SUJET TP - SGBD (R)

POKEMON GO, la base de donnée SQL

VERSION BETA



Sujet crée pour:

ei.CESI / Strasbourg

GMSI15 - 2015-2016

Versions

Date	Auteur	Commentaire
15.07.2016	Georges OLIVARÈS	Création du document et du TP
28.08.2016	Georges OLIVARÈS	Ajout du corrigé

Licences



Cette œuvre est mise à disposition selon les termes de la
Licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions 4.0 International.



Ce document analyse le jeu **Pokémon GO**, crée, édité et distribué par la société Niantic, Inc.

Pour toutes remarques ou question associées à ce document, vous pouvez vous référer à ce lien:

http://blog.olivares-georges.net/[insert pokemon go article]

Description (Wikipedia)

Pokémon Go est un jeu en réalité augmentée. Le joueur dirige un avatar, dont il peut personnaliser l'apparence et la tenue à la création de son compte, sur une carte du monde. Cette carte du monde est liée à la position du joueur sur Terre et celui-ci doit ainsi se déplacer avec son téléphone pour se déplacer dans le jeu. Le joueur peut observer à une certaine distance les environs, ce qui permet d'afficher les différents éléments du jeu à portée : Pokémon, PokéStops et arènes. Il peut ainsi parcourir les villes, capturer des Pokémon ou encore combattre dans des « arènes ».

Lorsque le joueur se déplace, il rencontre des Pokémon, qu'il peut attraper à l'aide de pokéballs récupérées dans les PokéStops ou en gagnant des niveaux. L'emplacement des Pokémon est aléatoire et ils disparaissent au bout d'un certain temps ; plus le Pokémon est rare, plus son temps d'apparition est faible. Enfin, le type de l'espèce influe sur son apparition : un Pokémon de type eau aura plutôt tendance à apparaitre près des côtes et lacs qu'en pleine terre. Lorsqu'un Pokémon est visible sur la carte, il peut être attrapé une fois par dresseur à portée.

Contrairement aux autres jeux de la licence, le joueur n'a pas à combattre les créatures qu'il rencontre. Lorsque le joueur appuie sur un Pokémon visible sur la carte, le jeu passe dans un mode où le joueur peut lancer des pokéballs afin d'attraper la créature. Le joueur peut rater son tir ou le Pokémon s'échapper (et éventuellement fuir), nécessitant fréquemment plusieurs pokéballs lors d'une rencontre. Une fois le Pokémon attrapé, le joueur en devient le propriétaire et se voit attribuer de la poussière d'étoiles et des bonbons. Ces deux objets permettent de rendre les créatures plus fortes, en augmentant leurs points de combat (CP). Les bonbons permettent également de faire évoluer le Pokémon, ou un autre de la même espèce. Le joueur peut également transférer le Pokémon au professeur afin de gagner un bonbon supplémentaire et faire de l'espace dans son sac. En effet, le joueur a un nombre limité de Pokémon dans son sac. Le but du jeu est de capturer chaque espèce de Pokémon existante, afin de remplir le Pokédex, encyclopédie recensant les créatures connues. Certains Pokémon ne sont cependant accessibles que via des événements spéciaux organisés par Niantic ou par évolution.

Lors de la rencontre avec le joueur, chaque Pokémon se voit attribuer des points de combat qui définissent sa puissance. Plus le joueur a un niveau élevé, plus il rencontrera des Pokémon avec un grand nombre de CP. Cependant, des créatures de faible niveau apparaitront encore également.

Les PokéStops et les arènes sont eux à des lieux précis sur la carte, principalement à des points d'intérêts dans le monde réel, et sont repris pour la plupart du jeu Ingress. Le joueur doit se trouver à proximité du lieu d'intérêt (parc, monument, fresque, etc.) pour pouvoir agir avec dans le jeu. Les PokéStops fournissent au joueur un certain nombre d'objets aléatoirement : des œufs, pokéballs, des leurres ou encore des potions. Les arènes sont des lieux de combat, où les équipes du jeu s'affrontent afin de couvrir le plus d'arènes possible à travers le monde.

Il existe dans Pokémon Go trois équipes qui disposent chacune de leur spécialité :

- l'équipe Jaune - Intuition ;
- l'équipe Bleu - Sagesse ;
- l'équipe Rouge - Bravoure.

Le choix d'une équipe se fait à partir du niveau 5 et est à l'heure actuel définitif. Chaque arène est défendue par une équipe, qui en a pris le contrôle en mettant un ou plusieurs Pokémon en défense. Le joueur peut alors affronter les membres des autres équipes dans les arènes afin de récupérer le contrôle de celles-ci ou défendre celles déjà protégées par son équipe en y ajoutant un de ses Pokémon.












Sujet

Reproduire une base de donnée simplifiée du jeu, en se basant sur les remarques suivantes:

- Dans un monde libre, il existe des **emplacements** spécifiques (géolocalisation):
  - **source** d'apparition de Pokémons: ceux-ci sont générés par le serveur, via un programme qui régulièrement fera apparaitre les créatures. *Le programme n'est pas demandé, mais quelques requêtes SQL seront abordées.*
  - de **Pokéstops**: un nom, une image
  - ou d'**arènes**:
    - les arènes peuvent être « avec leurre » (avec le nom de la personne qui l'a déposé)
    - Il y a un niveau, et un nombre de Pokémons à combattre avec leur dresseur
- Il y a des **Pokémons** (un catalogue, et une liste personnelle)
  - #, nom, taille, poids, CP, PV, et le lieu où le Pokémon a été attrapé.
  - deux **attaques** parmi une liste donnée: Les attaques ont un type et une force d'attaque et sont sélectionnée à partir d'une table de référence et se basent sur les types (deux là aussi) du Pokémon. La 1ere à un fort taux de chance d'être de type normal (Charge, ...).
    Un modèle plus complexe permettrait de gérer certains Pokémons comme Magicarpe qui a toujours les mêmes attaques. Il faudrait alors faire une table de correspondance pokemon/Attaques (ce qui permettrait d'inclure les DPS, ...)
- Il y a des **objets**
  - Œufs chances, potions, rappels, pokéballs (de différent types), encens, incubateurs, …
- Et un **dresseur**
  - avec un pseudo (compte), des points d'expériences/niveau, une équipe (b/j/r), un **porte monnaie** ;
- Il y a aussi une gestion de l'**argent**
- des **médailles**, avec un système de règle pour les obtenir
- Et pour finir, un **système de journal**

Requêtes

Dans le jeu, nous avons besoin de récupérer les informations suivantes:

- Les Pokémons
  - La liste de tous les Pokémons (les 151) du catalogue
  - La liste de tous les pokémons d'un type particulier
  - La liste de tous les Pokémons , présent dans une zone spécifique *(pour un soucis de simplification, on se base sur un carré)*
- Les lieux
  - La liste des points spécifiques de la zone (arènes, Pokéstops, zones d'apparition)
    - Une fois sélectionné, les informations du Pokéstops
    - Une fois sélectionné, les informations des arènes (les X Pokémons présents, le niveau de l'arène, savoir si il reste une place en fonction du niveau de celle-ci)
    - La liste des 6 plus forts (CP) Pokémons du joueur
- Les options du menu
  - Les informations d'un Pokémon : Pokédex
  - La liste des Pokémons capturés avec leur niveau. On peut filtrer par PV/Points de combat/Numéro/…
  - Les œufs et leur état (nombre de Km parcourus, et si sous incubateur)
  - Les objets du sac (le catalogue + le nombre d'items en procession)
  - Le niveau du joueur
  - La liste des médailles disponibles
  - ...
- Do coté serveur
  - Générer aléatoirement un nombre aléatoire de Pokémons dans les zones d'apparitions : prendre le type de la zone (eau, sol, montagne, ville, campagne, ...) et générer une liste de pokémons de ce type.
  - Une autre requête peut cette fois ci générer des pokémons plus rares.

Créez les tables pour chacun des cas, en répondant aux questions suivantes. A vous de créer l'architecture adéquate pour reproduire le modèle pensé.

Pour information, il existe une multitude d'architectures pour répondre à un problème ; les solutions apportées ici ne sont que personnelles : vous pouvez avoir une tout autre solution :)

Les points marqués par des (★) montrent la difficulté de la requête (*Pour information, vous pouvez faire plusieurs requêtes avec du code dans votre application ou parfois, faire toutes les opérations directement dans SQL. C'est cette dernière qui sera corrigé dans ce sujet, bien que la même chose pourra être fait via 2 ou plus, requêtes plus simples*)





Le dresseur

0100 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible (le dresseur, ses informations de connexion au jeu, son équipe, la personnalisation de son personnage).

0101 - Donner les informations de base du joueur (nom, niveau (xp), équipe, ...)

0102 - Ajouter un compte utilisateur

0103 - Verifier si l'utilisateur à bien un compte, et que son pseudo/mdp est bon

0104 - Donner le niveau (level) du dresseur





Les pokémons

0200 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible.

