*IFT-6800, Automne 2016*

# Cours #5– Design de bases de données, les diagrammes E/R

Louis Salvail
André-Aisenstadt, #3369
salvail@iro.umontreal.ca

# Relations entre entités

- Nous allons voir comment faire le design d'une base de données en utilisant le modèle d'*entité-relation*.
- Cette façon de voir les choses permet d'obtenir un design de haut niveau d'une base de données d'une façon qui explicite la structure de l'information que nous voulons représenter.
- Nous dénotons ce modèle par le modèle E/R.

# Le modèle E/R

- Le modèle E/R est une représentation graphique du schéma d'une base de données.
- Souvent, les designers représentent en premier lieu ce qu'ils veulent dans le modèle E/R.
- Par la suite, ils traduisent le schéma dans le modèle relationnel (une collection de tables).
- Il n'y a pas de DBMS qui accepte la représentation E/R directement.
- La traduction est cependant essentiellement mécanique.

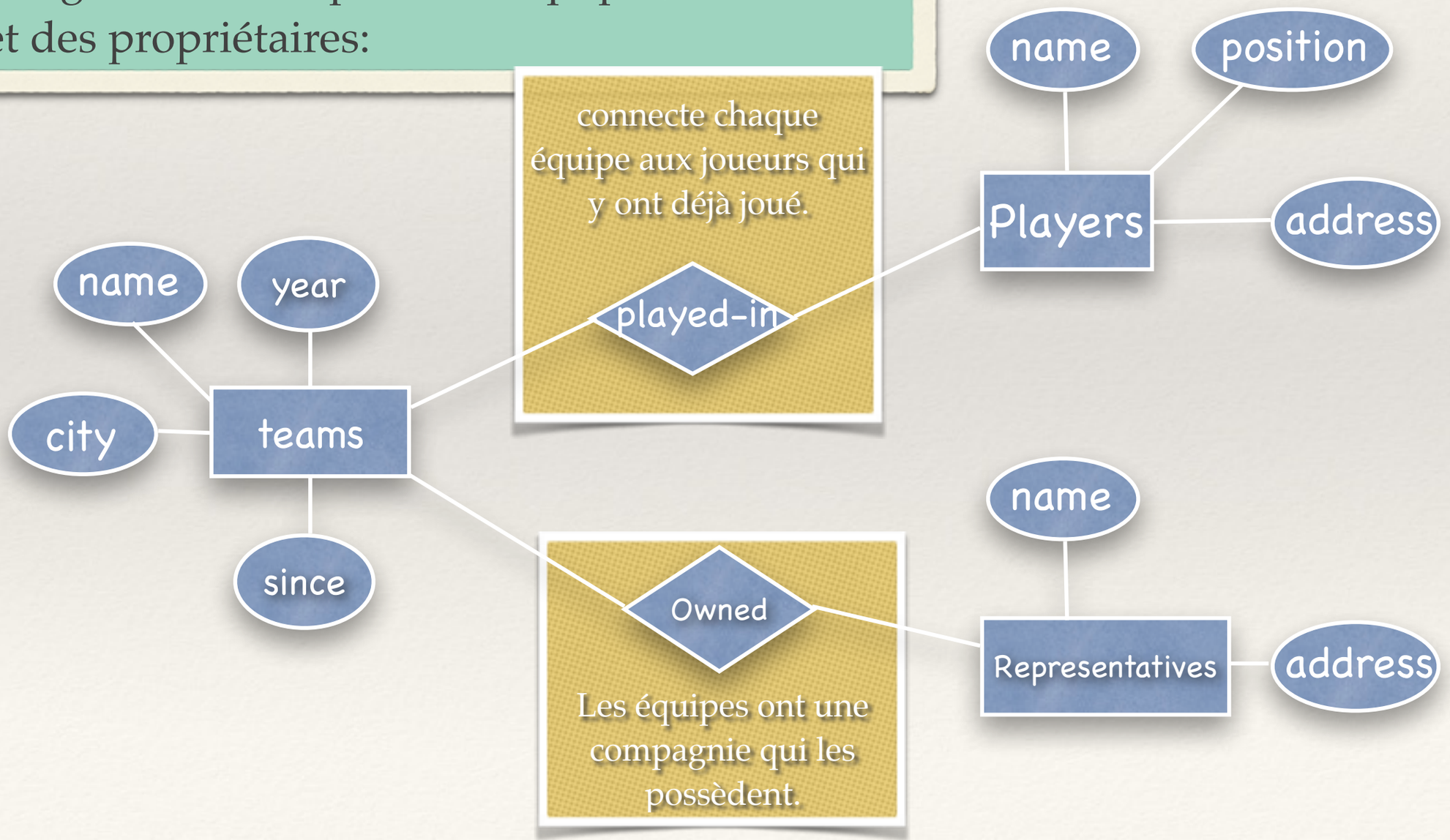
# Les éléments du modèle E/R

- **Ensembles d'entités:** Une entité est un objet abstrait d'un certain type. Elle regroupe un ensemble d'éléments ayant une forme commune. Exemple: les *conducteurs* au Québec.
- **Attributs:** Propriétés d'ensembles d'entités dans notre modèle. Exemple: les *conducteurs* ont un attribut *nom*.
- **Relations:** Des connexions entres deux entités ou plus. Exemple: les *conducteurs* sont en relation avec le *type de véhicule* qu'ils peuvent conduire.

