Relazione Progetto Basi di Dati

Federico Omodei, Luca Violato

Nome gruppo: Survivors

A.A. 2016/2017

Abstract

“Dynasty Crysis” e’ un nuovo videogioco fantasy MMORPG (Massive Multiplayer Online Role-Playing Game), rilasciato di recente, che comprende un vasto assortimento di tutti gli elementi caratteristici del genere: La mappa di gioco e’ molto ampia, suddivisa in Regioni, a loro volta suddivise in località. Sono presenti numerose città visitabili in cui è possibile interagire con altri giocatori. In molte aree è possibile imbattersi in diversi Mostri, ciascuno con le proprie caratteristiche, dal più debole al più ostico. I giocatori scelgono il proprio Eroe, modellandolo a loro piacimento e decidendone la classe, i giocatori possono anche unirsi in squadre formate da un massimo di 6 membri. Per sviluppare la meglio il proprio personaggio i giocatori dovranno cacciare i numerosi mostri che popolano il mondo di gioco, allo scopo di ottenere gli Item che tali mostri lasciano cadere (da qui in avanti “droppano”). Tali Item sono numerosi e di diversa utilità, possono essere armi oppure oggetti indossabili, come anche oggetti di carattere generico come oggetti curativi, libri, oggetti commemorativi e molti altri. I giocatori possono inoltre commerciare con i diversi NPC (Non Playable Character) mercanti presenti in alcune città allo scopo di comprare gli Item che non hanno ancora ottenuto. Non ci sono limiti al denaro che può accumulare un giocatore. I giocatori più impavidi potranno misurarsi con Mostri particolarmente impegnativi come Boss o addirittura Mostri Leggendari allo scopo di ottenere Armi Uniche, ovvero armi di cui esiste una sola copia droppabili solamente dai Boss, oppure l’ambito titolo di GodSlayer ottenibile attraverso l’uccisione di un Mostro Leggendario.

Allo scopo di gestire questo vasto universo di gioco, e per semplificare la vita ai nuovi giocatori che volessero approcciarvisi (o per I giocatori veterani in cerca di informazioni approfondite), si è deciso di creare una base di dati nota come DCG (“Dynasty Crysis Guide”). Gli scopi fondamentali della DCG sono, ad esempio, quello di rendere facile e comprensibile agli utenti la ricerca di ogni Item di gioco e del particolare mostro che lo droppa o il mercante/i che lo vende; tenere traccia di tutte le Squadre di Eroi con i rispettivi leader, e rendere disponibile una ricerca basata sui particolari Mostri che si possono voler cacciare, fornendo quindi immediate informazioni sulla posizione del mostro.

Descrizione dei requisiti

Si vuole descrivere l'organizzazione dei dati di interesse dell'universo di un videogioco fantasy MMORPG.
Le specifiche di interesse sono le seguenti:

- Si vuole tenere traccia dei **Mostri**, di cui ci interessano: Nome (astratto della specie) e Codice (univoco per ciascun esemplare di ciascuna specie), Esperienza rilasciata all'uccisione.I mostri sono catalogabili a seconda delle loro caratteristiche: Mostri Terrestri, Volatili, Acquatici e Infernali. I Mostri possono anche essere suddivisi in base al loro rango: Mostri comuni, Rari, Semi-Boss, Boss, e Mostri Leggendari.
  Dei **Boss** e dei **Mostri** Leggendari in particolare si vuole sapere lo Stato (Vivo o Morto)
  Inoltre ogni mostro si trova in almeno una **Localita'**, e puo' o meno lasciar cadere (Droppare) una volta ucciso uno o più **Item** di interesse con relativi Drop Rate espressi in %.

- Delle **Località** ci interessa sapere: Nome (univoco all'interno del database) **Regione** di appartenenza, clima tipico. Ogni **Regione** ha almeno una Località ma può averne più di una, ogni Località può o meno avere al suo interno **Insediamenti Abitati**.
  Le Località possono essere suddivise in: **Ostili** e **Non Ostili**. Le località Non Ostili NON ospitano alcun Mostro. Le Località Ostili invece ospitano almeno un Mostro.

- Si vuole inoltre tener traccia delle **Aree Infernali**, particolari zone dove è possibile incontrare **Mostri Infernali**. Le Aree Infernali possono essere accedute solo attraverso le **Faglie**

- Le **Faglie** sono portali dimensionali che conducono alle cosiddette **Aree Infernali**, ogni Faglia conduce a una e una sola Area Infernale, tuttavia possono esistere più Faglie che conducono alla medesima Area Infernale. Delle Faglie ci interessa sapere la **Località** dove sono state avvistate, che se c'è' è unica.

