Rendu Projet BDD2

BOUTIN Loïc, HACQUES Florian, NYUNTING Elbert, PINEL Félix

Mars 2017

Résumé

Dans le cadre de l'UE X6I0050 "Bases de Données 2" nous avons réalisé ce projet. Il comporte le bestiaire de Dungeons and Dragons $5^{ème}$ édition tiré du livre Monster Manual. Une base de données afin d'aider un maître de jeu à plus facilement trier et trouver des monstres à lancer aux aventuriers non préparé, comme le livre lui même est un cauchemar à naviguer, contrairement à cette base de données. Ce rapport expliquera la structure et contenu de notre base de donnée ainsi que le déroulement de ce projet.

Table des matières

1 Organisation du travail

Pour le déroulement de ce projet, nous avons travaillé chacun sur des différents aspects de la base de données. Ceci dit, nous partageons le travail pour éviter d'avoir un membre qui est isolé du groupe. Nous avons fortement utilisé Git pour pouvoir rassembler nos travaux et les comparer/corriger.

Nous avons fait des petits changements sur notre base de données par rapport à ce qu'on voulait faire précédemment. Nous avons ajouté un attribut `idMob` qui servira de clé primaire sur le tableau `Caracteristique`. Vu que nous avons maintenant un rôle (`maitreJeuCollaborateur`) qui peut ajouter un monstre qui a le même nom qu'un montre existant mais avec des attributs différents, idMob permet donc de différencier ces monstres homonymes. Nous avons ajouté un attribut `vu` qui est en booléen, cet attribut nous permet de déterminer si un utilisateur de rôle `joueur` a déjà vu ce monstre, ce qui déterminera ce qu'il peut voir avec les procédures de `VUE`. Cet attribut est caché des joueurs lorsque ces dernier font une `SELECT`.

La diagramme de notre base de donnée est donc :

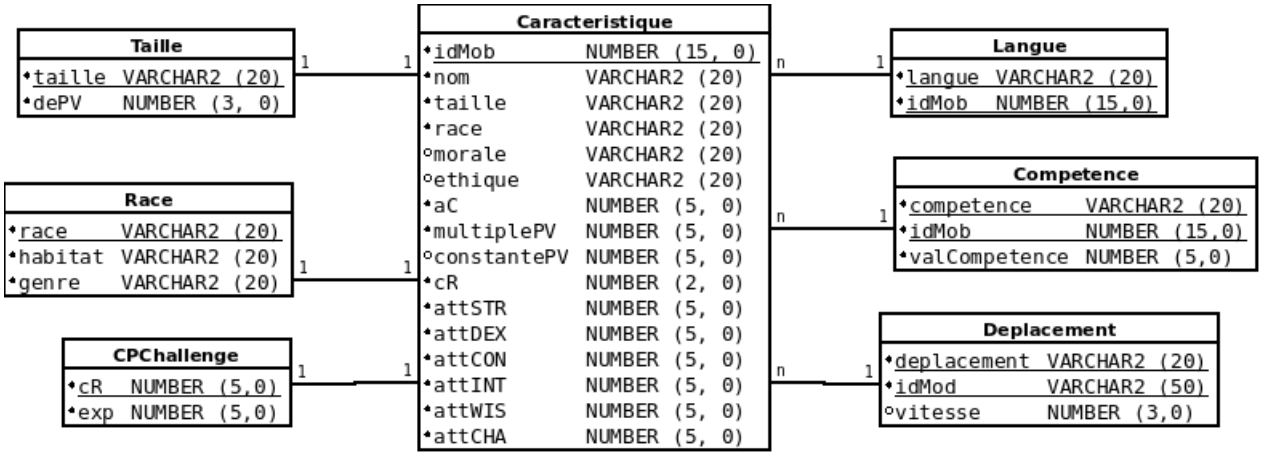

Figure 1 – UML de la base de données finale

La table R7 est supprimé, la table Langue est mis à sa place. Les tables `Competence`, `Langue` et `Deplacement` sont mùaintenant directement liées à `Caracteristique`. Ces trois tables ont chacun deux clés primaires dont un est `idMob` afin d'avoir un monstre capable d'avoir plusieurs compétences, langues et méthodes de déplacement, ou aucun. Dans le cas de ce dernier, le monstre n'est pas présent dans la table (ex : un monstre incapable de parler n'aura pas son `idMob` dans la table `Langue`.)