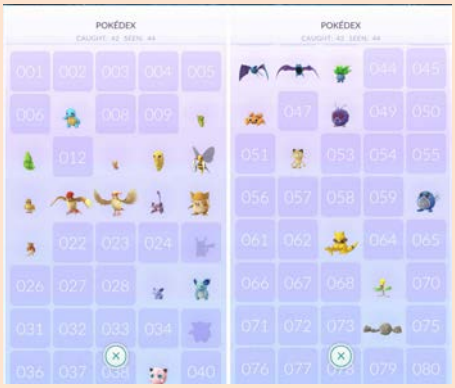
0201 - Donner le nombre de pokémons attrapés (sac) et le nombre de pokémons dans des oeufs

0202 - Lister tous les pokémons attrapés (nom, CP, vie)

0203 - Donner tous les détails d'un pokémon selectionné (plusieurs reqûetes).

0204 - Donner le nombre de pokébonbons disponibles associés aux pokémons pour le dresseur

0205 - Filtres: trier les pokémons du joueur par plusieurs filtres possibles (PV, PC, #, ...)



Le pokédex

0300 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible.

0301 - Lister les 151 pokémons du jeu

0302 - La liste de tous les pokémons d'un type particulier

0303 - Donner le #nombre du pokémon attrapé/vue le plus loin dans la liste des pokémons

0304 - Dire si un pokémon a été attrapé ou vu.

0305 - Rattata a été attrapé, l'inscrire au pokédex

0306 - Le pokédex d'un pokémon particulier





Les pokestops

0400 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible.

0401 - Lister les pokestops disponibles dans la base de donnée. On pourrait se baser sur les coordonnées GPS pour ne lister que ceux disponibles dans un rayon de X m/km (★★, utilisation des coordonnées GPS)

0402 - Donner les caractéristiques du pokéstop sélectionné





Arènes

0500 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible.

0501 - Quels sont les pokemons dans l'arène selectionnée ? Donner le dresseur, et sa team ainsi que le nom du pokémon et son nombre de PC.

0502 - Selectionner l'équipe associé à l'arène

0503 - Selectionner les 6 pokémons les plus puissant du joueur

0504 - Retrouver les caractéristiques des deux pokémons en train de combattre (nom, pv, pc). Lister aussi les deux attaques disponibles





Boutique

0600 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible.

0601 - Donner le nombre de poképièces du joueur

0602 - Lister tous les articles disponibles dans la boutique (objet, prix)

0603 - Acheter 8 Œufs Chance, les ajouter au dresseur

0604 - Acheter 500 poképièces, les ajouter au dresseur



Les médailles

0700 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible.

0701 - Lister les médailles disponibles dans le jeu

0702 - Donner pour chaque médaille, le niveau pour le joueur (= nombre de Pokémons/actions, une formule mathématique peut être utilisé pour savoir si c'est le niveau 1, 2 ou 3)



Le Journal

0800 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible.

0801 - Insérer une entrée dans le journal

0802 - Lister les 25 dernières actions du journal



Les œufs

0900 - Décrire le modèle. Créer les tables pour le reproduire le plus fidèlement possible.

0901 - Lister les œufs disponibles pour le joueur. Donner ceux en cours d'incubation et leur état en terme de Km.

0902 - Lister les œufs du joueur à faire éclore

0903 - Lister les Pokémons pouvant sortir des œufs, triés par Km

0904 - Donner le Pokémon pouvant sortir de l'œuf une fois éclos (se baser sur une liste pour 2/5/10 et utiliser une fonction "aléatoire") (★★, décomposer pour simplifier)

0905 - Ajouter une certaine distance parcourue (entre deux points GPS) aux œufs dans l'incubateur et ajouter cette même distance dans la médaille associée (★★★, *il faut passer par du code applicatif*)

Info: Les Km sont incrémentés coté GPS.

D'autres aspects pourraient être abordés plus en détail: les évolutions, les attaques, les traductions (fr/en/jp/...), ...

Corrigé - Requêtes

Le corrigé ne présente qu'une vision personnelle de l'architecture de ce jeu. Toutes les données utilisées appartiennent à la société éditrice du jeu, et ont été trouvé sur internet.

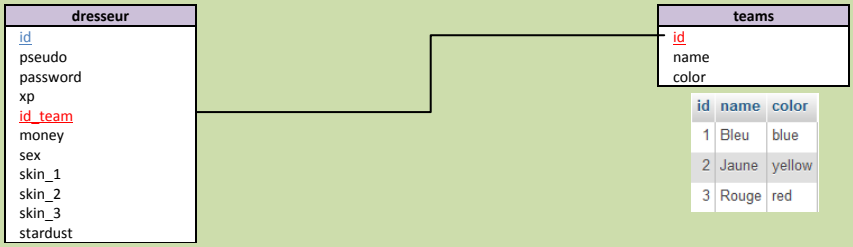
Le modèle SQL crée est attaché à ce document:

[insert icon of .SQL file]
[lien URL to download the .SQL file]

Le dresseur

Tables

Les informations du joueur sont regroupées dans une table qui va servir à se connecter et à stocker les caractéritiques du joueur (physique, nom, ...). Ici on ne considère que la connection physique pseudo/mdp et non pas via Google.
Il faut aussi s'arranger pour enregistrer l'équipe (rouge, bleu, jaune) du joueur : pour cela, un champ dans la table "dresseur" au format texte ou une relation avec une table "teams" tiers.
Les champs "money" et "stardust" (poussière d'étoile) de la table "dresseur" s'incrementent (à l'achat ou quand on attrape un pokémon).



id	name	color
1	Bleu	blue
2	Jaune	yellow
3	Rouge	red



Pour la gestion de l'avatar: Ici, je ne présente que 3 options sur les 9 disponibles. Etant donné qu'il n'y a que 8 options + 1 pour le sexe, il est tout à fait possible de stocker cette information dans la table "dresseur" (pas besoin de créer une table supplémentaire).
Ainsi, nous aurons *sex*, *skin_1*, *skin_2*, *skin_3*, ..., *skin_8*. Au niveau des types, un simple *integer* est suffisant (au niveau du code, il suffira de créer des ressources du type "face1", "face2", "skin=1 -> #ffffff", ...).

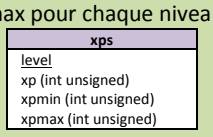
id	pseudo	sex	skin_1	skin_2	skin_3
1	Thiktak	1	2	2	3
29069011	MatchouPitchou	2	1	2	1
29069014	Toto	1	0	0	0

Technique pour augmenter la sureté: Ici, le mot de apsse est stocké en MD5. Le problème du MD5 est qu'il existe des dictionnaires permettant de faire du "reverse décryptage" et donc de retrouver les mots de passe. Ainsi, un hacker, si il récupère votre table pourra aisaiement récupérer tous els comptes de votre application.
Pour augmenter la sureté, on peut utiliser une technique dite de "SALT" (trad: sel) : nous alons insérer dans le mot de passe une chaine de caractère n'étant stockée que dans le code source de l'application et/ou une chaine de caractère aleatoire générée pour chaque utilisateur (ici, champs hash). A la vérification de la connexion, il ne faudra pas oublier de "hasher" le mot de passe.

Nous aurions alors: password = MD5(CONCAT(hash, 'monSuperP@ssword', hash))

id	pseudo	password	hash
1	Thiktak	b61dd600c9afbbff9025810aa84c14cb	96u]Ps{Y.QPb5T@+

Pour les niveau, plusieurs solutions: calculé dans le code source, ou via une table. Si on utilise la table, il existe plusieurs méthodes dont stocker l'XPmin et l'XPmax pour chaque niveau et chercher le niveau en fonction de l'expérience du joueur (REQ#0104).



level	xp	xpmin	xpmax
1	0	0	999
2	1000	1000	2999
3	2000	3000	5999
4	3000	6000	9999
5	4000	10000	14999

0101 Donner les informations de base du joueur (nom, niveau (xp), équipe, ...)

SQL

```
SELECT id, pseudo, xp, id_team, money, sex, skin_1, skin_2, skin_3
  FROM dresseur
  WHERE id = '29069011'
```

id	pseudo	xp	id_team	money	sex	skin_1	skin_2	skin_3
29069011	MatchouPitchou	35621	3	2000	2	1	2	1

0102 Ajouter un compte utilisateur

SQL

```
SELECT LEFT(UUID(), 8) INTO @hash; -- Créer une variable '@hash' avec la valeur d'un UID (token unique)
INSERT INTO dresseur
  SET pseudo = 'Toto', password = MD5(CONCAT(@hash, 'MyPwd', @hash)), hash = @hash
```

id	pseudo	password	hash	xp	id_team	money	skin_1	skin_2	skin_3	pstar
29069014	Toto	9dd641677f165f2484e119529e957490	fbb68578	0	*NULL*	0	0	0	0	0

0103 Verifier si l'utilisateur à bien un compte, et que son pseudo/mdp est bon

SQL

```
SELECT id
  FROM dresseur
  WHERE pseudo = 'Toto' AND password = MD5(CONCAT(hash, 'MyPwd', hash))
```

id
29069014

Il y a un ID/un résultat, donc le compte existe.