- Gli **Insediamenti Abitati** sono zone in cui è possibile incontrare altri Eroi. Degli insediamenti abitati si vuole sapere: Nome (univoco), Numero Abitanti, Nome e Livello dei **Mercanti** residenti nell'Insediamento. Il Numero Abitanti non tiene conto degli Eroi creati dai giocatori, i quali non hanno una residenza. Il Numero Abitanti consiste invece nel numero di NPC (Non Playable Character) presenti nell'Insediamento Abitato.
  Gli Insediamenti Abitati possono inoltre essere suddivisi a seconda della grandezza in: Villaggi, Avamposti e Citta'.

- Degli **Item** droppati dai Mostri ci interessa sapere: Nome, Codice, Peso, Valore. Gli Item sono univocamente identificati dal loro Codice, inoltre possono essere suddivisi in **Armi, Componenti e Indossabili**, gli Item che non rientrano nelle precedenti categorie sono considerati **Item Generici**.
  Ogni Item inoltre viene classificato a seconda della sua rarità: Item Comuni, Item Rari, Item Leggendari.

  Delle **Armi** ci interessa sapere: Tipologia, Resistenza, Danno.
  Ci sono inoltre **Armi Uniche**, ovvero armi di cui esiste un unico esemplare, queste Armi Uniche sono droppate unicamente da Boss, ogni Arma unica e' Droppata da uno e un solo Boss, e ogni Boss droppa solo un'Arma Unica. Delle Armi Uniche ci interessa sapere il Boss che la può droppare.

- Degli **Eroi** ci interessa sapere: Nome (Univoco), Livello, Denaro. Per ogni Eroe e' registrato il Luogo di Nascita (in riferimento agli Insediamenti Abitati) e la data di nascita. Gli Eroi possono essere suddivisi per Categorie: Tank, Attacco, Difesa, Supporto, Jolly.
  Se un Eroe e' stato in grado di uccidere almeno un **Mostro Leggendario** acquisisce il titolo di **GodSlayer**. Per i GodSlayer ci interessa sapere il Mostro Leggendario ucciso e la data dell'uccisione per l'ottenimento del rango.

  Ogni Eroe ha a sua disposizione un **Inventario** contenente tutti e soli gli Item che gli appartengono. Ogni Inventario appartiene ad uno e un solo Eroe, e ha una Capienza massima che viene espressa in peso trasportabile.

  Gli Eroi possono o meno unirsi tra loro per formare **Squadre**.

- Delle **Squadre** ci interessa sapere il Nome (Univoco all'interno del Database) e tenerne uno storico, ogni Eroe può o meno far parte di una e una sola Squadra:
  Le squadre al momento attive sono dette **Squadre Attuali,** sono composte da uno e un solo Leader e da un numero di Eroi aggiuntivi variabile tra 0 e 5 Ogni **Squadra Attuale** dunque è composta da un numero totale di **Membri** variabile tra 1 e 6.
  Tra le **Squadre Attuali** ci sono inoltre le **Squadre Speciali**. Queste Squadre Speciali hanno la particolarità di essere le uniche Squadre ad avere come Leader un **GodSlayer**. Se un Leader di una Squadra Attuale ottiene il rango di GodSlayer dopo aver già formato la Squadra Attuale allora la sua Squadra Attuale viene promossa automaticamente a **Squadra Speciale.**

  Delle **Squadre Passate** è di primario e principale interesse il Nome della Squadra Passata. Nel caso in cui la Squadra Passata si sia sciolta per una decisione del Leader si vuole tenere conto anche della Data di Scioglimento.
  Vista la possibilità da parte dei giocatori di eliminare i propri Eroi sono ugualmente considerate **Squadre Passate** quelle Squadre il cui Eroe Leader sia stato eliminato. In questo caso dunque si conserva la sola informazione del Nome della **Squadra Passata** ed eventualmente le informazioni sui membri di tale Squadra Passata a meno che anch'essi non siano stati eliminati.
  Ogni **Squadra Passata** è dunque composta da un Leader, che viene indicato se ne è conservata l'informazione, e da un numero totale di **Membri** sempre inferiore o uguale a 6.

Politiche di ristrutturazione schema E-R

ITEM:

- Accorpamento delle entità figlie **Comune, Raro, Leggendario** nell'entità genitore **Item** con l'inserimento dell'attributo Rarità.
- Sostituzione della Generalizzazione tra **Item** e **Arma** con una associazione tra le medesime entità.
- Accorpamento dell'entità figlia **Componente** nell'entità padre **Item** con inserimento dell'attributo Tipologia.
- Accorpamento dell'entità figlia **Indossabile** nell'entità padre **Item** con inserimento dell'attributo Tipologia.
- **NB:** Attributo Tipologia distingue tra: **Arma, Componente, Indossabile e Generico.**

MOSTRO:

- Accorpamento delle entità figlie **Terrestre, Volante, Acquatico** nell'entità genitore **Mostro** con l'inserimento dell'attributo Tipologia.
- Sostituzione della Generalizzazione tra **Mostro** e **Mostro Infernale** con una associazione tra le medesime entità.
- Accorpamento delle entità figlie **Comune, Raro, Semi-Boss** nell'entità genitore **Mostro** con l'inserimento dell'attributo Rarità.
- Sostituzione della Generalizzazione tra **Mostro** e **Boss** con una associazione.
- Sostituzione della Generalizzazione tra **Mostro** e **Mostro Leggendario** con una associazione.