# Diagrammes E/R

Légende: Ensemble d'entités — Relations — Attributs

Voici un diagramme E/R pour des équipes, des joueurs et des propriétaires:

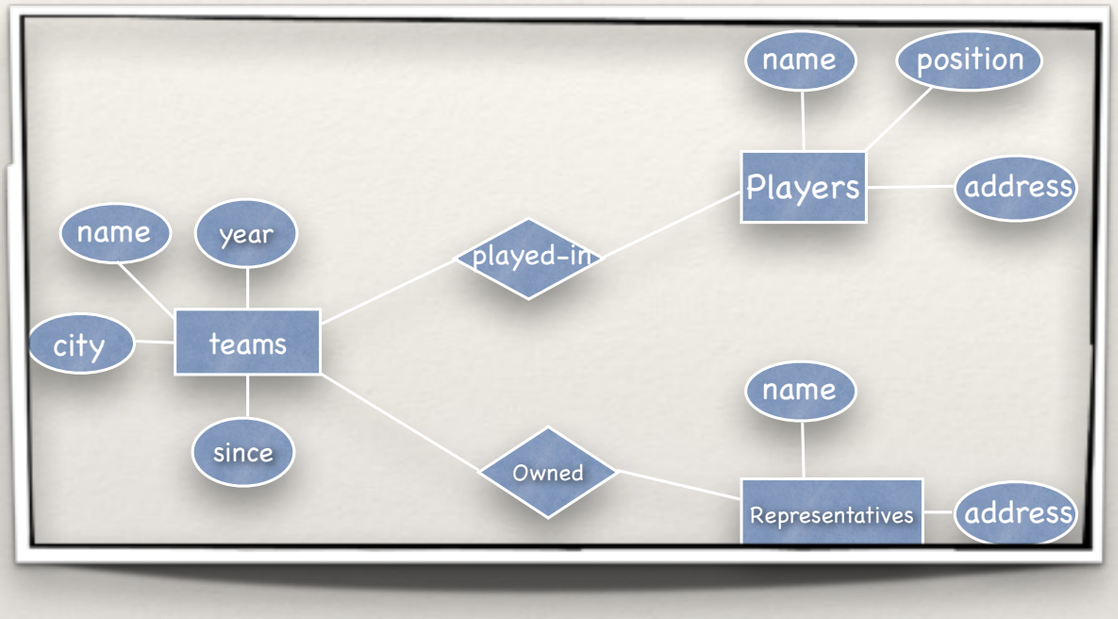


# Instances d'un diagramme

- Le diagramme E/R est une représentation du schéma d'une base de données.
- Une base de données en tant que telle, contient des données particulières nommées *instances de base de données*. Chaque ensemble d'entités contient des entités avec des valeurs pour chaque attribut.
- Une instance contient également des choix spécifiques pour les relations entre les entités.

# Instances d'ensembles d'entités

Pour les entités:




Teams

| city | name | year | since |
|---|---|---|---|
| Montréal | Expos | 1983 | 1967 |
| Montréal | Carabins | 1985 | 1922 |
| Québec | Rouge et Or | 1996 | 1996 |
| Victoriaville | Tigres | 1990 | 1982 |
| Montréal | Expos | 1984 | 1967 |

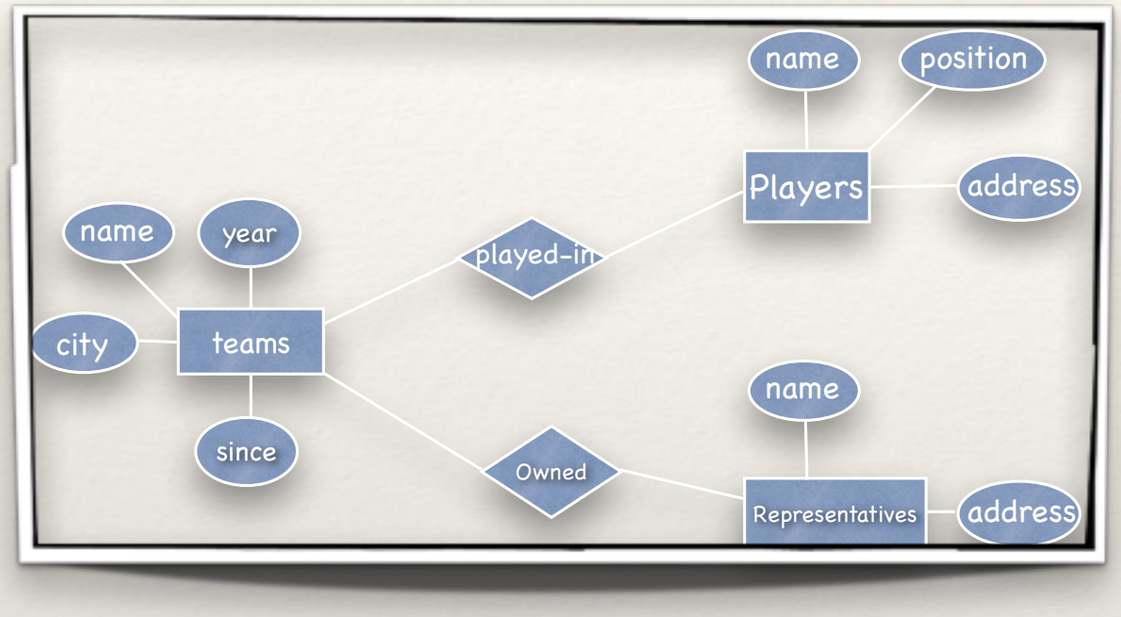
Representatives

| name | address |
|---|---|
| Fried Hot Dog enr. | 2 rue Xoyz |
| Plastic Chairs inc. | 134 rue Wilopo |
| Molson | 3645 rue Un |
| Saputo | 98 rue Barrée |