0104 Donner le niveau (level) du dresseur

SQL

```
SELECT d.*, (SELECT level FROM xps WHERE d.xp BETWEEN xpmin AND xpmax) as level
  FROM dresseur d
  WHERE d.id = '29069011'
```

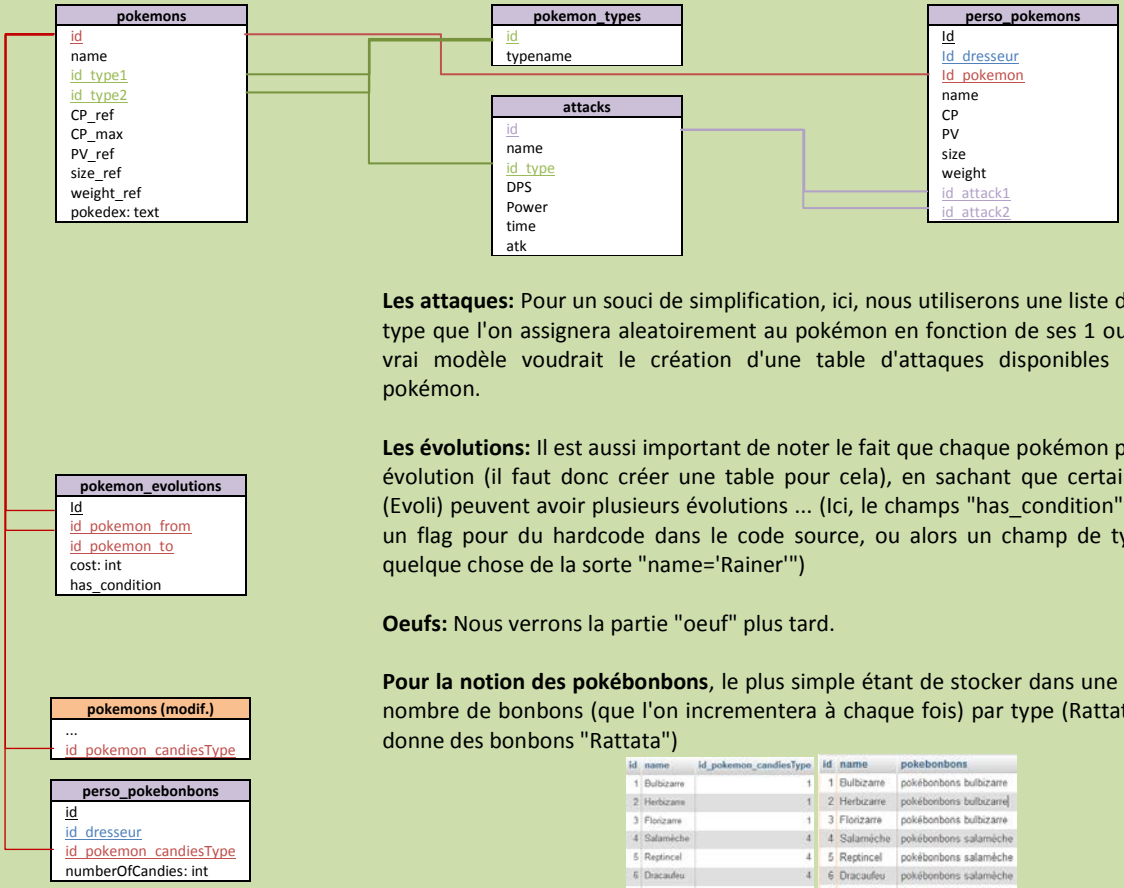
id	pseudo	password	hash	xp	id_team	money	skin_1	skin_2	skin_3	pstar	level
29069011	MatchouPitchou			35621	3	2000	1	2	1	45362	8

Les pokémons

Tables

Il nous faut une table listant tous les pokémons du jeu (un catalogue), puis une autre table regroupant les pokémons attrapés.

Chaque pokémon peut avoir certaines caractéritiques: un nom, une taille/PV moyenne, un nombre de CP, deux types distincts maximum, ... et une description du pokédex. Le plus simple serait d'avoir une table (catalogue) avec les statistiques moyennes, puis un pokémon spécifique généré par le serveur.



Les attaques: Pour un souci de simplification, ici, nous utiliserons une liste d'attaques par type que l'on assignera aleatoirement au pokémon en fonction de ses 1 ou 2 type(s). Le vrai modèle voudrait le création d'une table d'attaques disponibles pour chaque pokémon.

Les évolutions: Il est aussi important de noter le fait que chaque pokémon peut avoir une évolution (il faut donc créer une table pour cela), en sachant que certains pokémons (Evoli) peuvent avoir plusieurs évolutions ... (Ici, le champs "has_condition" pourrait être un flag pour du hardcode dans le code source, ou alors un champ de type text avec quelque chose de la sorte "name='Rainer'")

Oeufs: Nous verrons la partie "oeuf" plus tard.

Pour la notion des pokébonbons, le plus simple étant de stocker dans une autre table le nombre de bonbons (que l'on incrementera à chaque fois) par type (Rattata et Rattatac donne des bonbons "Rattata")

id	name	id_pokemon_candiesType
1	Bulbizarre	1
2	Herbizarre	1
3	Florizarre	1
4	Salamèche	4
5	Reptincel	4
6	Dracaufeu	4
7	Carapuce	7
8	Carabaffe	7
9	Tortank	7

id	name	pokebonbons
1	Bulbizarre	pokébonbons bulbizarre
2	Herbizarre	pokébonbons bulbizarre
3	Florizarre	pokébonbons bulbizarre
4	Salamèche	pokébonbons salamèche
5	Reptincel	pokébonbons salamèche
6	Dracaufeu	pokébonbons salamèche
7	Carapuce	pokébonbons carapuce
8	Carabaffe	pokébonbons carapuce
9	Tortank	pokébonbons carapuce

0201 Donner le nombre de pokémons attrapés (sac) et le nombre de pokémons dans des oeufs

SQL

```sql
  SELECT 'pokemons' as type, COUNT(id) as nb
    FROM perso_pokemons
    WHERE id_dresseur = '29069011'
UNION
  SELECT 'eggs' as type, COUNT(id) as nb -- Cette table est abordée plus tard
    FROM perso_eggs
    WHERE id_dresseur = '29069011'
```

type	nb
pokemons	32
eggs	4

Ici, utilisation de l'UNION pour l'exemple. Il est tout à fait possible de séparer les deux reqûetes et de les simplifier

0202 Lister tous les pokémons attrapés (nom, CP, vie)

SQL

```sql
SELECT * -- il est préférable de lister tous les champs que d'utiliser '*'
  FROM perso_pokemons
  WHERE id_dresseur = '29069011'
```

id	id_dresseur	name	PC	PV	id_pokemon	id_attack_1	id_attack_2	longitude	latitude
297685	29069011	Abo	46	16	23	1	3	0.000000	0.000000
297686	29069011	Evoli	529	65	133	45	29	0.000000	0.000000

0203 Donner tous les détails d'un pokémon selectionné (plusieurs reqûetes).

SQL

```sql
SELECT pp.name, pp.pc, pp.PV, atk1.name atk1name, atk2.name atk2name -- , pp.size, pp.weight, ...
  FROM perso_pokemons pp
    LEFT JOIN attacks atk1 ON atk1.id = pp.id_attack_1
    LEFT JOIN attacks atk2 ON atk2.id = pp.id_attack_2
  WHERE pp.id = '297685'
```

name	pc	PV	atk1name	atk2name
Abo	46	16	Charge	Acide

SQL

```sql
SELECT pe.id_to, pe.cost, pe.has_condition
  FROM pokemon_evolutions pe
    RIGHT JOIN perso_pokemons pp ON pe.id_from = pp.id_pokemon
  WHERE pp.id = '297686' -- Si résultat = le Pokémon peut évoluer
```

id_to	cost	has_condition
134	50	name="Rainer"
135	50	name="Sparky"
136	50	name="Pyro"

Evoli a trois évolutions.