LOCALITÀ:

- Strategia Mista: Accorpamento dell'entità figlia **Località non Ostile** in **Località** e Sostituzione della Generalizzazione tra **Località** e **Località Ostile** con una associazione.
- **NB:** Nessun attributo richiesto per distinguere Località da Località non Ostile che vanno a coincidere.

INSEDIAMENTI ABITATI:

- Accorpamento delle entità figlie **Villaggio, Città, Avamposto** nell'entità genitore **Insediamenti Abitati** con l'inserimento dell'attributo Tipologia.

EROE:

- Accorpamento delle entità figlie **Tank, Attacco, Difesa, Jolly** nell'entità genitore **Eroe** con l'inserimento dell'attributo Classe.

SQUADRA:

- Accorpamento dell'entità genitore **Squadra** nelle entità figlie **Squadra Passata, Squadra Attuale.**
- Accorpamento dell'entità figlia **Squadra Speciale** nell'entità genitore **Squadra Attuale** con l'inserimento dell'attributo Speciale. (Accorpamento implicito dell'associazione LeaderS nell'entità Squadra Speciale).

Analisi di ridondanza

Inserimento di un attributo **Numero di Abitanti** in Località:
Si tratta di un attributo derivato, ottenibile da operazioni di conteggio dell'attributo Numero Abitanti dell' entità Insediamenti Abitati
Analizzando le Tabelle dei Volumi e Delle Operazioni possiamo fare una stima degli accessi necessari per le due operazioni rilevanti in questo caso nel caso di presenza e assenza del dato ridondante.

In caso di **Assenza del dato Ridondante:**

- Per l'Operazione 1 (Aggiornare il campo dati Numero Abitanti di tutti gli Insediamenti Abitati) sono semplicemente necessari 20 accessi in scrittura all'entità Insediamento Abitato.
- Per l'Operazione 2 (Stampare tutti i dati di una Località, incluso il numero di abitanti) sono necessari: 1 accesso in lettura all'entità Località per individuare la Località di interesse e i suoi dati direttamente deducibili, 1 accesso in media in lettura alla relazione Ubicazione, avendo mediamente ogni Località un solo Insediamento Abitato (Essendoci stimate 20 Località e 20 Insediamenti Abitati) e 1 accesso in lettura a Insediamento Abitato per ottenere l'informazione sul numero di abitanti

In caso di **Presenza del dato Ridondante:**

- Per l'Operazione 1 sono necessari 20 accessi in scrittura all'entità Insediamento Abitato, e, dovendo mantenere il dato aggiornato, ulteriori 20 accessi in lettura alla relazione Ubicazione, 20 accessi in lettura (per cercare la Località di interesse) e 20 in scrittura (per aggiornare il dato sul numero di abitanti) all'entità Località
- Per l'Operazione 2 è necessario invece un unico accesso in lettura all'entità Località

RICAPITOLANDO:
Essendo l'operazione 1 eseguita 2 volte al giorno e l'operazione 2 invece 3 volte al giorno e valutando come doppi gli accessi in scrittura abbiamo in media un totale di:

- **Senza attributo ridondante:** (20*2)*2 + (1+1+1)*3 = **89 accessi giornalieri**
- **Con attributo ridondante:** (20*2 + 20 +20 +20*2)*2 + 1*3 = **243 accessi giornalieri**

In conclusione gli accessi necessari per mantenere aggiornato un eventuale attributo ridondante risultano essere quasi il triplo degli accessi necessari per ottenere il dato indirettamente. Ciò significa che con questo particolare rapporto Insediamenti Abitati / Località e con queste Operazioni da eseguire non conviene l'inserimento di un attributo ridondante.

Mantenimento dell'associazione che lega **ArmaUnica** a **Boss:**
Si tratta di un'associazione derivabile dalla composizione delle associazioni: **Arma-ArmaUnica, Arma-Item, ItemDrop.**
Tuttavia, essendo le Armi Uniche e i Boss da cui ottenerle di particolare interesse in quanto, per l'appunto uniche, difficilmente ottenibili e altrettanto ricercate, un'eventuale rimozione dell'associazione che lega **ArmaUnica** a **Boss** comporterebbe la perdita dell'informazione riguardo al Boss da cui si è ottenuta nel caso in cui l'Arma Unica in questione appartenga a un Eroe.
Infatti per mantenere la consistenza delle informazioni nel database nel caso in cui uno specifico Item (univocamente riconosciuto dal suo Codice) sia aggiunto all'appartenenza di un Eroe allora se tale oggetto era ottenibile tramite Drop di un mostro esso deve essere eliminato dalla tabella ItemDrop. Altrimenti si verrebbe a creare una situazione in cui un medesimo Item sia contemporaneamente appartenente ad un Eroe e droppabile da un Mostro, cosa chiaramente impossibile.
Volendo dunque mantenere l'informazione per ogni Arma Unica (sia essa appartenente o meno a un Eroe) su quale sia, o sia stato, il Boss da cui ottenerla si è deciso di **mantenere la relazione in questione che lega ArmaUnica a Boss**.