Players

| name | position | address |
|---|---|---|
| Fidel Castro | lançeur | 2 rue Xoyz |
| Roger Rogers | gardien de but | 134 rue Wilopo |
| Pol Pot | champs gauche | 3645 rue Un |
| Batman | ailier droit | 98 rue Barrée |

# Instances d'une relation




Owned

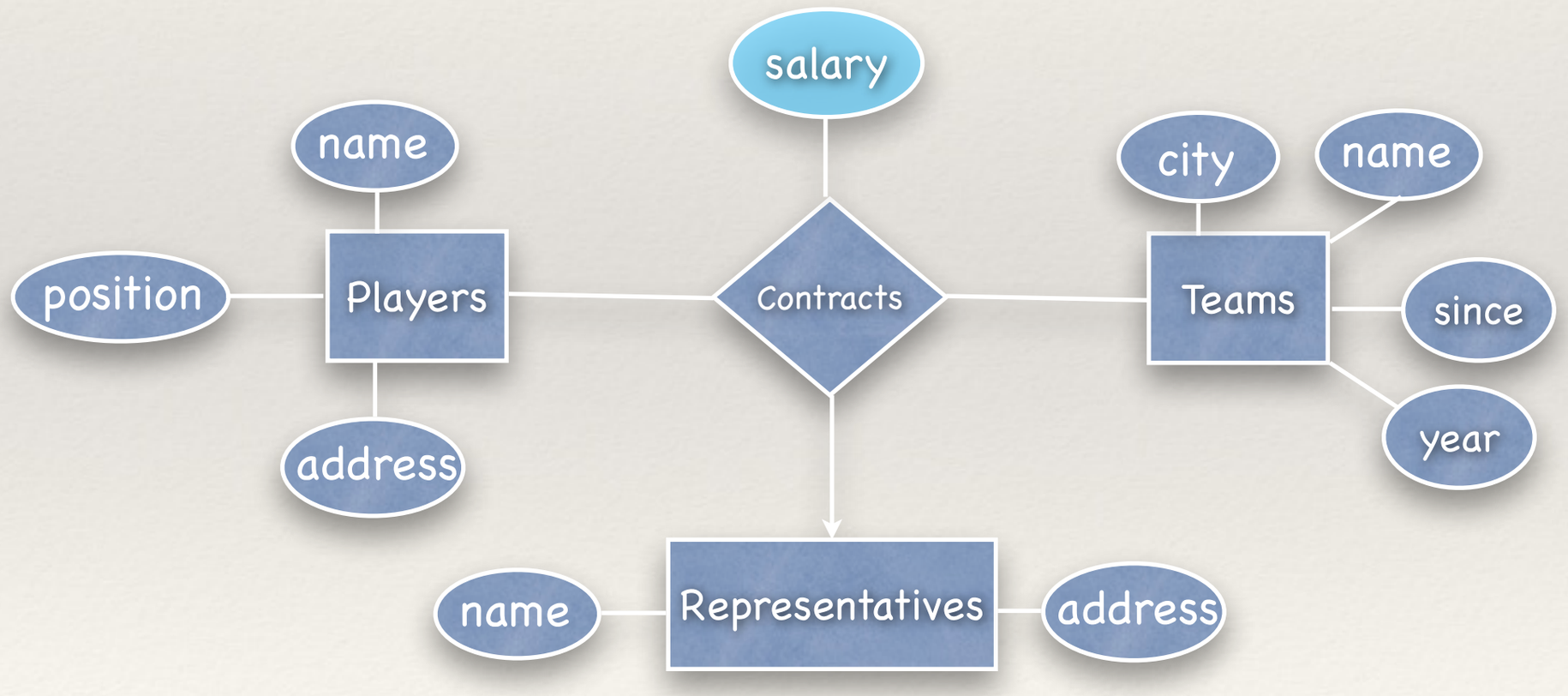
| Repres. name | team name |
|---|---|
| Plastic Chairs inc. | Expos |
| Molson | Carabins |
| Saputo | Rouge et Or |
| Fried Hot Dog enr. | Tigres |

played-in

| player name | team name |
|---|---|
| Fidel Castro | Expos |
| Roger Rogers | Carabins |
| Pol Pot | Rouge et Or |
| Roger Rogers | Expos |