0204 Donner le nombre de pokébonbons disponibles associés aux pokémons pour le dresseur

SQL

```sql
SELECT pb.id_dresseur, pb.id_pokemon_candiesType, pb.numberOfCandies, CONCAT('pokébonbons ', p.name) as texte
  FROM perso_pokebonbons pb
    LEFT JOIN pokemons p ON p.id = pb.id_pokemon_candiesType
  WHERE pb.id_dresseur = '29069011'
```

id_dresseur	id_pokemon_candiesType	numberOfCandies	texte
29069011	23	3	pokébonbons Abo
29069011	29	6	pokébonbons Nidoran?
29069011	32	3	pokébonbons Nidoran?
29069011	50	3	pokébonbons Taupiqueur
29069011	63	3	pokébonbons Abra
29069011	84	6	pokébonbons Doduo
29069011	88	3	pokébonbons Tadmorv
29069011	95	9	pokébonbons Onix

```sql
-- Pour avoir le nombre de pokébonbons pour un Pokémons, il faut vérifier son type, puis récupérer l'information
SELECT pb.id_dresseur, pb.id_pokemon_candiesType, pb.numberOfCandies, CONCAT('pokébonbons ', p.name) as texte
  FROM perso_pokebonbons pb
    LEFT JOIN pokemons p ON p.id = pb.id_pokemon_candiesType
    LEFT JOIN pokemons p2 ON p2.id_pokemon_candiesType = p.id
  WHERE pb.id_dresseur = '29069011' AND p2.id = '24'
-- #24 = Arbok
```

id_dresseur	id_pokemon_candiesType	numberOfCandies	texte
29069011	23	3	pokébonbons Abo

0205 Filtres: trier les pokémons du joueur par plusieurs filtres possibles (PV, PC, #, ...)

SQL

```sql
-- Sans filtre.
SELECT *
  FROM perso_pokemons
  WHERE id_dresseur = '29069011'

-- Trier par "pc", ordre décroissant (plus fort -> moins fort)
SELECT *
  FROM perso_pokemons
  WHERE id_dresseur = '29069011'
  ORDER BY pc DESC

-- Trier par "#" de pokémon, ordre croissant (1 -> 150)
SELECT *
  FROM perso_pokemons
  WHERE id_dresseur = '29069011'
  ORDER BY id_pokemon ASC

-- ...
```

Le pokédex

Tables

Pour le pokedex, nous avons déjà crée le catalogue de pokémons.
Maintenant, il reste simplement à créer une table pour le joueur, avec le status vu/attrapé.

La clé (id_dresseur, id_pokemon) est UNIQUE, ce qui permet de n'avoir su'une seule entrée par dresseur/pokémon.

perso_pokedex
id
id_dresseur
id_pokemon
is_view: boolean
is_catch: boolean

0301 Lister les 151 pokémons du jeu

SQL

```
SELECT *
  FROM pokemons
  ORDER BY id ASC
```

id	name	id_type1	id_type2	CP_ref	CP_max	PV_ref	size_ref	weight_ref	pokedex	fromegg
1	Bulbizarre	1	6	13.800	1071	4.000	0.730	6.880	La graine sur le dos de Bulbizarre est une réserve...	2
2	Herbizarre	1	6	21.600	1632	4.000	0.000	0.000		NULL
3	Florizarre	1	6	34.800	2580	4.000	0.000	0.000		NULL
4	Salamèche	2	NULL	12.200	955	4.000	0.000	0.000		2
5	Reptincel	2	NULL	20.500	1557	4.000	0.000	0.000		NULL
6	Dracaufeu	2	21	35.100	2602	4.000	0.000	0.000		NULL
7	Carapuce	3	NULL	13.000	1008	4.000	0.000	0.000		2
8	Carabaffe	3	NULL	20.900	1582	4.000	0.000	0.000		NULL
9	Tortank	3	NULL	34.300	2542	4.000	0.000	0.000		NULL
10	Chenipan	4	NULL	5.300	443	4.000	0.000	0.000		2
11	Chrysacier	4	NULL	5.800	477	4.000	0.000	0.000		NULL
12	Papilusion	4	21	19.100	1454	4.000	0.000	0.000		NULL
13	Aspicot	4	6	5.400	449	4.000	0.000	0.000		2

0302 La liste de tous les pokémons d'un type particulier

SQL

```
SELECT *
  FROM pokemons
  WHERE (id_type1 = 2 OR id_type2 = 2)
  ORDER BY id ASC -- type 2 = Feu
-- ou
SELECT p.*
  FROM pokemons p
   LEFT JOIN pokemon_types t ON p.id_type1 = t.id OR p.id_type2 = t.id
  WHERE t.typename = 'Fire'
```

id	name	id_type1	id_type2	CP_ref	CP_max	PV_ref	size_ref	weight_ref	pokedex	fromegg
4	Salamèche	2	NULL	12.200	955	4.000	0.000	0.000		2
5	Reptincel	2	NULL	20.500	1557	4.000	0.000	0.000		NULL
6	Dracaufeu	2	21	35.100	2602	4.000	0.000	0.000		NULL
37	Goupix	2	NULL	10.600	831	4.000	0.000	0.000		5
38	Feunard	2	NULL	29.300	2188	4.000	0.000	0.000		NULL
58	Caninos	2	NULL	17.500	1335	4.000	0.000	0.000		5
59	Arcanin	2	NULL	40.500	2983	4.000	0.000	0.000		NULL
77	Ponyta	2	NULL	20.000	1516	4.000	0.000	0.000		5
78	Galopa	2	NULL	29.500	2199	4.000	0.000	0.000		NULL
126	Magmar	2	NULL	30.400	2265	4.000	0.000	0.000		10
136	Pyroli	2	NULL	35.600	2643	4.000	0.000	0.000		NULL
146	Sulfura	2	21	44.100	3240	4.000	0.000	0.000		NULL

0303 Donner le #nombre du pokémon attrapé/vue le plus loin dans la liste des pokémons

SQL

```
SELECT MAX(pp.id_pokemon)
  FROM pokemons p
    LEFT JOIN perso_pokemons pp ON p.id = pp.id_pokemon
  WHERE pp.id_dresseur = '29069011'
  GROUP BY pp.id_dresseur -- Il faut le GROUP BY pour utiliser le MAX
```

MAX(pp.id_pokemon)
143

0304 Dire si un pokémon a été attrapé ou vu.

SQL

```
SELECT id_pokemon, is_view, is_catch
  FROM perso_pokedex
  WHERE id_dresseur = '29069011'
    AND id_pokemon = '100' -- #100 Voltobre, #19 Rattata
```

id_pokemon	is_view	is_catch
100	1	1

id_pokemon	is_view	is_catch
19	1	0

0305 **Rattata a été attrapé, l'inscrire au pokédex**

SQL
```
INSERT INTO perso_pokedex
  SET id_pokemon = '19', id_dresseur = '29069011', is_view = 1, is_catch = 1
  ON DUPLICATE KEY UPDATE is_view = 1, is_catch = 1
```

0306 **Le pokédex d'un pokémon particulier**

SQL
```
SELECT pokedex
  FROM pokemons
  WHERE id = '1' -- id = # du pokemon
```

pokedex
La graine sur le dos de Bulbizarre est une réserve de nutriments. Il la fait doucement pousser en absorbant les rayons du soleil, et emmagasine de l'énergie dedans en vue de son évolution. S'il dépense trop souvent son énergie, il n'évoluera pas car l'ouverture de son bulbe dépend de ses réserves de nutriments.

SQL
```
SELECT p.pokedex
  FROM perso_pokemons pp
    LEFT JOIN pokemons p ON p.id = pp.id_pokemon
  WHERE pp.id = '297685' -- id du Pokémon (généré par le serveur et non pas son numéro)
```

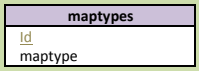
pokedex
Abo peut ravager toute une colonie de Piafabec ou de Roucool rien qu'en dévorant ses œufs ou ses petits. Quand il s'endort, il s'enroule de façon à laisser sa tête au dessus, pour pouvoir mordre à tout moment. Son poison n'est pas mortel, mais il provoque une terrible sensation de brûlure, comme le venin d'une méduse.

Les pokestops

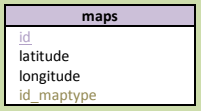
Tables

Les **Pokéstops** et autres lieux sont basés sur des coordonnées GPS (latitude et longitude). Il existe plusieurs manières de stocker ces points particuliers sur la carte.
Par exemple, une table "maps" contenant les cardinalités, et les informations de l'arène ou du pokestop dedans. Un champs "type" serait là pour donner le type de lieu.

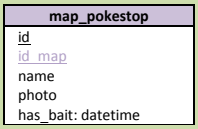
Ici, on peut aussi créer une table de coordonnées, et deux autres tables pour les lieux.



id	maptype
1	Pokestop
2	Arène
3	Source



id	latitude	longitude	id_maptype
1	48.560721	7.694761	1
2	48.573019	7.763940	2



id	id_map	name	photo	has_bait (null = no bait, datetime for end of bait life)
1	1	CESI	fr/67400/cesi.png	2016-07-29 12:00:00

Pour utiliser les coordonnées GPS, rien de mieux que de se renseigner sur internet:

- http://www.phpsources.org/calcul-de-la-distance-entre-deux-coordonnees-gps-avec-mysql_101.html
- http://www.primuscreation.fr/comment-lister-des-points-geolocalises-a-proximite-de-coordonnees-gps-de-referenc/
- http://www.developpez.net/forums/d1191451/bases-donnees/langage-sql/latitude-longitude-plus-proches/
- ...