CON RIDONDANZA			
OPERAZIONE 1			
CONCETTO	COSTR.	ACC.	TIPO
Insediamento A.	E	20	S
Ubicazione	R	20	L
Località	E	20	L
Località	E	20	S

OPERAZIONE 2			
CONCETTO	COSTR.	ACC.	TIPO
Insediamento A.	E	1	L

SENZA RIDONDANZA			
OPERAZIONE 1			
CONCETTO	COSTR.	ACC.	TIPO
Insediamento A.	E	20	S

OPERAZIONE 2			
CONCETTO	COSTR.	ACC.	TIPO
Insediamento A.	E	1	L
Ubicazione	R	1	L
Località	E	1	L

TAVOLE DELLE OPERAZIONI		
OPERAZIONE	TIPO	FREQUENZA
Op.1	I	2 al giorno
Op.2	I	3 al giorno

RELATIVA TAVOLA DEI VOLUMI		
CONCETTO	TIPO	VOLUME
Insediamento A.	E	20
Località	E	20
Ubicazione	R	20

Schema relazionale

Di seguito lo schema delle relazioni del database con le proprie chiavi primarie:

MOSTRO (Codice, Nome, Tipologia, Rarita, Esperienza)

ITEM DROP (Item, Mostro, DropRate)

HABITAT (Mostro, Localita)

ITEM (Codice, Nome, Tipo, Rarita, Peso, Valore**)**

INVENDITA (Mercante, LuogoNegozio, Item)

ARMA (Codice, Tipo, Danno, Resistenza)

ARMAUNICA (CodiceArma, DropBoss)

BOSS (Codice, Stato)

EROE (Nome, Denaro, Livello, Classe, LuogoNascita, DataNascita, Squadra)

SQUADRAPASSATA (Nome, DataScioglimento, Leader)

SQUADRAATTUALE (Nome, Leader, Speciale)

MEMBROPASSATO (Squadra, Eroe)

GODSLAYER(Nome, LeggendarioUcciso, DataUccisione)

MOSTROLEGGENDARIO (Codice, Stato**)**

MOSTROINFERNALE (Codice)

ORIGINE (MostroInfernale, Area**)**

AREAINFERNALE (Nome)

FAGLIA (Codice, Destinazione, Posizione)

LOCALITAOSTILE (Nome**)**

LOCALITA (Nome, Clima, Regione)

REGIONE (Nome)

INSEDIAMENTOABITATO (Nome, NumAbitanti, Tipologia, Localita)

MERCANTE (Nome, LuogoNegozio, Livello)

INVENTARIO (Proprietario, Capienza)

APPARTENENZA (Inventario, Item)

Query e procedure

QUERY 1) Visualizzare Nome, Livello e Classe degli Eroi “solitari” ovvero coloro che non fanno parte di una squadra e non hanno mai fatto parte di alcuna squadra in passato.

CODICE :

```
SELECT    Eroe.Nome as Nome, Eroe.Livello as Livello, Eroe.Classe as Classe
FROM      Eroe
WHERE     Eroe.Squadra is null and not exists (select * from MembroPassato where
                                    MembroPassato.Eroe=Eroe.Nome)
ORDER BY  Eroe.Livello desc, Eroe.Nome;
```

OUTPUT :

```
+--------+---------+--------+
| Nome   | Livello | Classe |
+--------+---------+--------+
| Sirius |      29 | Tank   |
+--------+---------+--------+
```

QUERY 2) Visualizzare il nome e il patrimonio totale delle Squadre Speciali. Il calcolo del patrimonio deve avvenire nel seguente modo: si somma il denaro totale di tutti i membri e a questo si aggiunge il valore di tutti gli Item in loro possesso.

CODICE :

```
SELECT    SquadraAttuale.Nome as NomeSquadra, sum(Eroe.Denaro)+sum(Item.Valore) as
          Patrimonio
FROM      Eroe JOIN SquadraAttuale ON Eroe.Squadra=SquadraAttuale.Nome
          JOIN Appartenenza ON Eroe.Nome=Appartenenza.Inventario
          JOIN Item ON Appartenenza.Item=Item.Codice
WHERE          SquadraAttuale.Speciale='S'
GROUP BY  SquadraAttuale.Nome;
```

OUTPUT :

```
+-----------------+------------+
| NomeSquadra     | Patrimonio |
+-----------------+------------+
| Globetrotters   |     185454 |
| Legion of Doom  |     164685 |
+-----------------+------------+
```

QUERY 3) Visualizzare il nome, il numero di Armi Uniche e l'eventuale squadra di appartenenza dell'eroe con il più alto numero di Armi Uniche in suo possesso ed ordinare i risultati in ordine decrescente in base al livello degli eroi.

