Pour approfondir, vous pouvez allez voir le site de P. Rigaux [http://sql.bdpedia.fr/].

Considérons le journal d'une MJC qui souhaite faire une étude sur les jeux informatiques. La MJC souhaite pouvoir sauvegarder les jeux informatiques qu'elle possède, l'identité des personnes ayant joué à au moins à l'un de ces jeux et quels étaient leur avis.

Il faut donc un système qui puisse réponde à cette problématique : c'est l'objet d'un système d'informations.

Pour le construire, il va falloir sélectionner les informations qui seront utiles et les stocker sous un certain format : il s'agit de l'étape de ..... Il faut ensuite pouvoir ajouter/modifier/supprimer des données et en extraire de l'information : c'est le propos d'un système de gestion de bases de données (.....).

Schématiquement cela donnerait :

..... (le problème abordé)
modélisation (modèle relationnel)
modèle physique (implémentation d'un .....)

I Modèle relationnel

Afin de modéliser un problème, on va chercher à représenter les objets et leurs relations par des n-uplet de valeurs. On obtiendra alors le modèle relationnel, défini par l'informaticien E. F. Codd en 1 970 alors qu'il travaillait chez IBM.

1 Associer des entités entre elles

Définition 1

Une classe d'entité est un ensemble ..... qui possèdent tous les mêmes attributs (dont on précisera le type). Une instance d'entité sera l'un de ces .....

Définition 2

L'identifiant (ou la clef) d'une entité est un ensemble d'attributs qui permet de déterminer de manière ..... une instance de la classe.

Dans notre exemple, les personnes peuvent être représentées par la classe **personne** :
Leur identifiant est l'attribut .....

personne
idpersonne (....)
.....
.....
genre (char)

.

Les jeux peuvent être représentés par la classe **jeu** :
Leur identifiant est l'attribut .....

jeu
idjeu (....)
titre (....)
sortie (int)

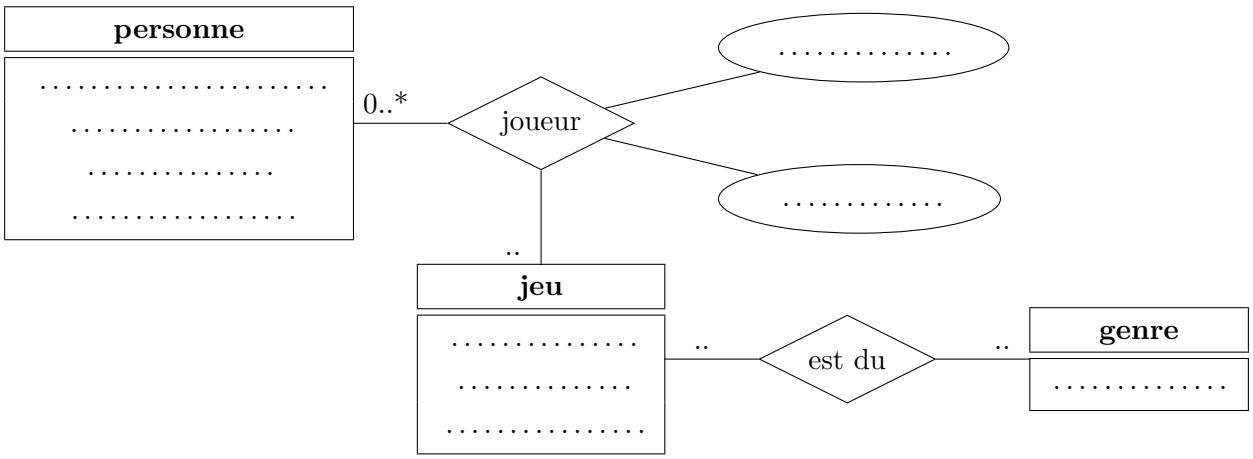
.

En plus de cela, un jeu peut-être apparenté à un ou plusieurs type de genres. Plutôt que repéter la table jeu autant de fois qu'il a de type de genres, on va associer un jeu avec des **genres** :
Leur identifiant est l'attribut .....

genre
type (str)

.

Enfin, une personne peut jouer à un jeu à une certaine date et donner son avis, on associera donc la personne avec les jeux en spécifiant pour chaque association la date et l'avis de la personne.



Les liens des associations comportent le nombre d'instances d'une entité pour chaque instance de l'autre. Ainsi un jeu a pu être joué par ..... personnes (0..*) mais une personne a forcément jouer à au moins un jeu (.....). Un genre peut être possédé par ..... jeux (0..*) mais un jeu possède ..... un genre (.....).

2 Traduction en modèle relationnel

On traduit alors le schéma obtenu en modèle relationnel où chaque entité reste une table dont les colonnes seront ses attributs.