0401 — Lister les pokestops disponibles dans la base de donnée. On pourrait se baser sur les coordonnées GPS pour ne lister que ceux disponibles dans un rayon de X m/km

SQL

```
SELECT *
  FROM maps
  WHERE id_maptype = 1 -- 1 = pokestop, 2 = arene
   AND (
    6366 *
    acos(
      cos(radians(48.560719)) -- $latitude_ref
      *
      cos(radians(`latitude`))
      *
      cos(radians(`longitude`) - radians(7.694731)) -- $longitude_ref
      +
      sin(radians(48.560719)) * sin(radians(`latitude`)) -- $latitude_ref
    )
  ) <= 2 -- rayon en Km, ici 2km
```

id	latitude	longitude	id_maptype
1	48.560721	7.694761	1

0402 — Donner les caractéristiques du pokéstop sélectionné

SQL

```
SELECT *
  FROM map_pokestop
  WHERE id_map = '1' -- Id=1, identifiant du pokestop = maps.id
```

id	id_map	name	photo	has_bait (null = no bait, datetime for end of bait life)
1	1	CESI	fr/67400/cesi.png	2016-07-29 12:00:00

Arènes

Tables

Pour les **arénès**, ici aussi on se base sur le même processus pour les autres lieux.

map_arena
id
id_map
name
photo
id_team
xp

Par contre, l'arène doit contenir des pokémons, tous de la même équipe. L'équipe pourrait être calculé via le 1er pokémon placé (REQ#0302) ou directement stocké dans la table.

0501 Quels sont les pokemons dans l'arène selectionnée ? Donner le dresseur, et sa team ainsi que le nom du pokémon et son nombre de PC.

SQL

```
SELECT d.pseudo, d.xp, d.id_team, pp.id, pp.name, pp.pc, pp.pv
  FROM map_arena ma
    LEFT JOIN arena_pokemons ap ON ap.id_maparena = ma.id
    LEFT JOIN perso_pokemons pp ON ap.id_persopokemon = pp.id
    LEFT JOIN dresseur       d  ON d.id = pp.id_dresseur
  WHERE ma.id_map = '2'
  ORDER BY pp.pc ASC
```

pseudo	xp	id_team	id	name	pc	pv
MatchouPitchou	35621	3	297685	Abo	46	16
MatchouPitchou	35621	3	297686	Evoli	529	65

0502 Selectionner l'équipe associé à l'arène

SQL

```
-- Méthode simple, si on stock l'information dans la table
SELECT ma.id_map, ma.id_team, t.name, t.color
  FROM map_arena ma
    LEFT JOIN teams t ON t.id = ma.id_team
  WHERE ma.id_map = '2'
```

id_map	id_team	name	color
2	1	Bleu	blue

SQL

```
SELECT DISTINCT t.*
  FROM map_arena ma
    LEFT JOIN arena_pokemons ap ON ap.id_maparena = ma.id
    LEFT JOIN perso_pokemons pp ON ap.id_persopokemon = pp.id
    LEFT JOIN dresseur       d  ON d.id = pp.id_dresseur
    LEFT JOIN teams          t  ON t.id = d.id_team -- Pour avoir des informations sur l'équipe
  WHERE ma.id_map = '2'
```

id	name	color
3	Rouge	red

0503 Selectionner les 6 pokémons les plus puissant du joueur

SQL

```
SELECT pp.*
  FROM perso_pokemons pp
  WHERE pp.id_dresseur = '29069011'
  ORDER BY pp.PC DESC
  LIMIT 0, 6
```

id	id_dresseur	name	PC	PV	id_pokemon	id_attack_1	id_attack_2	longitude	latitude
297738	29069011	Scarabrute	564	31	127	41	35	0.000000	0.000000
297686	29069011	Evoli	529	65	133	45	29	0.000000	0.000000
297735	29069011	Scarabrute	518	39	127	41	35	0.000000	0.000000
297711	29069011	Kicklee	495	52	106	5	10	0.000000	0.000000
297709	29069011	Tadmorv	454	71	88	19	18	0.000000	0.000000
297727	29069011	Saquedeneu	369	42	114	28	2	0.000000	0.000000

0504 Retrouver les caractéristiques des deux pokémons en train de combattre (nom, pv, pc). Lister aussi les deux attaques disponibles

SQL

```
SELECT pp.*, atk1.name,
       atk1.type atk1type, atk1.DPS atk1DPS, atk1.power atk1power, atk1.time atk0power, atk1.atk atk1atk,
       atk2.type atk2type, atk2.DPS atk2DPS, atk1.power atk2power, atk2.time atk1power, atk2.atk atk2atk
  FROM perso_pokemons pp
    LEFT JOIN attacks atk1 ON atk1.id = id_attack_1
    LEFT JOIN attacks atk2 ON atk2.id = id_attack_2
  WHERE pp.id IN('1', ' 297685')
```

id	id_dresseur	name	PC	PV	id_pokemon	id_attack_1	id_attack_2	longitude	latitude	name	atk1type	atk1DPS	atk1power	atk0power	atk1atk	name	atk2type	atk2DPS	atk2power	atk1power	atk2atk
1	1	MonBulby	14	12	1	1	2	0.000000	0.000000	Charge	5	0.00	12	0.000	0	Canon Graine	1	0.00	12	0.000	0
297685	29069011	Abo	46	16	23	1	3	0.000000	0.000000	Charge	5	0.00	12	0.000	0	Acide	6	9.52	12	1.050	200

Boutique

Tables

Qui dit **boutique**, dit argent. Il faut donc que notre joueur puisse avoir accès à une monnaie virtuelle.

dresseurs
id
...
money
...

La boutique est aussi composée d'objets uniques (pokéballs, encens, oeufs chance, ...), et d'un catalogue (20/100/400 pokéballs, ...). Nous allons donc créer plusieurs tables. De plus, le dresseur devra aussi avoir ses propres objets.

objects
id
name

shops
id
id_object (null)
name
nb
price: float(6, 2)
is_real_money:boolean

perso_objects
id
id_object
id_dresseur
number

Toute la logique d'achat (argent réel), se fera dans le code. Il faut aussi pour le catalogue, faire la distinction entre ce qui peut s'acheter avec de l'argent ou avec de la monnaie virtuelle : pour cela, on peut stocker un flag (is_real_money).
Au niveau de la table perso_objects, il faudra penser à incrementer le nombre d'objet si celui-ci est déjà dans la table.

0601 Donner le nombre de poképièces du joueur

SQL

```
SELECT money
  FROM dresseur
  WHERE id = '29069011'
```

money
150

0602 Lister tous les articles disponibles dans la boutique (objet, prix)

SQL

```
SELECT id, id_object, nb, name, price, is_real_money
  FROM shops
  ORDER BY is_real_money ASC, id_object, nb
```

id	id_object	nb	name	price	is_real_money
1	1	20	20 poké balls	100.00	0
2	1	100	100 poké balls	400.00	0
3	1	200	200 poké balls	800.00	0
4	6	1	Oeuf chance	80.00	0
7	6	8	8 Oeufs Chance	500.00	0
8	6	25	25 Oeufs Chance	1250.00	0
5	*NULL*	100	100 poképièces	0.99	1
6	*NULL*	500	500 poképièces	4.99	1

0603 Acheter 8 Oeufs Chance, les ajouter au dresseur

SQL

```
INSERT INTO perso_objects
  SET id_dresseur = '29069011', id_object = 6, number = 8
  ON DUPLICATE KEY UPDATE
    number = number + 8

-- Ou:
INSERT INTO perso_objects (id_dresseur, id_object, number)
  SELECT '29069011' as id_dresseur, id_object, nb
    FROM shops
    WHERE id = 7 -- #7 = 8 Oeufs Chances, pour 500 poképièces
  ON DUPLICATE KEY UPDATE
    number = number + nb

-- Sans oublier d'enlever l'argent au joueur :)
UPDATE dresseur SET money = money - 500 WHERE id = '29069011'

-- Ou:
UPDATE dresseur
  SET money = money - (SELECT price FROM shops WHERE id = 7)
  WHERE id = '29069011'

-- /!\ il faudra vérifier qu'il reste de l'argent dans le code source avant de faire l'achat
```

0604 Acheter 500 poképièces, les ajouter au dresseur

SQL

```
UPDATE dresseurs SET money = money + 500 WHERE id = '29069011'

-- Ou (le payement se fait au niveau de code source):
UPDATE dresseur
  SET money = money + (SELECT nb FROM shops WHERE id = 6)
  WHERE id = '29069011'
```

id	id_object	nb	name	price	is_real_money
1	1	20	20 poké balls	100.00	0
2	1	100	100 poké balls	400.00	0
3	1	200	200 poké balls	800.00	0
4	6	1	Oeuf chance	80.00	0
7	6	8	8 Oeufs Chance	500.00	0
8	6	25	25 Oeufs Chance	1250.00	0
5	*NULL*	100	100 poképièces	0.99	1
6	*NULL*	500	500 poképièces	4.99	1

Les médailles

Tables

Pour le système de médailles, rien de nouveau: on a une table "catalogue" et une table associée au joueur.

La table perso_rewards à pour but de stocker chaque étapes des médailles du joueur: quand celui-ci attrapera un pokémon de type "plante", alors on ajoutera dans cette table une ligne dans cette table pour le dresseur ; si cette ligne existe déjà, il faudra alors incrémenter la valeur.