CODICE :

```
SELECT    Eroe.Nome as Nome, Eroe.Squadra as Squadra, count(ArmaUnica.CodiceArma) as
          NumeroArmiUniche
FROM      Eroe JOIN Appartenenza ON Eroe.Nome=Appartenenza.Inventario
          JOIN Item ON Appartenenza.Item=Item.Codice
          JOIN ArmaUnica ON Appartenenza.Item=ArmaUnica.CodiceArma
          LEFT JOIN SquadraAttuale ON Eroe.Squadra=SquadraAttuale.Nome
GROUP BY  Eroe.Nome
HAVING    count(*) >= all (select count(*)
          from   Eroe join Appartenenza on Eroe.Nome=Appartenenza.Inventario
          join ArmaUnica on Appartenenza.Item=ArmaUnica.CodiceArma
          group by Eroe.Nome)
ORDER BY  Eroe.Livello desc;
```

OUTPUT :

```
+--------+----------------+------------------+
| Nome   | Squadra        | NumeroArmiUniche |
+--------+----------------+------------------+
| Tiw    | Legion of Doom |                1 |
| MrFoxy | Globetrotters  |                1 |
| Nihal  | Marines        |                1 |
+--------+----------------+------------------+
```

PROCEDURA 1) Una Procedura per l'acquisto da parte di un Eroe di un determinato Item in vendita presso un dato Mercante. La procedura richiede 4 parametri: il nome dell'Eroe che intende effettuare l'acquisto, il codice dell'Item in questione, Il nome e il Luogo del Negozio del Mercante. Nello specifico la Procedura ha cura di verificare se il denaro dell'Eroe indicato è sufficiente per effettuare l'acquisto, in caso affermativo inserisce nella tabella Appartenenza i dati dell'Eroe e dell'Item in questione, aggiorna il denaro dell'Eroe con il valore restante dopo l'acquisto ed elimina l'Item dalla tabella InVendita.

NB: La procedura sfrutta il trigger "ControllaAppartenenza" (descrizione nelle pagine successive, trigger n.3) per verificare che la capienza massima dell'inventario dell'eroe in questione non sia superata inserendo il nuovo oggetto. In caso contrario l'operazione viene annullata.

CODICE :

```
delimiter $$
DROP PROCEDURE IF EXISTS Acquisto $$
CREATE PROCEDURE Acquisto(E varchar(30), I char(8), Me varchar(30), L varchar(30))
BEGIN          declare Prezzo int;     declare Disponibilita int;
SELECT Valore into Prezzo
FROM Item
WHERE Codice=I;
SELECT Denaro into Disponibilita
from Eroe
where Nome=E;
IF Disponibilita >= Prezzo
        THEN  insert Appartenenza values (E, I);
        UPDATE Eroe SET Denaro=Disponibilita-Prezzo where Nome=E;
        DELETE FROM InVendita where Item=I and Mercante=Me and LuogoNegozio=L;
        ELSE  signal sqlstate '45000'  set message_text = 'Denaro insufficiente per acquistare';
END IF;     END; $$
delimiter ;
```

OUTPUT :

Prima di invocare la procedura:

```
+-------+--------+
| Nome  | Denaro |
+-------+--------+
| Nihal |   1086 |
+-------+--------+
```

```
+------------+----------+
| Inventario | Item     |
+------------+----------+
| Nihal      | AU000005 |
| Nihal      | IN000007 |
| Nihal      | IT000002 |
| Nihal      | IT000004 |
+------------+----------+
```

```
+------------+--------------+----------+
| Mercante   | LuogoNegozio | Item     |
+------------+--------------+----------+
| James Ford | Empire Bay   | AC000021 |
| James Ford | Empire Bay   | AC000039 |
| James Ford | Empire Bay   | IN000017 |
| James Ford | Empire Bay   | IT000026 |
+------------+--------------+----------+
```

Dopo la seguente chiamata: `call Acquisto('Nihal', 'IN000017', 'James Ford', 'Empire Bay');`

```
+-------+--------+
| Nome  | Denaro |
+-------+--------+
| Nihal |   1011 |
+-------+--------+
```

```
+------------+----------+
| Inventario | Item     |
+------------+----------+
| Nihal      | AU000005 |
| Nihal      | IN000007 |
| Nihal      | IN000017 |
| Nihal      | IT000002 |
| Nihal      | IT000004 |
+------------+----------+
```

```
+------------+--------------+----------+
| Mercante   | LuogoNegozio | Item     |
+------------+--------------+----------+
| James Ford | Empire Bay   | AC000021 |
| James Ford | Empire Bay   | AC000039 |
| James Ford | Empire Bay   | IT000026 |
+------------+--------------+----------+
```

PROCEDURA 2) Una Procedura che consenta la fusione di due Squadre Attuali. Nello specifico: Vengono indicati i nomi delle due Squadre da fondere e il nome che la nuova Squadra avrà. Viene eseguito un primo controllo per verificare che i membri complessivi delle due Squadre da fondere non superino quota 6, ovvero il massimo stabilito di membri per Squadra. Se ciò è verificato allora le due Squadre vengono sciolte e quindi inserite tra le Squadre Passate (A questo punto si attiva il trigger che si occupa di Eliminare le Squadre dalle Squadre Attuali e aggiungere i rispettivi membri in Membro Passato). Viene Creata la nuova Squadra con il Nome indicato come terzo parametro della procedura, il leader di tale Squadra sarà il Leader delle Squadre fuse avente livello maggiore e se la vecchia Squadra di tale leader era una Squadra Speciale allora anche la nuova Squadra sarà Speciale.