# Attributs sur une relation

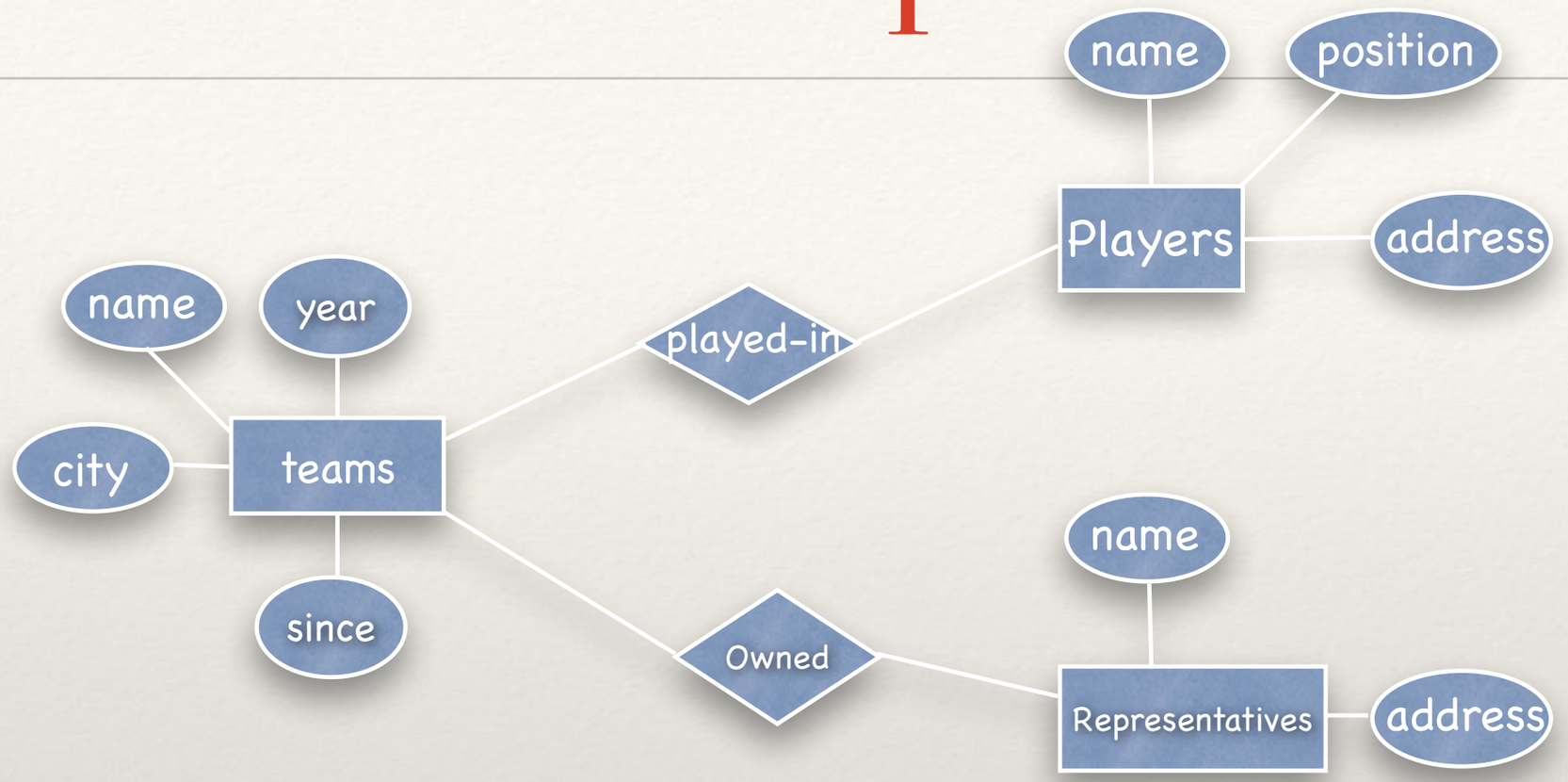
- Il est parfois utile de placer des attributs sur des relations:



# Principes d'un bon design

- **Fidelité de la représentation:** Les ensembles d'entités et les attributs doivent représenter la réalité. Les relations doivent modéliser des faits réels du monde que l'on veut représenter.
- **Pas de redondance:** Les choses doivent être dites qu'une seule fois. Ceci permet de minimiser l'espace utilisée pour ranger les relations et de faciliter les mises à jour des données.
- **Simplicité:** Éviter d'ajouter plus d'éléments que nécessaire.
- **Choisir les bonnes relations:** Celles qui sont utiles et qui minimisent les redondances. Pas de relation qui peuvent être dérivées des ensembles d'entités et des relations qui existent déjà.

# Exemple:




- Devrait-on mettre les noms et adresses des représentants dans la description des équipes?
  - Si oui alors nous introduisons de la redondance. Les informations (adresses et noms) sur le représentant sont copiés à chaque année pour laquelle  le représentant est propriétaire de l'équipe.
    - S'il n'y avait que le nom du représentant alors on pourrait le faire, car l'information ne serait plus redondante, elle est nécessaire pour spécifier un représentant.
  - Que se passe-t-il si un représentant n'a pas d'équipe pour le moment?

# Contraintes

- **Clés:** Un ensemble d'attributs (un ou plus) qui identifie chaque entité dans un ensemble d'entités de façon unique. Il ne peut y avoir deux entités de l'ensemble qui coïncident entièrement pour les attributs de la clé.
- **Valeur unique:** Une contrainte qui indique que la valeur d'un attribut est unique. Une clé est à valeur unique. Un numéro d'assurance social également pour un ensemble d'entités ''Personnes''.
- **Intégrité référentielle:** Une contrainte qui dit qu'un certain objet existe dans la base de données.
- **Contraintes de domaine:** La valeur d'un certain attribut est dans un ensemble de valeurs: {oui,non}, {1..18}, {patate, orange, carotte}, etc…
- **Contraintes générales:** D'autres types de contraintes comme '*aucune équipe n'a plus de 65 joueurs*'.