Afin de pouvoir calculer les niveau, il y a ici deux méthodes: tout est stocké dans une table (rewarditems) avec un min/max et il suffira de faire un perso_rewards.number BETWEEN rewarditems.imin AND rewarditems.imax pour connaitre le niveau de chaque médaille ; ou alors faire ce calcul dans le code source.



0701 Lister les médailles disponibles dans le jeu

SQL

```
SELECT r.id, r.name, r.description
  FROM rewards r
```

id	name	description
1	Jogger	Marcher une certaine distance
2	Kanto	Enregistrer des Pokémon dans le Pokédex
3	Collectionneur	Capturer des Pokémon
4	Scientifique	Evoluer des Pokémon
5	Éleveur	Faire éclore des oeufs
6	Randonneur	Visiter des PokéStops
7	Pêcheur	Capturer des Magicarpe taille Large
8	Combattante	Gagner des matchs d'arène
9	Top Dresseur	S'Entraîner à une arène que vous contrôlez
10	Élève	Capturer des Pokémon de type Normal
11	Karatéka	Capturer des Pokémon de type Combat
12	Ornithologue	Capturer des Pokémon de type Vol
13	Vaurienne	Capturer des Pokémon de type Poison
14	Ruinemaniac	Capturer des Pokémon de type Sol
15	Hiker	Capturer des Pokémon de type Roche
16	Scout	Capturer des Pokémon de type Insecte
17	Mystimaniac	Capturer des Pokémon de type Spectre

0702 Donner pour chaque médaille, le niveau pour le joueur (= nombre de pokémon/actions, une formule mathématique peut être utilisé pour savoir si c'est le niveau 1, 2 ou 3)

SQL

```
SELECT r.name, pr.number
  FROM rewards r
    LEFT JOIN perso_rewards pr ON pr.id_reward = r.id
  WHERE pr.id_dresseur = '29069011'
```

name	number
Jogger	12.234
Top Dresseur	2.000

SQL

```
-- Plus complexe: récupération du niveau avec les min/max

SELECT pr.*, r.name,
     (
       SELECT level
         FROM rewarditems ri
         WHERE ri.id_reward = pr.id_reward
           AND pr.number BETWEEN imin AND imax
     ) as level
  FROM perso_rewards pr
    LEFT JOIN rewards r ON r.id = pr.id_reward
  WHERE pr.id_dresseur = '29069011'
```

id	id_reward	id_dresseur	number	name	level
1	1	29069011	12.234	Jogger	1
2	9	29069011	2.000	Top Dresseur	0

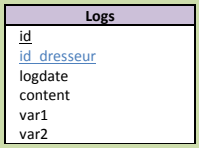
Le Journal

Tables

La journalisation est une tache plutôt facile : une table, des informations. Par contre, le point le plus compliqué est de savoir comment stocker ces inforamtions et comment les traiter.

Nous pourrions très bien enregistrer des phrases du genre "Rattata a été attrapé, CP 126". Mais si une évolution doit être apporté à l'interface, ou si des analyses plus poussées doivent être faites, alors la solution a plus simple est de rendre le tout plus dynamique.

Si on enregistre des types d'actions (1=pokémon attrapé, 2=pokéstop, 3=oeuf, ...) et que l'on laisse des champs comme variables (pour type1, var1 = id pokémon, var2 = nb de PC ; ???), alors on peut gérer le tout à l'affichage.



0801 Inserer une entrée dans le journal

SQL

```
INSERT INTO logs
  SET id_dresseur = '29069011', logdate = NOW(), content = '1', var1 = '19', var2 = '96'
  -- #1 = pokémon attrapé, #19 = Rattata, 96 CP

INSERT INTO logs
  SET id_dresseur = '29069011', logdate = NOW(), content = '2', var1 = '2'  -- #2 = pokestop, #2 = Superball
```

0802 Lister les 25 dernières actions du journal.

SQL

```
SELECT logdate, content, var1, var2
  FROM logs
  WHERE id_dresseur = '29069011'
```

logdate	content	var1	var2
2016-08-27 00:53:25	1	19	89
2016-08-27 00:53:38	2	2	
2016-08-27 00:53:38	2	2	

SQL

```
-- BONUS: Préparer le texte de sortie
SELECT id, logdate,
  CASE content
    WHEN 1 THEN CONCAT((SELECT p.name FROM pokemons p WHERE p.id = var1), ' was caught!!\nCP ', var2)
    WHEN 2 THEN CONCAT('Received ', (SELECT o.name FROM objects o WHERE o.id = var1), ' from Pokéstop.')
    ELSE content
  END as contenttext, var1, var2
  FROM logs
  WHERE id_dresseur = '29069011'
  ORDER BY logdate DESC
  LIMIT 0, 25
```

id	logdate	contenttext	var1	var2
2	2016-08-27 00:53:38	Received Superball from Pokéstop.	2	
3	2016-08-27 00:53:38	Received Superball from Pokéstop.	2	
1	2016-08-27 00:53:25	Rattata was caught!! CP89	19	89

Les œufs

Tables

Pour les œufs, nous allons modifier la table "pokemons" en ajoutant la liste des Pokémons pouvant sortir d'un œuf, de Km particuliers:

```
ALTER TABLE `pokemons` ADD `fromegg` SMALLINT NULL DEFAULT NULL AFTER `pokedex`;
```

id	name	fromegg
1	Bulbizarre	2
2	Herbizarre	NULL
3	Florizarre	NULL
4	Salamèche	2

Ensuite, il faut créer une table pour **sauvegarder les œufs** en notre procession et leur état (sous incubateur ou non, nombre de Km, ...). Une requête SQL se chargera de faire éclore les œufs où la distance parcourue (kmdone) > au nombre de km à parcourir (km = 2/5/10)

perso_eggs
id
id_dresseur
km
kmdone
hasincubator: boolean

Afin d'éviter la triche, **les Km parcourus sont incrémentés coté serveur**. Nous stockerons donc la position GPS du dresseur à intervalles réguliers, puis le serveur se chargera d'analyser si la distance est valide (triche, ou < 18 Km/h) et d'incrémenter les œufs sous incubateurs.

perso_gps
id
id_dresseur
posdate
longitude
latitude
calculated_dist
calculated_vit

0901 Lister les œufs disponibles pour le joueur. Donner ceux en cours d'incubation et leur état en terme de Km.

SQL

```
SELECT *
  FROM perso_eggs
  WHERE id_dresseur = '29069011'
```

id	id_dresseur	km	kmdone	hasincubator
1	29069011	5	2.456	1
2	29069011	2	0.000	0
3	29069011	10	10.012	1

0902 Lister les œufs du joueur à faire éclore

SQL

```
SELECT *
  FROM perso_eggs
  WHERE id_dresseur = '29069011' AND kmdone >= km AND hasincubator = true
```

id	id_dresseur	km	kmdone	hasincubator
3	29069011	10	10.012	1

0903 Lister les pokemons pouvant sortir des oeufs, triés par Km

SQL

```
SELECT *
  FROM pokemons
  WHERE fromegg > 0
  ORDER BY fromegg, id
```

id	name	id_type1	id_type2	CP_ref	PV_ref	size_ref	weight_ref	pokedex	fromegg
1	Bulbizarre	1	6	10.000	15.000	0.730	6.880	La graine sur le dos de Bulbizarre est une réserve...	2
4	Salamèche	2	NULL	0.000	0.000	0.000	0.000		2
7	Carapuce	3	NULL	0.000	0.000	0.000	0.000		2
10	Chenipan	4	NULL	0.000	0.000	0.000	0.000		2
13	Aspicot	4	6	0.000	0.000	0.000	0.000		2
16	Roucool	5	21	0.000	0.000	0.000	0.000		2
19	Rattata	5	NULL	0.000	0.000	0.000	0.000		2
21	Piafabec	5	21	0.000	0.000	0.000	0.000		2
25	Pikachu	7	NULL	0.000	0.000	0.000	0.000		2
39	Rondoudou	5	9	0.000	0.000	0.000	0.000		2
41	Nosferapti	6	21	0.000	0.000	0.000	0.000		2
74	Racaillou	13	8	0.000	0.000	0.000	0.000		2
129	Magicarpe	18	NULL	0.000	0.000	0.000	0.000		2
29	Nidoran?	6	NULL	0.000	0.000	0.000	0.000		5

0904 Donner le Pokémon pouvant sortir de l'œuf une fois éclos (se baser sur une liste pour 2/5/10 et utiliser une fonction "aléatoire")

```
SELECT DISTINCT pe.*, p.*
  FROM perso_eggs pe,
       (SELECT * FROM pokemons p ORDER BY RAND()) as p
  WHERE pe.id_dresseur = '29069011' AND pe.kmdone >= pe.km AND p.fromegg = pe.km
  GROUP BY pe.id_dresseur
```

id	id_dresseur	km	kmdone	hasincubator	id	name	id_type1	id_type2	CP_ref	CP_max	PV_ref	size_ref	weight_ref	pokedex	fromegg	id_pokemon_candiesType
3	29069011	10	10.012	1	124	Lippoutou	14	12	22.800	1716	4.000	0.000	0.000		10	124

id	id_dresseur	km	kmdone	hasincubator	id	name	id_type1	id_type2	CP_ref	CP_max	PV_ref	size_ref	weight_ref	pokedex	fromegg	id_pokemon_candiesType
3	29069011	10	10.012	1	123	Insécateur	17	21	27.700	2073	4.000	0.000	0.000		10	123