CODICE :

```
delimiter $$
DROP PROCEDURE IF EXISTS Fusione $$
CREATE PROCEDURE Fusione(S1 varchar(30), S2 varchar(30), NuovaSquadra varchar(30))
BEGIN
declare N1 smallint;	declare N2 smallint;	declare E varchar(30);
declare S char; declare L1 varchar(30); declare L2 varchar(30);
SELECT count(Nome) into N1
FROM Eroe
WHERE Squadra=S1;
SELECT count(Nome) into N2
FROM Eroe
WHERE Squadra=S2;
IF N1 + N2 <= 6
	then
	SELECT Eroe.Nome, SquadraAttuale.Speciale into E, S
	FROM Eroe join SquadraAttuale on SquadraAttuale.Leader=Eroe.Nome
	WHERE SquadraAttuale.Nome=S1 or SquadraAttuale.Nome=S2
	ORDER BY Eroe.Livello desc limit 1;
	SELECT Leader into L1
	FROM SquadraAttuale
	WHERE Nome=S1;
	SELECT Leader into L2
	FROM SquadraAttuale
	WHERE Nome=S2;
	insert SquadraPassata values (S1, NULL, L1);
	insert SquadraPassata values (S2, NULL, L2);
```

```
insert SquadraAttuale values (NuovaSquadra, E, S);
update Eroe set Squadra=NuovaSquadra where Squadra is null and Nome in (select Eroe from
MembroPassato where Squadra=S1 or Squadra=S2);
ELSE signal sqlstate '45000' set message_text = 'Numero Massimo Membri Superato';
END IF; END; $$
delimiter ;
```

OUTPUT :

Prima della chiamata:

```
+-----------+----------------+
| Nome      | Squadra        |
+-----------+----------------+
| Sirius    | NULL           |
| Morpheus  | Globetrotters  |
| MrFoxy    | Globetrotters  |
| Mystica   | Globetrotters  |
| Trinity   | Globetrotters  |
| Albus     | Legion of Doom |
| Eragon    | Legion of Doom |
| Tiw       | Legion of Doom |
| Aldebaran | Marines        |
| Nihal     | Marines        |
+-----------+----------------+
```

```
+-----------------+--------+----------+
| Nome            | Leader | Speciale |
+-----------------+--------+----------+
| Globetrotters   | MrFoxy | S        |
| Legion of Doom  | Tiw    | S        |
| Marines         | Nihal  | N        |
+-----------------+--------+----------+
```

Dopo la seguente chiamata: `call Fusione('Legion of Doom', 'Marines', 'SquadraNuova');`

```
+-----------+---------------+
| Nome      | Squadra       |
+-----------+---------------+
| Sirius    | NULL          |
| Morpheus  | Globetrotters |
| MrFoxy    | Globetrotters |
| Mystica   | Globetrotters |
| Trinity   | Globetrotters |
| Albus     | SquadraNuova  |
| Aldebaran | SquadraNuova  |
| Eragon    | SquadraNuova  |
| Nihal     | SquadraNuova  |
| Tiw       | SquadraNuova  |
+-----------+---------------+
```

```
+---------------+--------+----------+
| Nome          | Leader | Speciale |
+---------------+--------+----------+
| Globetrotters | MrFoxy | S        |
| SquadraNuova  | Tiw    | S        |
+---------------+--------+----------+
```

QUERY 4) Visualizzare il nome della Città più “costosa”, ovvero la città in cui il prezzo medio di tutti gli oggetti di ciascun mercante sia superiore alle altre.

CODICE :

```
CREATE VIEW PrezzoMedio (Media, Insediamento) AS
SELECT      avg(Item.Valore), InsediamentoAbitato.Nome
FROM        InsediamentoAbitato JOIN InVendita on InsediamentoAbitato.Nome =
            InVendita.LuogoNegozio
            JOIN Item on InVendita.Item=Item.Codice
GROUP BY    InsediamentoAbitato.Nome;

SELECT      Insediamento, Media
FROM        PrezzoMedio
HAVING      Media = any (select max(Media)  from PrezzoMedio)
ORDER BY    Insediamento;
```

OUTPUT :

```
+--------------+---------+
| Insediamento | Media   |
+--------------+---------+
| Empire Bay   | 62.5000 |
+--------------+---------+
```

Triggers

TRIGGER 1) Trigger Before Delete on Eroe che verifica se l’eroe eliminato era un Leader di una Squadra Attuale; in tal caso inserisce il Nome della Squadra nella Tabella Squadra Passata impostando a NULL l’attributo Leader.