# L'avantage des contraintes

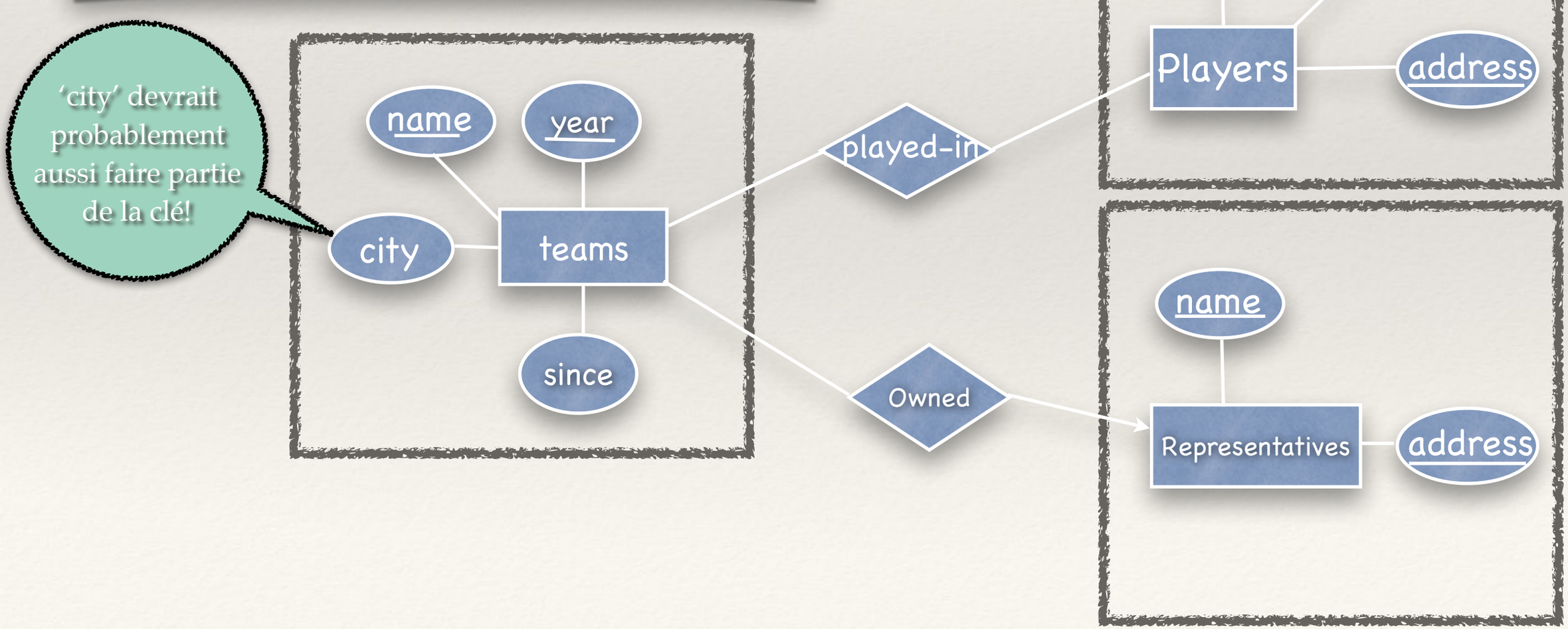
- Les contraintes que nous avons vues permettent:
  - d'identifier une entité de façon unique sans confusion possible (clés),
  - de sauver de l'espace et du temps (valeur unique),
  - un accès plus rapide aux données (intégrité référentielle), et
  - d'assurer la consistance (contraintes de domaine et générales).

# Contrainte #1: Clés

- **Une clé** pour un ensemble d'entités est un ensemble d'attributs qui définissent chaque élément de l'ensemble de façon unique. Deux entités de l'ensemble ne peuvent pas partager tous les attributs de la clé.
    - Chaque ensemble d'entités doit avoir une clé,
    - Une clé peut contenir plusieurs attributs,
    - Il peut y avoir plus d'une clé pour un même ensemble d'attributs.
    - On souligne les attributs qui forme la clé d'un ensemble d'entités. Il est rare qu'il soit utile de spécifier plus d'une clé pour un ensemble d'entités. Si nous avions à le faire alors on pourrait souligner différemment pour chaque clé.

# Notre exemple avec ses clés

Dans certains cas, un ensemble d'entités doit faire intervenir les attributs des ensembles d'entités auxquels il est connecté par des relations pour former une clé. Nous n'utiliserons pas cette possibilité, mais je vais la mentionner plus tard (pour votre culture).



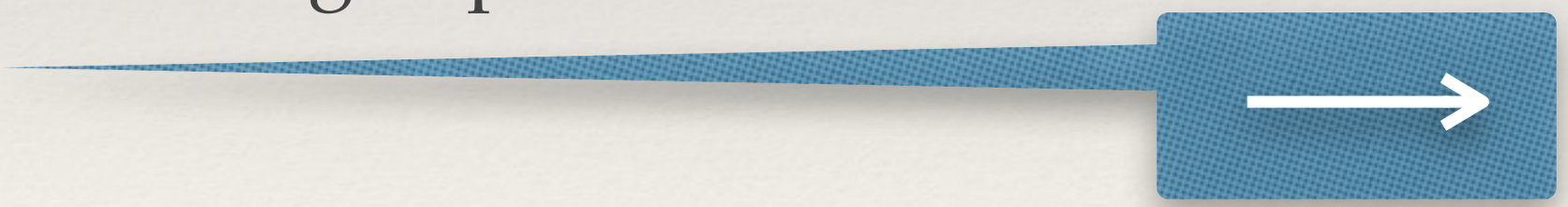
# Contrainte #2: Valeur unique

- Une relation qui connecte un ensemble d'entités E et un ensemble d'entités F peut avoir la propriété que pour toutes les instances dans E il existe au plus une instance dans F qui satisfait la relation.
- Nous indiquons les valeurs uniques en ajoutant une *tête de flèche pleine* à la ligne qui part de la relation entre E et F et entre dans F.

- Exemple: Un représentant ne possède pas plus d'une équipe (auquel cas il est propriétaire), possiblement aucune.