2 La création des tables

2.1 Caracteristique

La table `Caracteristique` est contenu dans le fichier `Crea_Table_Carac.sql`. Le fichier contient la création de la table ainsi que certains `CHECK` sur certain attributs. L'armure (l'attribut `aC`) d'un monstre ainsi que ses attributs (`attSTR`, `attDEX`) ne peux pas dépasser 30. `attSTR` représente la force physique d'un monstre, `attDEX` représente sa dextérité et réflexes, `attCON` sa constitution et résistances physiques, `attINT` son intelligence, `attWIS` sa sagesse et `attCHA` son charisme. Tout ses attributs sauf `morale` et `ethique` ne peut etre `NULL` (un montre peut ne pas avoir de morale ou d'éthique. La séquence `seqIdMob` nous permet une incrémentation automatique des monstres lorsqu'ils sont insérés dans la table. L'attribut `vu` est un pseudo-booléen qui nous permet de déterminer si un utilisateur de rôle `Joueur` a déjà rencontrer la créature. Dans cette table, la clé primaire est bien sur `ifMob`.

Pour les indexes nous avons pris l'attribut `nom` de la table `Caracteristique` car c'est ce qui est le plus recherché dans un bestiaire. En effet un utilisateur cherche surtout le nom d'un montre au lieu de ses points de vie,

langues, etc. L'index `CREATE INDEX ind_nom ON Caracteristique(nom);` se trouve dans le fichier création du table `Caracteristique Crea_Table_Carac.sql`.

2.2 Competence

La table `Competence` contient les compétences acquises de chaque monstre ainsi que la valeur numérique de chaque compétence. Elle prend comme clés primaires `idMob` et `competence` car nous avons içi une liste de compétence pour chaque monstre, donc nous devons prendre deux clés primaires et non pas une seule. Un monstre qui n'a aucune compétence ne se trouve pas dans cette table.

2.3 Langage

La table `Langage` contient, comme la table `Competence`, une liste d'attributs que chaque monstre peut avoir, sauf cette fois ci ce sont les langues connues par les monstres. Elle se trouve dans le fichier `Crea_Table_Depla.sql` a comme clés primaires `idMob` et `langue` pour la même raison que `Competence` en a deux : un monstre peut connître plusieurs langues. Si un monstre est incapable de parler, il aura comme langue `analphabete` et ce cas sera traité par les triggers `Trigg_Lang` et `TriggerSuppAnal`.

2.4 Deplacement

La table `Deplacement` contient, vous le savez, les modes de déplacements d'une créature donnée (vol, nage, à pieds, multi-dimensionnel, etc) et se trouve dans le fichier `Crea_Table_Depla.sql`. Elle a comme clés primaires `idMod` et `deplacement` car comme les deux tables précédentes, une créature peut se déplacer de différentes manières. Elle a aussi comme attribut la vitesse de déplacement pour chaque méthode de déplacement.

2.5 Race

Chaque créature à une race (d'un humble kobolds aux fiers dragons) et vit forcement quelque part. C'est pour cela que nous avons mis la table `Race` contenu dans `Crea_Table_Race.sql` et a comme clé primaire `race`. Ceci dit, pas toutes race ont un type (ici appelé `genre` pour éviter une confusion de syntaxe avec `type` de SQL) et donc c'est pour cela que l'attribut `genre` peut être `NULL` (un dragon peut avoir plusieurs genres comme sa couleur, mais il n'y a pas beaucoup de différents genres pour des Manticores.)

2.6 Taille

Chaque monstre a bien sur des tailles différentes. Un gobelin est bien sur plus petit qu'un géant de tonnerre. La table `Taille` trouvée dans le fichier `Crea_Table_Taille.sql` contient les différentes tailles (triées en petit, moyen, grand, jusqu'à gargantuan) ainsi que les dés de points de vie associés à ces tailles (plus un monstre est grand, plus il a de faces sur son dé.) La clé primaire est l'attribut `taille`.

2.7 XPChallenge

Un monstre offre un défi auprès les aventuriers, et le pallier difficulté de ces monstres changent de monstre en monstre (un diablotin n'offre pas autant de difficulté qu'un maître vampire) et la table `XPChallenge` (située dans le fichier `Crea_Table_XPChal.sql`) nous montre cela. Un monstre peut commencer à 0 de difficulté (ce qui représente un ennemi inconséquent pour les aventuriers comme un simple gobelin) et monte de un à un jusqu'à 30 de difficulté (seul un monstre a 30 de difficulté, le légendaire et intuable **Tarrasque** qui nécessite une intervention de plusieurs dieux pour simplement le faire endormir à nouveau.) L'incrémentation de ces "étapes" de difficulté se fait avec la `SEQUENCE seqXP` et le `CHECK` qui force sa valeur entre 0 et 30.