CODICE :

```
Delimiter $$
DROP TRIGGER IF EXISTS EliminaEroe $$
CREATE TRIGGER EliminaEroe
BEFORE DELETE ON Eroe
FOR EACH ROW  BEGIN
declare X varchar(30);
SELECT SquadraAttuale.Nome into X
FROM SquadraAttuale
```

```
WHERE SquadraAttuale.Leader=old.Nome;
insert SquadraPassata values (X, NULL, NULL);
END; $$
```

TRIGGER 2) Trigger After Insert on SquadraPassata che verifica se esistono Eroi il cui contenuto dell'attributo Squadra coincida con la Squadra inserita in SquadraPassata. In tal caso inserisce all'interno della tabella MembroPassato il nome dell'Eroe e il contenuto dell'attributo Squadra; inoltre imposta a NULL il contenuto dell'attributo Squadra nella corrispondente tupla della tabella Eroe.

CODICE :

```
DROP TRIGGER IF EXISTS VecchioMembro $$
CREATE TRIGGER VecchioMembro
AFTER INSERT ON SquadraPassata
FOR EACH ROW  BEGIN
insert into MembroPassato
        SELECT Eroe.Squadra, Eroe.Nome
        FROM Eroe
        WHERE Squadra=new.Nome;
delete from SquadraAttuale where Nome=new.Nome;
END; $$
```

TRIGGER 3) Trigger Before Insert on Appartenenza verifica che l'inserimento sia corretto. Nello specifico verifica che la somma del peso di tutti gli Item all'interno dell'inventario dell'Eroe in questione sommato al peso dell'item inserito non superi la capienza dell'inventario in questione; altrimenti blocca l'operazione con un apposito messaggio di errore. Verifica inoltre che l'Item in questione non sia già in possesso di un altro Eroe e nel caso in cui lo stesso Item sia droppabile da un Mostro elimina la relativa tupla in ItemDrop.

CODICE :

```
DROP TRIGGER IF EXISTS ControllaAppartenenza $$
CREATE TRIGGER ControllaAppartenenza
BEFORE INSERT ON Appartenenza
FOR EACH ROW  BEGIN
declare X smallint;        declare Y smallint;        declare Z int;
SELECT sum(Item.Peso) into X
FROM Item join Appartenenza on Item.Codice=Appartenenza.Item
WHERE Appartenenza.Inventario=new.Inventario
GROUP BY Appartenenza.Inventario;
SELECT Item.Peso into Y
```

```
FROM Item
WHERE Item.Codice=new.Item;
SELECT Inventario.Capienza into Z
FROM Inventario
WHERE Inventario.Proprietario=new.Inventario;
IF X + Y > Z
        then    signal sqlstate '45000'   set message_text = 'Capienza Massima inventario superata';
END IF;
IF exists (select Item from Appartenenza where Item=new.Item)
        then    signal sqlstate '45000'   set message_text = 'Item gia in possesso di un altro Eroe';
END IF;
IF exists (select Item from ItemDrop where Item=new.Item)
        then    delete from ItemDrop where Item=new.Item;
END IF;         END; $$
```

TRIGGER 4) Trigger Before Insert on InVendita verifica che sia possibile un inserimento corretto, nello specifico verifica che l'Item non sia già in possesso di un Eroe altrimenti blocca l'operazione con un apposito messaggio di errore. Inoltre verifica che l'Item non sia droppabile da un Mostro altrimenti blocca l'operazione con un apposito messaggio di errore (s'intende lo stesso identico Item, identificato univocamente dal suo codice).

CODICE :

```
DROP TRIGGER IF EXISTS ControllaVendita $$
CREATE TRIGGER ControllaVendita
BEFORE INSERT ON InVendita
FOR EACH ROW        BEGIN
IF exists (select Item from ItemDrop where Item=new.Item)
        then
        signal sqlstate '45000'
        set message_text = 'Errore! Item impossibile da mettere in vendita perche Droppabile da un Mostro';
END IF;
IF exists (select Item from Appartenenza where Item=new.Item)
        then    signal sqlstate '45000'   set message_text = 'Item gia in possesso di un Eroe';
END IF;         END; $$
```

TRIGGER 5) Trigger After insert on GodSlayer che verifica se l'Eroe inserito sia leader di una Squadra Attuale ed in tal caso imposta l'attributo Speciale della corrispondente tupla nella tabella Squadra Attuale a 'S'.