Tandis que les associations se transforment en table contenant pour colonnes les identifiants des entités mises en relations et les informations liées spécifiquement à l'association. On définit ainsi une classe pour chaque association, *e.g.* :

association
id1 (int)
id2 (int)
info (str)

.

Définition 3

Une relation est l'ensemble des valeurs prises par les entités d'une classe (objet ou association).
Le schéma d'une relation est alors la .....
Une base de données est alors ..... des relations.

Dans notre exemple, nous pouvons définir 4 schémas (en regroupant les classes "est du" et "genre") :

personne
idpersonne (int)
prenom (str)
birth (int)
genre (char)

joueur
idpersonne (int)
idjeu (int)
date (int)
avis (str)

jeu
idjeu (int)
titre (str)
sortie (int)

genre
type (str)
idjeu (int)

La relation joueur peut être identifiée par le couple (*idpersonne*, *idjeu*), on définira ce couple comme ..... La relation "est du genre" peut être identifiée par le couple (*type*, *idjeu*), on définira ce couple comme .....

Schémas que l'on peut également noter (on soulignera la clef primaire) :

- ◇ **personne**(idpersonne : `int`, prenom : `str`, birth : `int`, genre : `char`)
- ◇ **joueur**(idpersonne : `int`, idjeu : `int`, date : `int`, avis : `str`)
- ◇ **jeu**(idjeu : `int`, titre : `str`, sortie : `int`)
- ◇ **genre**(type : `str`, idjeu : `int`)

3 Les contraintes d'intégrité

Afin d'assurer une certaine cohérence (= intégrité) dans l'implémentation, on va contraindre les valeurs prises par les entités.

Définition 4

Une transaction est une suite d'opérations qui part d'un état intègre et se termine à un autre état intègre.

Définition 5

Une contrainte d'intégrité est une propriété logique conservée (un invariant) par la base de données avant et après une transaction.

Par ordre alphabétique,

i **Contrainte de domaine : *restreint les valeurs possibles prises par les attributs***

Dans notre exemple, le schéma personne possède quatre attributs dont les types sont définis et restreignent donc leur valeur.

◇ *idpersonne* ne pourra être qu'un .....

◇ *prenom* ne pourra être qu'une .....

◇ *birth* ne pourra être qu'un .....

◇ *genre* ne pourra être qu'un .....

On voit que cela n'empêche pas *birth* de prendre une valeur incohérente comme 2 222. Pour traiter cela, on utilisera une contrainte d'utilisateur.

ii **Contrainte d'entité : *toute relation est unique et possède une clef primaire***

Dans notre exemple, deux relations identiques codent la même information (*e.g.* jeu(1, Starcraft, 1998) et jeu(1, Starcraft, 1998) sont la même instance de jeu).

De plus, chaque schéma possède une clef .....

iii **Contrainte de référence : *garantit la pertinence d'une référence entre deux entités***

Définition 6

Une clef étrangère est un ensemble d'attributs d'une relation qui sont une ..... dans une autre relation.

Dans notre exemple, le schéma **joueur** possède deux attributs (*idpersonne, idjeu*) qui font référence à d'autres schémas (**personne** et **jeu**). Chacun de ces deux attributs forment une .....

Lorsqu'une instance de la classe **joueur** est définie, la valeur de son attribut *idpersonne* doit référencer une personne existante, mais également la valeur de son attribut *idjeu* qui doit aussi référencer un jeu existant.

La contrainte de référence assure que les valeurs données à une clef ..... soient bien la valeur d'une clef ..... existante.

Ainsi si seuls les jeux n° 1, n° 2, n° 3 ont été définis, on ne pourra pas définir une instance *joueur* faisant référence au jeu n° 4.

iv **Contrainte d'utilisateur (ou métier) : *restreint les valeurs prises par les attributs afin qu'elles aient un sens***

Dans notre exemple, les attributs du schéma personne peuvent être resteints comme suit :

◇ *idpersonne* ne pourra être qu'un entier .....,

◇ *prenom* ne pourra être qu'une chaîne de caractères de moins de ... caractères,

◇ *birth* ne pourra être qu'un entier compris supérieur à .....,

◇ *genre* ne pourra être qu'un seul caractère parmi ... et ...

II Implémentation via mySQL

Afin de préserver l'intégrité (= la cohérence) de notre système, on supposera que l'on effectue une suite d'opérations qui la préserve (*i.e.* on effectue une transaction).

Une transaction débute par `START TRANSACTION` et se termine par `COMMIT`. Si l'on souhaite annuler une transaction en cours (*e.g.* à cause d'une erreur), on pourra utiliser la commande `ROLLBACK`.