3 Fonctions de notre base de données

3.1 Procédures

3.1.1 coup_en_combat

Notre première procédure est une procédure qui nous calcule les points de vie d'un monstre victime (`nomB`) après avoir reçu une attaque du monstre `nomA`. Les points de vie sont mis en paramètre `vie` et c'est ce qui sera retourné ainsi que le nombre de dégâts avec lequel `vie` sera soustrait. Les autres paramètres sont `esq` qui prend de l'attribut `attDEX` du monstre victime et `sndCP` aui prend de `attINT` du monstre attaquant. Le paramètre `esq` permet au monstre victime d'éviter le coup et recevoir aucun dégâts tandis que `sndCP` permet au monstre attaquant d'attaquer une deuxième fois en un tour (un monstre rapide ne se fait pas toucher, mais un monstre malin sait qu'il peut frapper plusieurs fois.) Les chances d'esquive ou de coup repeté reste très bas car avec les contraintes, un `attDEX` et `attINT` ne peuvent dépasser 30 et ces esquives et second coups ne sont déclenchés que si le pourcentage lancée par `DBMS_RANDOM.VALUE(1,100)` est inférieur aux valeur de `esq` et `sndCP`.

3.1.2 combat2monstres

La procédure `combat2Monstres` prend en paramètre deux `idMob` de deux monstres différentes et en utilisant `coup_en_combat`, fait une "simulation" de tour par tour où ses monstres se frappent entre eux jusqu'à la vie d'un monstre est réduit à 0. La procédure `coup_en_combat` est utilisé de manière que chaque tour, le monstre attaquant et le monstre victime est inversé, les points de vie du monstre victime ainsi que le de dégâts calculé. La formule du dégât est la force du monstre attaquant multiplié par son pourcentage (100 - aC du monstre victime divisé par 100) de réussite (`hit := mobA.attSTR*((100-mobB.aC)/100);`). Le résultat de simulation n'est jamais fixe avec l'utilisation de `RANDOM`.

3.1.3 Fonct_Att_bonus

Cette fonction calcule la valeur des bonus des attributs (dans le sens `attSTR`, `attDEX`, `attINT`, etc et pas dans le sens attributs base de données) des monstres afin que l'utilisateur concerné (dont le rôle est maître du jeu) peut faire varier les lancers de dés des joueurs. Elle prend comme paramètre l`idMob` du monstre concerné et une chaîne de caractère `chxAtt` pour savoir quel bonus d'attribut (STR, DEX, WIS, etc) est cherché. Le bonus est calculée avec la formule (valeurAttribut/2)-5 (donc par exemple un monstre à 18 force et 8 de dextérité à un bonus de force de 4 et un malus en dextérité de 1.)

Des exceptions sont levés si `chxAtt` ne donne aucun attribut ou si l'`idMob` n'existe pas.

3.2 Triggers

Dans cette base de donnée nous avons utilisé certains triggers pour mieux utiliser les bases de données. Ces triggers travaillent surtout sur les compétences et les points de vie des monstres. Ce ne sont pas des triggers complexes, mais ils facilitent fortement l'utilisation de la base de données ainsi que l'ajout des langues et le fait que certains monstres ne connaissent pas de langues.

3.2.1 Trigg_Lang

Cet trigger ne fait qu'ajouter `analphabete` dans l'attribut `Langage` du monstre ajouté. Ce monstre ne connaît pas de langue jusqu'on lui ajoute une langue, ce qui déclenchera `TriggerSuppAnal`.

3.2.2 TriggerSuppAnal

Ce trigger est déclenché lorsqu'un monstre apprend une langue. Si la langue ajoutée n'est pas `analphabete`, la créature perd ce dernier comme langue et prend comme nouvelle langue la langue ajoutée. Une créature peut bien sur apprendre plusieurs langues, donc si une créature non analphabète apprenne une autre langue, ce trigger n'est pas déclenché.

3.2.3 TriggerTopv

Ce trigger calcule les points de vie d'un monstre lorsqu'il est ajouté dans la table `Caractéristique`. Ce trigger prend le type de dé à lancer pour aléatoirement les points de vie d'un monstre dans la table `Taille` (plus un monstre est grand, plus le dé contient de faces). Le trigger lance ensuite le dé le nombre de fois que ce monstre à de `multiplePV` en ajoutant à chaque fois le jet obtenu. Ensuite on ajoute la `constantePV` pour avoir le résultat final.

Par exemple, un Henzrou a comme taille `Large`, donc il a comme dé à lancer un d10 (dé à 10 faces). Son `multiplePV` est 13 donc un `LOOP` est fait 13 fois pour récupérer le résultat des lancers. Après ce loop les points de vie du Henzrou est de (par exemple, comme c'est aléatoire) 72. sa `constantePV` est de 65 donc ses points de vie est de 137 (un nombre assez considérable pour des aventuriers novices !)

