROM suggerimento;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET flag = TRUE;
    OPEN CursoreGiocatori;
```

```
    -- ciclo sulla chiave primaria della tabella suggerimento e aggiorno i valori
    read_loop: LOOP
        FETCH CursoreGiocatori INTO TargetGen, TargetMat;
        IF flag THEN
            LEAVE read_loop;
        END IF;
        CALL CalcoloSuggerimento(TargetMat, TargetGen);
    END LOOP;

    -- chiudo il cursore
    CLOSE CursoreGiocatori;
END //
DELIMITER ;
```

Procedura che restituisce i suggerimenti (classe, incarico, anomalia, bioma) per un determinato giocatore e per un determinato materiale

```
-- richieste di consigli per i materiali (interattiva)
ELIMITER //
CREATE PROCEDURE Consiglio (IN GenRichiesta VARCHAR (30), IN MatRichiesto VARCHAR(30))
BEGIN
    SELECT s.Classe, s.Incarico, s.Anomalia, s.Bioma FROM suggerimento s WHERE s.Genericità =
GenRichiesta AND s.Materiale = MatRichiesto;
END//
DELIMITER ;
```

Procedura che permette all'utente di modificare il contenuto del proprio inventario, inserendo le quantità dei materiali in suo possesso

```
-- aggiornamento inventario (interattiva)
DELIMITER //
CREATE PROCEDURE AggiuntaLicenza (IN NomeGiocatore VARCHAR(30), IN NomeBirra VARCHAR(30))
BEGIN
   UPDATE licenza l SET Possesso = TRUE WHERE l.Birra = NomeBirra AND l.giocatore = NomeGiocatore;
END //
DELIMITER ;
```

Procedura che permette all'utente di aggiungere al database le licenze che sblocca in gioco

```
-- aggiornamento licenze possedute (interattiva)
DELIMITER //
CREATE PROCEDURE AggiuntaLicenza (IN NomeGiocatore VARCHAR(30), IN NomeBirra VARCHAR(30))
BEGIN
   UPDATE licenza l SET Possesso = TRUE WHERE l.Birra = NomeBirra AND l.giocatore = NomeGiocatore;
END //
DELIMITER ;
```

Procedura che mostra all'utente le licenze che può sbloccare con i materiali che ha nell'inventario

```
-- richiesta licenze sbloccabili (interattiva)
DELIMITER //
CREATE PROCEDURE LicenzeSbloccabili (IN NomeGiocatore VARCHAR(30))
BEGIN
   SELECT l.Birra, b.Descrizione, b.BB, b.YC, b.MS, b.SN  FROM licenza l INNER JOIN birra b ON l.Birra = b.Nome INNER JOIN giocatore g ON l.Giocatore = g.Nome WHERE l.Giocatore = NomeGiocatore AND b.BB <= g.BB AND b.YC <= g.YC AND b.MS <= g.MS AND b.SN <= g.SN AND l.possesso = FALSE;
END //
```

```
DELIMITER ;
```

Procedura che restituisce all'utente un elenco di birre di cui non ha sbloccato la licenza con relativi descrizione e costo

```
-- richiesta licenze da sbloccare e relativi costi (interattiva)

DELIMITER //

CREATE PROCEDURE LicenzeBloccate (IN NomeUtente VARCHAR(30))

BEGIN

  SELECT l.Birra, b.Descrizione, b.BB, b.YC, b.MS, b.SN FROM licenza l INNER JOIN birra b ON l.Birra = b.Nome WHERE l.Giocatore = NomeUtente AND l.possesso = FALSE;

END //

DELIMITER ;
```