Par défaut toute opération est atomique et supposée préserver l'intégrité de la bdd, elle forme donc à elle-seule une ..... Néanmoins lorsque plusieurs opérations sont nécessaires avant de retrouver

une bdd intègre, il peut être utile de définir toutes ces opérations au sein d'une transaction, *i.e.* entre les tokken `START TRANSACTION` et `COMMIT`.

Implémentons la bdd de notre exemple.

1 Création de table et définition des contraintes

On peut définir une base de données par

```
DROP DATABASE gamers ;    /* pour ......... la base de données '......' */
CREATE DATABASE gamers ; /* pour ...... la bdd */
USE gamers ;              /* pour ......... la bdd */
```

Voici les types les plus courants en mySQL :

type	signification	type	signification
TINYINT	entier défini sur 8 bits	TIME	heure au format HH:MM:SS (3 octets)
INT	entier défini sur 32 bits	YEAR	entier compris entre 1901 et 2155 (1 octet)
BIGINT	entier défini sur 64 bits	DATE	date au format AAAA-MM-JJ (3 octets)
FLOAT	floattant défini sur 32 bits	DATETIME	date+heure aux mêmes formats (8 octets)
DOUBLE	floattant défini sur 64 bits	BOOL	similaire à CHAR(1)
CHAR(n)	chaîne de caractères de n caractères (si besoin compléter par des espaces) (n octets)		
VARCHAR(n)	chaîne de caractères d'au plus n caractères ($\leqslant n$ octets)		
TEXT	chaîne de caractères de longueur variable		

```
CREATE TABLE IF NOT EXISTS personne ( -- si la table ne pré-existe pas, elle est créée
   idpersonne INT PRIMARY KEY,  -- idpersonne est un entier et la clef .........
   prenom VARCHAR(20) NOT NULL, -- prenom est défini et contient au plus ... caractères
   birth YEAR NOT NULL, -- brith est défini et est un entier
   genre CHAR -- genre peut ne pas être défini et contiendra ... caractère
   ) ;
```

Rq : on peut définir une clef qui s'auto-incrémente à partir de 1 via `INT PRIMARY KEY AUTO_INCREMENT`.

On défini alors les autres tables :

```
CREATE TABLE ..................... (
   idjeu INT ............ ,
   ...... VARCHAR(25) NOT NULL,
   sortie ......  NOT NULL
   ) ;
CREATE TABLE ......................... (
   ...... ............(12) NOT NULL,
   ......... INT NOT NULL
   ) ;
CREATE ...... IF NOT EXISTS joueur (
   ......... INT NOT NULL,
   ...... INT NOT NULL,
   annee ...... NOT NULL,
   avis VARCHAR(...) -- parmi : top, bien, moyen, bof, nul
   ) ;
```

Il faut maintenant définir les clefs primaires qui n'ont pas encore définies, les clefs étrangères et les contraintes d'intégrités.

```
-- format : ALTER TABLE nomTable ADD CONSTRAINT nomRegle CHECK ( test ) ;
ALTER TABLE ......... ADD CONSTRAINT personne_... CHECK ( birth >= ...... ) ;
ALTER TABLE ......... ADD CONSTRAINT personne_CK2 ...... ( genre IN (..., ...) ) ;

```

```
ALTER ..... jeu ADD CONSTRAINT jeu_CK1 CHECK ( ..................... ) ;

-- format : ALTER TABLE nomTable ADD CONSTRAINT nomRegle FOREIGN KEY (attributs)
↪  REFERENCES table(attributs) ;
ALTER TABLE genre ADD CONSTRAINT ......_FK1 FOREIGN ... (......) REFERENCES ...(idjeu);
-- format : ALTER TABLE nomTable ADD CONSTRAINT nomRegle PRIMARY KEY ( attributs ) ;
ALTER ..... genre ADD ......... genre_PK ..... KEY (type, idjeu) ;

/* contraindre la table joueur */
...

...

...

...

...
.
```

Rq : on peut annuler une contrainte en 'la laissant tomber' via la commande :

```
ALTER TABLE nomTable DROP CONSTRAINT nomRegle ;
```

Une fois que les tables ne sont plus contraintes les unes aux autres, on peut les effacer via la commande :

```
DROP TABLE nomTable ;
```

2 Ajout/modification/suppression de données

Voici quelques exemples sur la table **personne** :

◇ **ajout :**

```
INSERT INTO personne (idpersonne, prenom, birth) VALUES (1, 'Abuelo', 1929);
```

renvoit l'erreur `Erreur dans la requête (4025): CONSTRAINT ‘personne_CK1‘ failed for ‘gamers‘.‘personne‘` puisque la contrainte `personne_CK1` n'est pas vérifiée.