# Contrainte #3: Intégrité référentielle

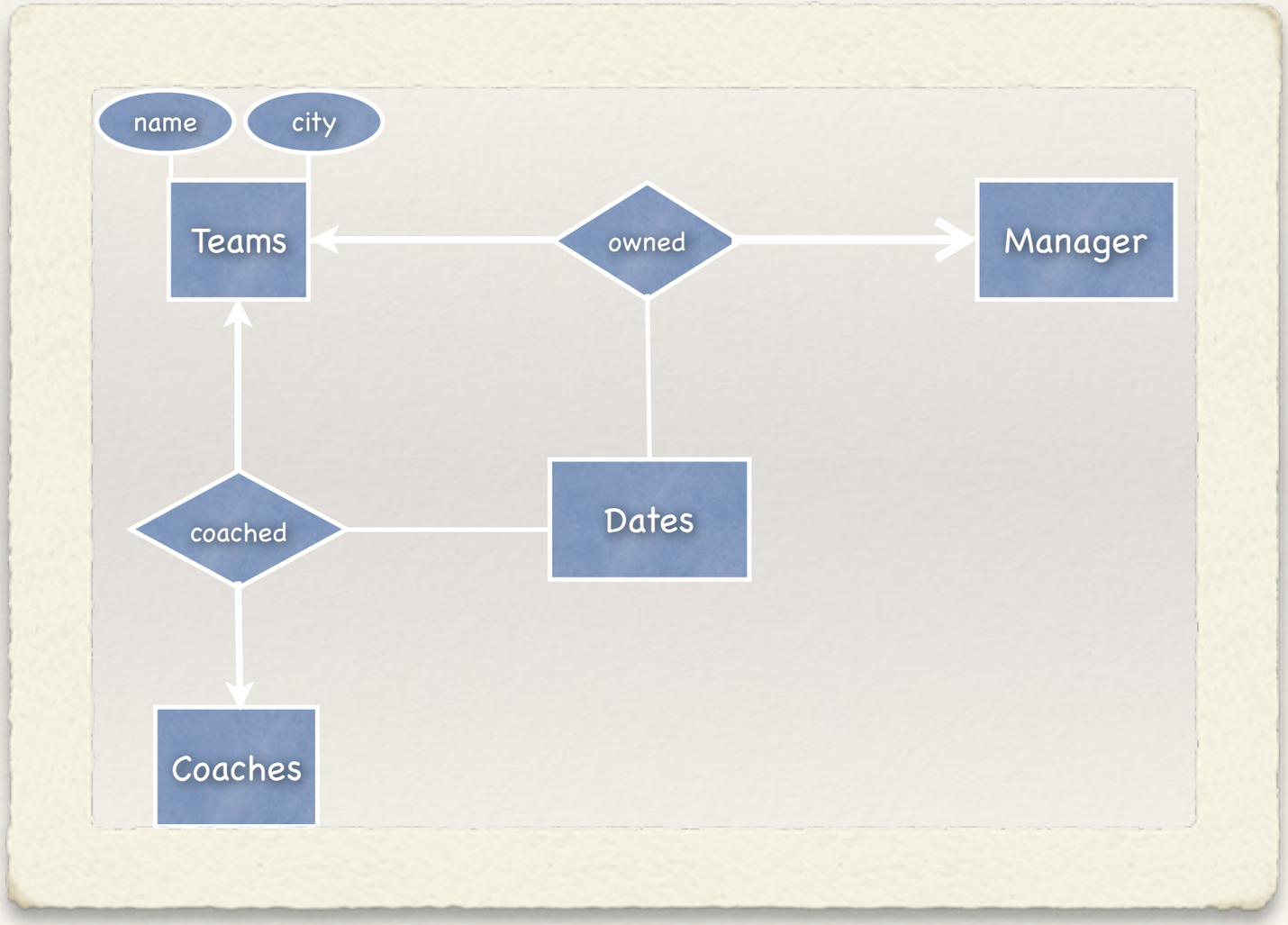
- Une relation qui connecte un ensemble d'entités E et un ensemble d'entités F peut avoir la propriété que pour toutes les instances dans E il existe toujours une et une seule instance dans F qui satisfait la relation.
- Nous indiquons l'intégrité référentielle en ajoutant une *tête de flèche vide* à la ligne part de la relation entre E et F et entre dans F.



- Exemple: Une équipe a toujours un et un seul représentant/ propriétaire.