SQL

```sql
-- BONUS: inserer dans la liste des pokemons du dreseurs, avec les bonnes informations = faire eclore un oeuf
INSERT INTO perso_pokemons (id_dresseur, name, PC, PV, id_pokemon, id_attack_1, id_attack_2)
  SELECT pe.id_dresseur, p.name, p.CP_ref * 22 * (.25+RAND()), p.PV_ref * 22 * (.1+RAND()), p.id, atk1.id atk1id,
atk2.id atk2id
    -- '22' etant le niveau du joueur
    FROM perso_eggs pe,
     (SELECT * FROM pokemons ORDER BY RAND()) as p,
     (SELECT id, type FROM attacks ORDER BY RAND()) as atk1,
     (SELECT id, type FROM attacks ORDER BY RAND()) as atk2
    WHERE pe.id_dresseur = '29069011'
      AND pe.kmdone >= pe.km
      AND p.fromegg = pe.km
      AND atk1.type = p.id_type1
      AND atk2.type IN (p.id_type1, p.id_type2)
      AND atk1.id <> atk2.id
    GROUP BY pe.id_dresseur
```

id	id_dresseur	km	kmdone	hasincubator	id	name	id_type1	id_type2	CP_ref	PV_ref	size_ref	weight_ref	pokedex	fromegg
3	29069011	10	10.012	1	133	Évoli	5	*NULL*	0.000	0.000	0.000	0.000		10

id	id_dresseur	km	kmdone	hasincubator	id	name	id_type1	id_type2	CP_ref	PV_ref	size_ref	weight_ref	pokedex	fromegg
3	29069011	10	10.012	1	126	Magmar	2	*NULL*	0.000	0.000	0.000	0.000		10

0905

Ajouter une certaine distance parcourue (entre deux points GPS) aux œufs dans l'incubateur et ajouter cette même distance dans la médaille associée.

```sql
-- Pour cela, il ne faut pas oublier d'enregistrer la dernière position GPS du dresseur
INSERT INTO perso_gps
  SET id_dresseur = '29069011', posdate = NOW(), lastpos_lat = '...', lastpos_lon = '...'
```

id	id_dresseur	posdate	longitude	latitude	calculated_dist	calculated_vit
1	29069011	2016-08-28 00:13:25	7.710372	48.516312	0.0188	1.578
2	29069011	2016-08-28 00:14:08	7.710409	48.516479	0.3107	21.510
3	29069011	2016-08-28 00:15:00	7.709646	48.519230	7.1877	995.222
4	29069011	2016-08-28 00:15:26	7.763940	48.573017	*NULL*	*NULL*

SQL

```sql
-- Ensuite le serveur va calculer la distance entre deux points. On prend alors pour chaque enregistrement la
positiond 'après et on calcul en fonction des coordonnées GPS. Reste ensuite à faire un UPDATE pour modifier le champ
'calculated_dist'
SELECT pg.id, pg.*, pg1.*, 6366 * 2 *
    atan2(
      sqrt(
        SIN((RADIANS(pg1.latitude) - RADIANS(pg.latitude)) / 2) * SIN((RADIANS(pg1.latitude) - RADIANS(pg.latitude)) /
2) + COS(RADIANS(pg.latitude)) * COS(RADIANS(pg1.latitude)) * SIN((RADIANS(pg1.longitude) - RADIANS(pg.longitude)) /
2) * SIN((RADIANS(pg1.longitude) - RADIANS(pg.longitude)) / 2)
      ),
      sqrt(
        1 - SIN((RADIANS(pg1.latitude) - RADIANS(pg.latitude)) / 2) * SIN((RADIANS(pg1.latitude) -
RADIANS(pg.latitude)) / 2) + COS(RADIANS(pg.latitude)) * COS(RADIANS(pg1.latitude)) * SIN((RADIANS(pg1.longitude) -
RADIANS(pg.longitude)) / 2) * SIN((RADIANS(pg1.longitude) - RADIANS(pg.longitude)) / 2)
      )
    ) as calculated_distance_final_km,
    (UNIX_TIMESTAMP(pg1.posdate)-UNIX_TIMESTAMP(pg.posdate)) as calculated_diff_seconds
  FROM perso_gps pg
    INNER JOIN perso_gps pg1 ON pg.id_dresseur = pg1.id_dresseur
      AND pg1.posdate = (SELECT MIN(posdate) FROM perso_gps WHERE id_dresseur = pg1.id_dresseur AND posdate >
pg.posdate)
  WHERE pg.calculated_dist IS NULL
```

id	id	id_dresseur	posdate	longitude	latitude	calculated_dist	calculated_vit	id	id_dresseur	posdate	longitude	latitude	calculated_dist	calculated_vit	calculated_distance_final_km	calculated_diff_seconds
1	1	29069011	2016-08-28 00:13:25	7.710372	48.516312	0.0188	*NULL*	2	29069011	2016-08-28 00:14:08	7.710409	48.516479	0.3107	*NULL*	0.018848880701467355	43
2	2	29069011	2016-08-28 00:14:08	7.710409	48.516479	0.3107	*NULL*	3	29069011	2016-08-28 00:15:00	7.709646	48.519230	7.1877	*NULL*	0.3107058082530724	52
3	3	29069011	2016-08-28 00:15:00	7.709646	48.519230	7.1877	*NULL*	4	29069011	2016-08-28 00:15:26	7.763940	48.573017	0.0000	*NULL*	7.187717600323826	26

On récupère la distance en Km entre les points parcourus, pour chaque dresseur, qui ne sont aps encore calculés

Avec un excel:

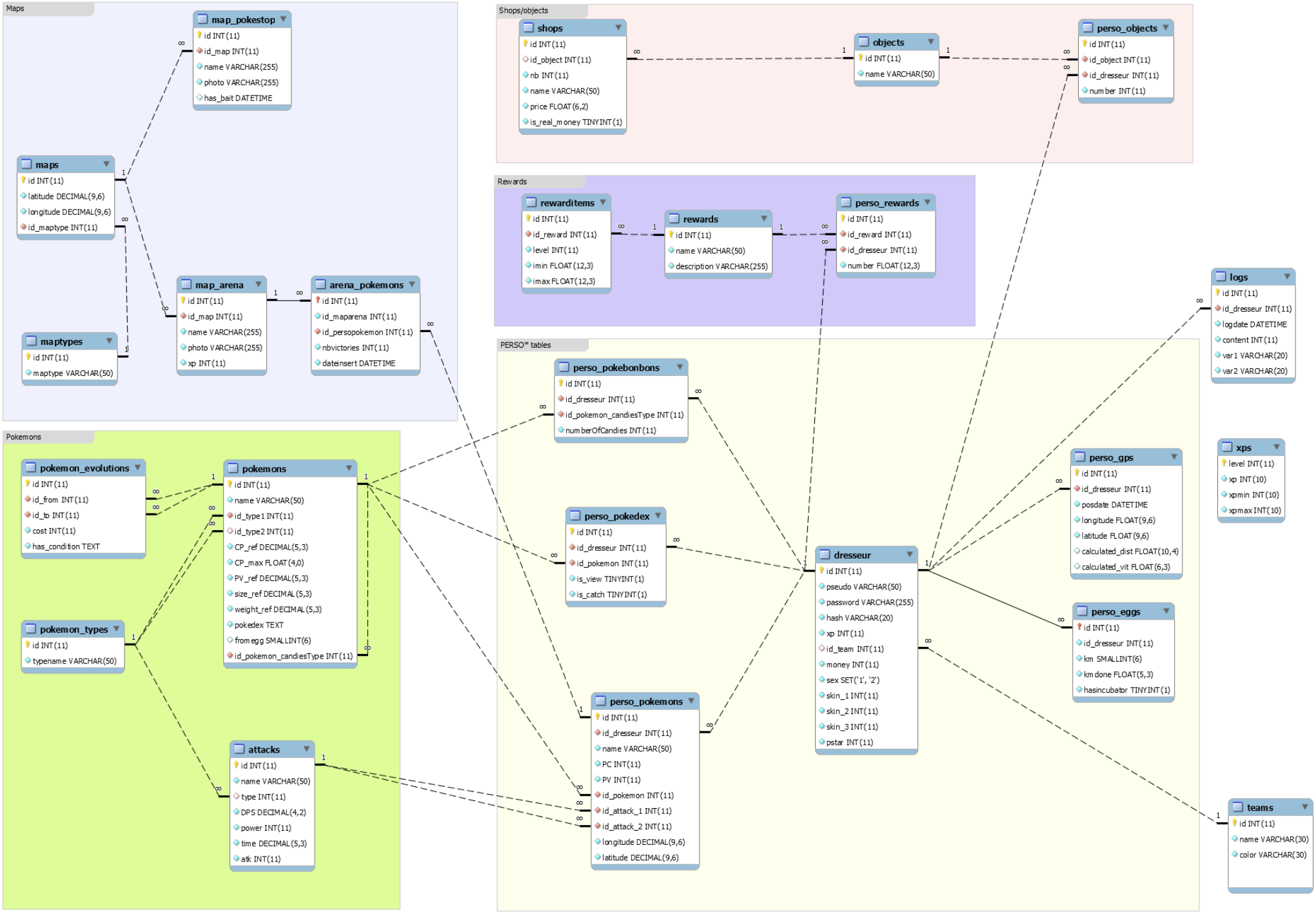
id	calc_dist	calc_sec	calc_sec_h	calc_vit_kmh	
1	0.01884888	43	0.011944444	1.578045826	<18 km/h OK
2	0.31070581	52	0.014444444	21.51040211	>18 km/h
3	7.1877176	26	0.007222222	995.222437	/!\ Triche !

```
[PSEUDO CODE]

$Results = ExecuteQuery('SELECT pg.id, pg.*, pg1.*, 6366 * 2 * [...]')
Foreach $Results => $Result
  $Distance = $Result['calculated_distance_final_km']
  /* On calcul la vitesse en Km/s */
  $Vitesse = $Distance / ($Result['calculated_diff_seconds']/3600)

  ExecuteQuery('UPDATE perso_gps SET calculated_dist = :Result-calculated_distance_final_km, calculated_vit = :Vitesse
WHERE id = :Result-id')

  If( $Vitesse < 18 ) /* 18 Km/h */
    /* On ajoute au compeur de chaque oeufs */
    ExecuteQuery('UPDATE perso_eggs SET number = number + :Distance WHERE id_dresseur = :Result-id_dresseur AND
hasincubator = true')
    /* On ajoute la médaille */
    ExecuteQuery('INSERT INTO perso_rewards SET id_dresseur = :Result-id_dresseur, id_reward = 1, number = :Distance
ON DUPLICATE KEY UPDATE number = number + :Distance')
  Else If( $Vitesse > 200 )
    SendReportToSupportTeam()
  EndIf
endforeach
```