CODICE :

```
DROP TRIGGER IF EXISTS PromozioneSquadra $$
CREATE TRIGGER PromozioneSquadra
AFTER INSERT ON GodSlayer
FOR EACH ROW        BEGIN
UPDATE SquadraAttuale SET Speciale='S' where Leader=new.Nome;
END; $$
```

TRIGGER 6) Trigger Before Insert on ArmaUnica che verifica se nella tabella ItemDrop esiste una tupla avente come Item l'ArmaUnica, nel caso verifica se il corrispondente Mostro è il Boss indicato in fase di inserimento e se DropRate=100. Se ciò non fosse annulla l'inserimento.
Se non viene trovata una corrispondenza di ArmaUnica in Item Drop il trigger provvede a crearla in modo corretto.

CODICE :

```
DROP TRIGGER IF EXISTS VerificaDropArmaUnica $$
CREATE TRIGGER VerificaDropArmaUnica
BEFORE INSERT ON ArmaUnica
FOR EACH ROW        BEGIN
declare Y char(8);        declare Z smallint;
SELECT Mostro, DropRate into Y, Z
FROM ItemDrop
WHERE Item=new.CodiceArma;

IF not exists (select Item from ItemDrop where Item=new.CodiceArma)
        then    insert into ItemDrop values (new.CodiceArma, new.DropBoss, 100);      END IF;
IF  Y != new.DropBoss or Z != 100
        then    signal sqlstate '45000'    set message_text = 'Corrispondente tupla in ItemDrop scorretta';
END IF;         END; $$
```

TRIGGER 7) Trigger Before Insert on ItemDrop. In modo analogo al trigger n.6 verifica che l'eventuale inserimento di una tupla avente come Item un'Arma Unica oppure come Mostro un Boss sia corretta, nello specifico verifica che: non sia inserita un'Arma unica già presente (poichè ogni Arma unica è droppata da uno e un solo Boss), non sia inserito un Boss già presente (poichè ogni Boss droppa unicamente la sua ArmaUnica) e infine che l'inserimento di un'ArmaUnica abbia indicato il DropRate=100.

CODICE :

```
DROP TRIGGER IF EXISTS VerificaDropArmaUnica2 $$
CREATE TRIGGER VerificaDropArmaUnica2
BEFORE INSERT ON ItemDrop
FOR EACH ROW BEGIN
IF exists (select * from ItemDrop join ArmaUnica on ItemDrop.Item=ArmaUnica.CodiceArma where
    ItemDrop.Item=new.Item)
    then signal sqlstate '45000' set message_text = 'Erroe! Ogni ArmaUnica puo essere droppata da
    uno e un solo Boss!'; END IF;
IF exists (select * from ItemDrop join Boss on ItemDrop.Mostro=Boss.Codice where
    ItemDrop.Mostro=new.Mostro)
    then signal sqlstate '45000' set message_text = 'Errore! Ogni Boss puo droppare unicamente la sua
ArmaUnica!'; END IF;
IF exists (select * from ArmaUnica where CodiceArma=new.Item) and new.DropRate != 100
    then signal sqlstate '45000' set message_text = 'Errore! Ogni ArmaUnica viene droppata con %
    DropRate = 100!';
END IF; END; $$
```

TRIGGER 8) Trigger After Insert on SquadraAttuale che Aggiorna la tabella Eroe, impostando per il Leader della Squadra Inserita il corrispondente attributo Squadra al nome della Nuova Squadra.

CODICE :

```
DROP TRIGGER IF EXISTS AssociaLeader $$
CREATE TRIGGER AssociaLeader
AFTER INSERT ON SquadraAttuale
FOR EACH ROW BEGIN
update Eroe set Eroe.Squadra=new.Nome where Eroe.Nome=new.Leader; END; $$
```

TRIGGER 9) Trigger Before Update on Eroe che verifica che non vi possano essere Squadre con un numero di membri superiore a 6, altrimenti blocca l’operazione con un apposito messaggio di errore.

CODICE :

```
DROP TRIGGER IF EXISTS ControllaNumMembri $$
CREATE TRIGGER ControllaNumMembri
BEFORE UPDATE ON Eroe
FOR EACH ROW BEGIN
declare N smallint;
SELECT count(*) into N
FROM Eroe
WHERE Squadra=new.Squadra
```

```
GROUP BY Squadra;
IF N > 5
        then    signal sqlstate '45000'   set message_text = 'Numero massimo di membri (6) superato per una
        Squadra';
END IF;         END; $$
```

TRIGGER 10) Trigger Before Insert on Eroe che ha un comportamento analogo al trigger precedente; verifica che non vi possano essere Squadre con un numero di membri superiore a 6.

CODICE :

```
DROP TRIGGER IF EXISTS ControllaNumMembri2 $$
CREATE TRIGGER ControllaNumMembri2
BEFORE INSERT ON Eroe
FOR EACH ROW        BEGIN
declare N smallint;
SELECT count(*) into N
FROM Eroe
WHERE Squadra=new.Squadra
GROUP BY Squadra;
IF N > 5
        then    signal sqlstate '45000'   set message_text = 'Numero massimo di membri (6) superato per una
        Squadra';
END IF;         END; $$
delimiter ;
```