# Valeur unique & intégrité référentielle dans un diagramme E/R

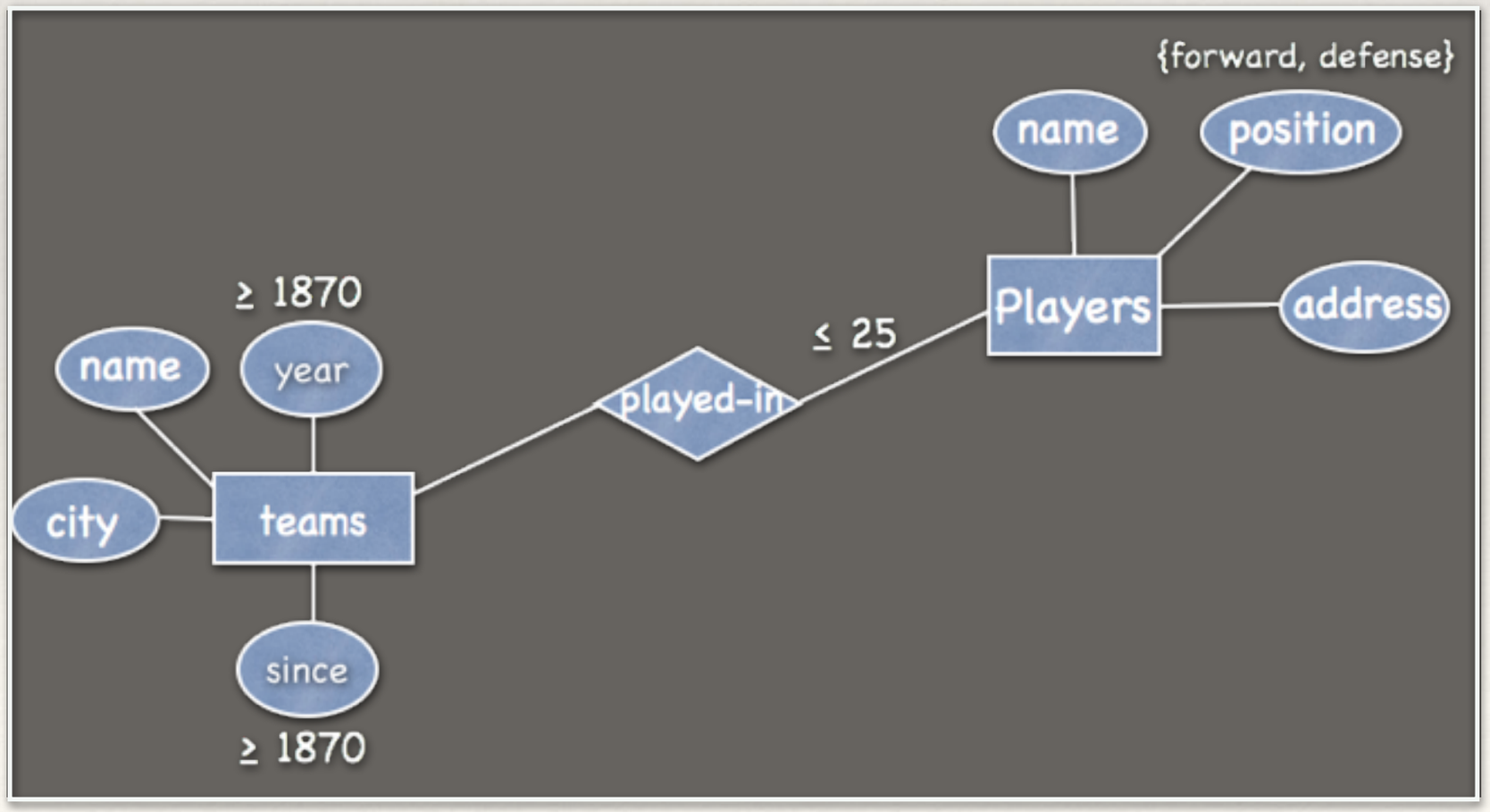
À chaque moment, chaque équipe est dirigée par un manager et au plus un manager dirige chaque équipe.




Même chose pour l'entraîneur sauf qu'à un moment donné, une équipe pourrait ne pas avoir d'entraîneur

# Contraintes #4&5: De domaine et générales

- Pour votre culture générale…
- On indique les contraintes de domaines de la façon indiquée plus bas.
- On peut aussi indiquer des contraintes générales comme *'pas plus de 25 joueurs par équipes'*.



# Les ensembles faibles d'entités

(culture générale)

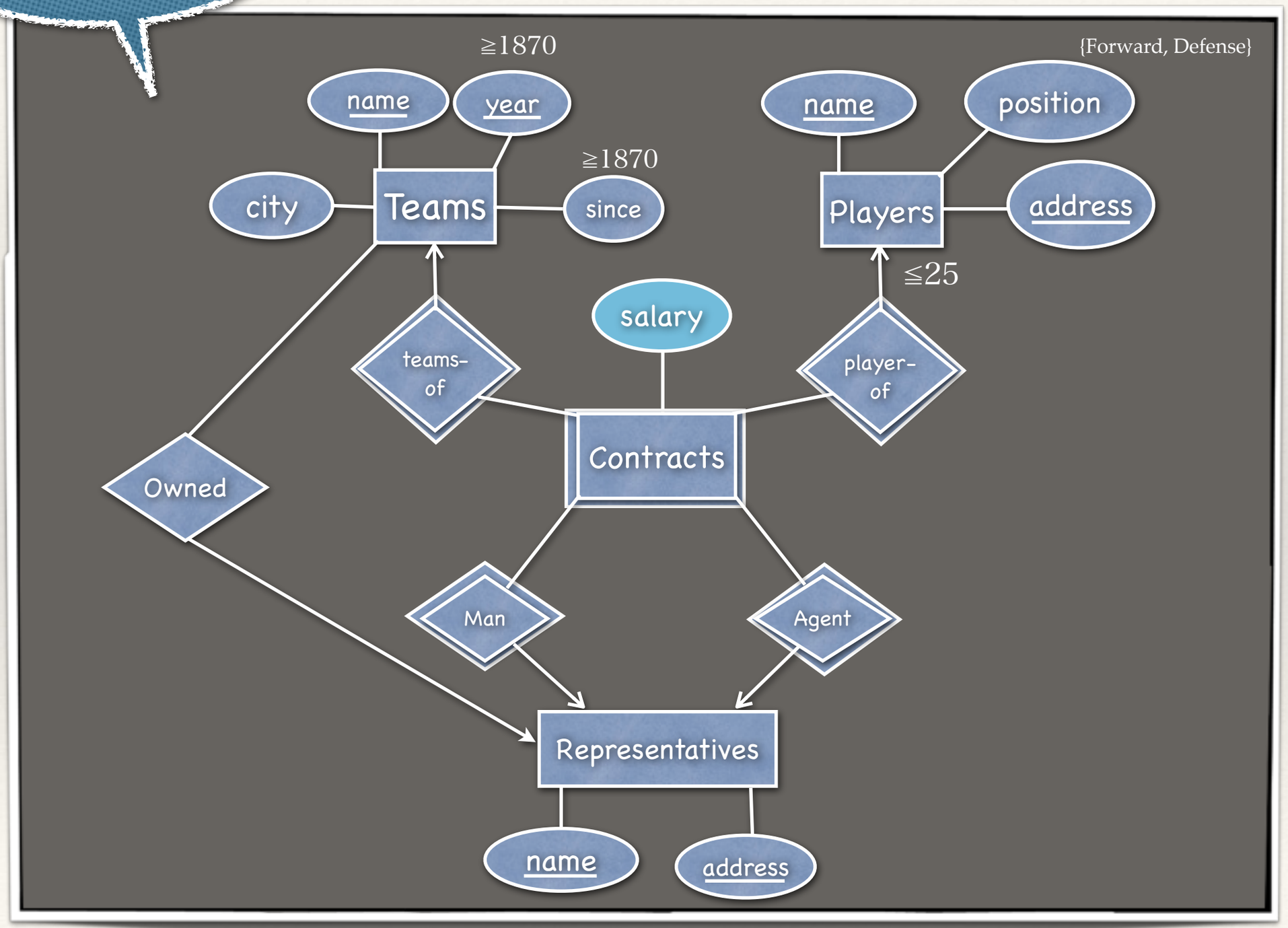
- Dans certain cas, un ensemble d'entités n'a pas d'attributs qui peuvent servir de clé. On dit de cet ensemble d'entités  qu'il est faible. Une clé peut être obtenu en prenant des attributs de l'ensemble d'entités plus des attributs d'autres ensembles d'entités connectés par une relation (dite *de support*) à l'ensemble de départ.