Mais la commande suivante fonctionne et ajoute bien la personne :

```
INSERT INTO personne (idpersonne, prenom, birth) VALUES (1, 'Abuelo', 1939);
```

◇ **modification :** Si on souhaite faire une mise à jour, on peut utiliser la commande `UPDATE`

```
UPDATE personne SET idpersonne = 1, prenom = 'Abuelo', birth = 1936, genre = 'M'
   WHERE idpersonne = 1 ;
```

◇ **suppression :** Enfin on peut supprimer une ligne d'une table en faisant référence à la clef primaire

```
DELETE FROM personne WHERE idpersonne = 30 ;
```

III Opérations via mySQL

Nous allons voir sur notre exemple, une petite partie des opérations possibles par mySQL sur une bdd.

1 Sélection

Pour sélectionner le contenu de différentes tables, on utilise le tokken `SELECT`.

```
SELECT * FROM personne ; -- sélectionne toutes les colonnes de toutes les entités de la
↪  table 'personne'
SELECT DISTINCT prenom FROM personne ; -- sélectionne tous les prenoms sans répétition
SELECT prenom, birth FROM personne ; -- sélectionne les prenoms et les dates de
↪  naissances
```

On peut également restreindre la sélection à l'aide du tokken `WHERE`.

```
SELECT * FROM personne  -- sélectionne toutes les colonnes
   WHERE birth = 2001 ; -- des entités de la table 'personne' nées en 2001
```

Nous avions vu que dans le shell Linux le caractère '*' (resp. '?') correspondant à n'importe quels caractères (resp. un unique caractère). Ce caractère est remplacé par '%' (resp. '_') en SQL. ainsi pour obtenir toutes les personnes dont le prénom commence par "B" on écrira :

```
SELECT * FROM personne
   WHERE prenom LIKE "B%" ;
```

2 Fonctions d'agrégat

fonction	résultat
COUNT	compte le nombre d'occurences trouvées
SUM	fait la ..... des valeurs trouvées
MIN	renvoye le ..... des valeurs trouvées
MAX	renvoye le ..... trouvées
AVG	fait la .....

On peut calculer la moyenne des années de naissances de toutes les personnes de la table **personne** avec la commande :

```
SELECT AVG( birth ) FROM personne ;
```

On peut également calculer l'année de naissance la plus grande des personnes dont le prénom se termine par 'e' :

```
SELECT MAX( birth ) FROM personne
   WHERE prenom LIKE "%e" ;
```

3 Tris

Il est possible de trier des données à l'aide de la clausse `ORDER BY`

Pour ordonner selon l'année la plus récente (grande) à la plus ancienne (petite) (ordre décroissant = `DESC`) :

```
SELECT * FROM personne
   ORDER BY birth DESC ;
```

On peut trier selon l'année la plus récente à la plus ancienne et par ordre alphabétique des prénoms :

```
SELECT * FROM personne
   ORDER BY birth DESC, prenom ASC ;
```

Rq : on verra en exercice comment obtenir les 3 premiers avec la clause `LIMIT`.

4 Jointures

On peut également faire des intersections (`INNER JOIN`), des unions (`FULL OUTER JOIN`), ou d'autres, à l'aide de jointures entre différentes tables.

Par exemple, pour déterminer les personnes ayant jouées à un jeu en 2020, on cherchera à faire l'intersection entre la table **personne** et la table **joueur** pour lesquelles on retrouve le même *idpersonne* et où l'année vaut ..... On écrit donc :

```
SELECT * FROM personne AS p
   INNER JOIN joueur AS j ON p.idpersonne = .........
   WHERE j.annee = ...... ;
```

Pour savoir quels sont les prénoms des personnes ayant donné un avis 'top' ou 'bien' et en quelles années, on peut faire l'intersection entre la table **personne** et la table **joueur** pour lesquelles on retrouve le même *idpersonne* et où l'avis est dans la liste ('top', 'bien').

```
SELECT p.prenom, j.annee FROM personne AS p
   INNER JOIN joueur AS j ON ...............
   WHERE j.avis IN ('...', '...') ;
```

Rq : plusieurs jointures peuvent se suivre (*cf.* exos)

5 Sous requêtes

Enfin on peut créer des sous-requêtes que l'on utilisera pour la requête suivante :
Si on souhaite les prénoms des personnes les plus jeunes, on pourrait vouloir simplement écrire :

```
SELECT MAX(birth), prenom FROM personne ;
```

⚠ mais cela détermine la date plus grande et l'associe au premier prénom. On découpe donc le problème,

◇ déterminer la date de naissance la plus récente

◇ sélectionner tous les prénoms dont l'année de naissance est égal à l'année la plus récente

```
SELECT birth, prenom FROM personne
   WHERE birth = ( SELECT ............ FROM ......... ) ;
```

Rq : les requêtes avec les tokken `GROUP BY` et `HAVING` sont hors-programme.

IV Interface web

Nous verrons en projet, comment on peut lier une bdd à une page web dynamique au travers de PHP.