Maps
map_pokestop
id INT(11)
id_map INT(11)
name VARCHAR(255)
photo VARCHAR(255)
has_bait DATETIME
maps
id INT(11)
latitude DECIMAL(9,6)
longitude DECIMAL(9,6)
id_maptype INT(11)
map_arena
id INT(11)
id_map INT(11)
name VARCHAR(255)
photo VARCHAR(255)
xp INT(11)
arena_pokemons
id INT(11)
id_maparena INT(11)
id_persopokemon INT(11)
nbvictories INT(11)
dateinsert DATETIME
maptypes
id INT(11)
maptype VARCHAR(50)
Shops/objects
shops
id INT(11)
id_object INT(11)
nb INT(11)
name VARCHAR(50)
price FLOAT(6,2)
is_real_money TINYINT(1)
objects
id INT(11)
name VARCHAR(50)
perso_objects
id INT(11)
id_object INT(11)
id_dresseur INT(11)
number INT(11)
Rewards
rewarditems
id INT(11)
id_reward INT(11)
level INT(11)
imin FLOAT(12,3)
imax FLOAT(12,3)
rewards
id INT(11)
name VARCHAR(50)
description VARCHAR(255)
perso_rewards
id INT(11)
id_reward INT(11)
id_dresseur INT(11)
number FLOAT(12,3)
PERSO* tables
perso_pokebonbons
id INT(11)
id_dresseur INT(11)
id_pokemon_candiesType INT(11)
numberOfCandies INT(11)
perso_pokedex
id INT(11)
id_dresseur INT(11)
id_pokemon INT(11)
is_view TINYINT(1)
is_catch TINYINT(1)
perso_pokemons
id INT(11)
id_dresseur INT(11)
name VARCHAR(50)
PC INT(11)
PV INT(11)
id_pokemon INT(11)
id_attack_1 INT(11)
id_attack_2 INT(11)
longitude DECIMAL(9,6)
latitude DECIMAL(9,6)
dresseur
id INT(11)
pseudo VARCHAR(50)
password VARCHAR(255)
hash VARCHAR(20)
xp INT(11)
id_team INT(11)
money INT(11)
sex SET('1', '2')
skin_1 INT(11)
skin_2 INT(11)
skin_3 INT(11)
pstar INT(11)
logs
id INT(11)
id_dresseur INT(11)
logdate DATETIME
content INT(11)
var1 VARCHAR(20)
var2 VARCHAR(20)
perso_gps
id INT(11)
id_dresseur INT(11)
posdate DATETIME
longitude FLOAT(9,6)
latitude FLOAT(9,6)
calculated_dist FLOAT(10,4)
calculated_vit FLOAT(6,3)
xps
level INT(11)
xp INT(10)
xpmin INT(10)
xpmax INT(10)
perso_eggs
id INT(11)
id_dresseur INT(11)
km SMALLINT(6)
km done FLOAT(5,3)
hasincubator TINYINT(1)
teams
id INT(11)
name VARCHAR(30)
color VARCHAR(30)
Pokemons
pokemon_evolutions
id INT(11)
id_from INT(11)
id_to INT(11)
cost INT(11)
has_condition TEXT
pokemons
id INT(11)
name VARCHAR(50)
id_type1 INT(11)
id_type2 INT(11)
CP_ref DECIMAL(5,3)
CP_max FLOAT(4,0)
PV_ref DECIMAL(5,3)
size_ref DECIMAL(5,3)
weight_ref DECIMAL(5,3)
pokedex TEXT
from egg SMALLINT(6)
id_pokemon_candiesType INT(11)
pokemon_types
id INT(11)
typename VARCHAR(50)
attacks
id INT(11)
name VARCHAR(50)
type INT(11)
DPS DECIMAL(4,2)
power INT(11)
time DECIMAL(5,3)
atk INT(11)

D'autres requêtes (à enlever de la version finale)

```
INSERT INTO perso_pokedex (id_pokemon, id_dresseur, is_view, is_catch)
  SELECT id_pokemon, id_dresseur, 1, 1
    FROM perso_pokemons
ON DUPLICATE KEY UPDATE is_view = 1, is_catch = 1
```

```
SELECT p.id, p.name, IFNULL((
    SELECT LEAST(MIN(pe.id_from), MIN(pe2.id_from))
      FROM pokemon_evolutions pe
        JOIN pokemon_evolutions pe2 ON pe.id_from = pe2.id_to OR pe.id_to =
pe2.id_from
      WHERE pe.id_from = p.id OR pe.id_to = p.id
  ), (
    SELECT MIN(pe.id_from)
      FROM pokemon_evolutions pe
        -- JOIN pokemon_evolutions pe2 ON pe.id_from = pe2.id_to OR pe.id_to =
pe2.id_from
      WHERE pe.id_from = p.id OR pe.id_to = p.id
  )) as id_evol
  FROM pokemons p
```

id	name	id_evol
1	Bulbizarre	1
2	Herbizarre	1
3	Florizarre	1
4	Salamèche	4
5	Reptincel	4
6	Dracaufeu	4
7	Carapuce	7
8	Carabaffe	7
9	Tortank	7
10	Chenipan	10
11	Chrysacier	10
12	Papilusion	10

```
UPDATE pokemons
  SET id_pokemon_candiesType = IFNULL((
    SELECT LEAST(MIN(pe.id_from), MIN(pe2.id_from))
      FROM pokemon_evolutions pe
        JOIN pokemon_evolutions pe2 ON pe.id_from = pe2.id_to OR pe.id_to =
pe2.id_from
      WHERE pe.id_from = pokemons.id OR pe.id_to = pokemons.id
  ), (
    SELECT MIN(pe.id_from)
      FROM pokemon_evolutions pe
        -- JOIN pokemon_evolutions pe2 ON pe.id_from = pe2.id_to OR pe.id_to =
pe2.id_from
      WHERE pe.id_from = pokemons.id OR pe.id_to = pokemons.id
  ))
```

```
INSERT INTO perso_pokebonbons (id_dresseur, id_pokemon_candiesType,
numberOfCandies)
  SELECT pp.id_dresseur, p.id_pokemon_candiesType, 3
    FROM perso_pokemons pp
      LEFT JOIN pokemons p ON pp.id_pokemon = p.id
  ON DUPLICATE KEY UPDATE numberOfCandies = numberOfCandies + 3
```

```
SELECT pb.id_dresseur, pb.id_pokemon_candiesType, pb.numberOfCandies,
CONCAT('pokébonbons ', p.name) as texte
  FROM perso_pokebonbons pb
    LEFT JOIN pokemons p ON p.id = pb.id_pokemon_candiesType
    RIGHT JOIN pokemons p2 ON p2.id = pb.id_pokemon_candiesType
  WHERE pb.id_dresseur = '29069011' AND p2.id = '24'
```