3.3 Rôles

Dans une partie de jeu de rôle, pas tous les personnes concernées ont le même... rôle. Il y a surtout des joueurs, mais il y a aussi le maître du jeu qui, derrière son écran de jeu impénétrable, concocte des scénarios meurtrières avec des monstres vicieux. C'est pour cela nous avons instauré des rôles différents dans nôtre base de données avec des privilèges et droits plus ou moins restrictifs qu'on puisse voir dans le fichier `Crea_Role.sql`.

Commençant par le bas, nous avons le joueur, nommée ici `DnDjoueur`, qui n'a droit que de `SELECT` sur les `VIEW Globale`, `Sociale` et `Combat` et rien d'autre (le joueur est représenté ici avec Elbert NYUNTING.)

Ensuite nous avons `DnDmaitreJeuCollaboratuer`, un rôle particulier lorsqu'un maître jeu externe veut mettre sa version d'un monstre particulier sur la base de données. LE fait que nous utilisons `idMob` comme clé primaire dans `Caracteristique` et pas `nom` est parce que un monstre peut avoir le même nom mais rien d'autre (par exemple son Dragon Vert est plus agile mais moins fort que celui du Dragon Vert de base) si un `DnDmaitreJeuCollaboratuer` veut rajouter sa version de ce monstre. Un `DnDmaitreJeuCollaboratuer` peut tout sélectionner et tout insérer (sauf des insertions dans la table `XPChallenge` et `Taille`, qui est fixée par l'admin) mais il ne peut faire aucune delétion dans la base de donnée. Félix PINEL prendra le rôle d'un maître jeu collaborateur.

Le maître jeu de cette base de données (`DnDmaitreJeu`, et représenté par Florian HACQUES) a tout le pouvoir d'un maître jeu collaborateur mais il peut aussi supprimer des données dans tous les tables sauf les tables `XPChallenge` et `Taille`. Il ne détient pas le pouvoir absolu, ceci est strictement pour...

...L'admin. L'administrateur à tout le pouvoir sur cette base de données. Il peut ajouter et supprimer des données sur toutes le tables (même les tables `XPChallenge` et `Taille`.) Il peut voir toutes les données dans sa totalité, entravé par aucun `VUES`. Il peut réarranger la structure des tables et même les liaisons au sein de cette base de données. Mais surtout, il peut donner et retirer des droits aux ceux qu'il souhaite. Ce privilège d'administrateur est de base dans les paramètres de l'utilisateur et ne nécessite donc pas de précision dans le fichier `Crea_Role.sql`. Loïc BOUTIN jouera le rôle de cet être absolu et tout puissant.

3.4 Vues

Afin de restreindre les joueurs sur ce qu'ils peuvent voir dans nôtre base de données, nous avons mis en place trois différents `VIEWS`. La première est un `VIEW Globale` (dans le fichier `Crea_View_Glob.sql`) qui permet aux joueurs de voir le nom, taille, habitat, genre, morale, éthique, points d'armure (cA) et points de vie (PV) d'un monstre qu'il a déjà rencontré (dont le `WHERE vu = 1;`)

Ensuite nous avons le `VIEW Combat`, qui permet aux joueurs de voir les attributs (dans les sens d'une base de données) pertinentes lorsqu'ils affrontent un monstre sur le champ de bataille. Les joueurs peuvent voir le nom, taille, genre, points d'armure et points de vie d'un monstre qu'il ont déjà affronté.

Et enfin nous avons le `VIEW Sociale`, utile pour des rares moments où les joueurs cherchent à parler aux monstres au lieux de les combattre. Ce `VIEW` montre aux joueurs le nom, la race, l'habitat, le genre ainsi que la morale, l'éthique et la langue (si la créature en possède) des créatures que le joueur à déjà vu.

4 Conclusion

Pour conclure, nous avons trouvé ce projet assez intéressant et utiles (deux entre nous sont des maîtres jeu.) Ceci dit, nous avons eu quelques difficultés sur certains aspects de cette base de données. Premièrement sur le fait que certains monstres parlent plusieurs langues tandis que d'autres n'en parlent aucune. Un autre problème est de transformer la collection impressionnantes de monstres dans l'ouvrage Monster Manual (qui n'est même pas une base de donnée de type 1) à une base de données de type 3 utilisable. Pour cela nous devons omettre certaines choses présentes dans l'ouvrage qui rendent l'insertion de ces monstres difficile dans une base de données de type 3 (par exemple le fait que les monstres ont des différentes façons d'attaquer qui sont uniques à eux, avec des effets d'attaques uniques qui peut prendre plusieurs phrases à expliquer.) Un autre défi est la quantité de monstres dans le livre, qui nous prendra encore une semaine à tous les insérer, c'est pour cela nous avons décidé de n'insérer qu'un petit échantillon de monstres dans la base de donnée.

En tout, ce projet est intéressant tout en étant réalisable et nous demande d'utiliser nos connaissances dans la matière.