  - Par exemple, lorsque vous avez des club-écoles pour un grand club, un pour chaque catégorie. Un club-école peut alors être exprimé d'une façon unique en prenant sa catégorie (attribut du club école) et les attributs de clé du grand club.

  - On dénote ceci par des *lignes doubles* sur les ensembles d'entités et les relations de support qu'il faut suivre pour obtenir une clé…

# exemple plus complet

Le plusimportant ici sont les ensembles d'entités, les relations, les attributs et les clés.



# Diagramme E/R en Base de données

- Les DBMS ne comprennent pas les diagrammes E/R pour réaliser un schéma particulier.
- Il faut donc les convertir dans le langage que comprend le DBMS.
- Nous allons voir les DBMS SQL qui sont des systèmes de bases de données relationnelles:
  - Dans ce cas, on remplace le diagramme par un ensemble de tableaux, un peu comme nous l'avons vu précédemment. Chaque tableau contient un ensemble de colonne qui font référence à des attributs.
  - Une base de données qui répond au schéma décrit par un diagramme E/R est obtenu en remplissant les tableaux de rangées.

# Transformation du diagramme E/R en ensemble de tables

- Nous avons vu comment transformer un ensemble d'entités en tableau où chaque attribut correspond à une colonne.
- Nous avons vu grosso modo comment transformer une relation entre deux ensembles d'entités (A et B) en un tableau. D'une façon plus précise, on procède ainsi:

A:

| clé | info |
|---|---|
| a1 | i1 |
| a2 | i2 |
| a3 | i3 |
| a4 | i4 |

B:

| clé | info |
|---|---|
| b1 | i1 |
| b2 | i2 |
| b3 | i3 |
| b4 | i4 |

Relation entre tableaux A et B

| clé pour A | clé pour B |
|---|---|
| … | … |
| a1 | b3 |
| … | … |
| … | … |

Indique que la rangée du tableau A avec clé a1 est en relation avec la rangée du tableau B avec clé b3.

# DBMS SQL

- Un DBMS SQL permet de créer les tables obtenus en traduisant le diagramme E/R.
- Une fois les tables créées, le DBMS nous permet d'ajouter,de retirer et de modifier des rangées dans ces tables.
  - Si des données sont incomplètes alors nous pouvons indiquer NULL pour certains attributs (quand c'est permis).
- Une base de données est ainsi créée.
- Le DBMS nous permet de faire des requêtes sur ces données, nous verrons comment la prochaine fois.

# Exemple: Sean Lahman's Baseball

- Je vais utiliser une base de données SQL qui contient beaucoup de données sur le baseball professionnel depuis ses débuts.
- Je ne suis pas un fan de baseball! Cependant, la base de données permet de faire beaucoup et a un design simple et est gratuite.
- Je vais l'utiliser pour cette raison et son site web est:
    - http://www.seanlahman.com
- Il s'agit d'une base de données relationelle sur les statistiques des individus et des équipes du baseball majeur depuis 1871.
- Elle contient 24 tables.

# Les tables de Lahman

```
+---------------------+
| Tables_in_baseball  |
+---------------------+
| AllstarFull         |
| Appearances         |
| AwardsManagers      |
| AwardsPlayers       |
| AwardsShareManagers |
| AwardsSharePlayers  |
| Batting             |
| BattingPost         |
| CollegePlaying      |
| Fielding            |
| FieldingOF          |
| FieldingPost        |
| HallOfFame          |
| Managers            |
| ManagersHalf        |
| Master              |
| Pitching            |
| PitchingPost        |
| Salaries            |
| Schools             |
| SeriesPost          |
| Teams               |
| TeamsFranchises     |
| TeamsHalf           |
+---------------------+
```

Les joueurs ayant joué pour une année.

Les statistiques des frappeurs chaque année.

Les statistiques des joueurs sur le terrain chaque année.

Les managers (entraîneurs)  depuis le début.

Tous les joueurs depuis le début.

Les lanceurs chaque année.

Les salaires des joueurs chaque année.

Les équipes chaque année.

# La table Master:

attributs

attribut qui peut être NULL

Une clé? Si oui alors quel type? PRI=principal

types d'attributs

Une clé n'est jamais NULL

Valeur par défaut pour attributs

L'attribut playerID est unique pour chaque joueur. Il est composé de 10 caractères.

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| playerID | varchar(10) | NO | PRI | NULL | |
| birthYear | int(11) | YES | | NULL | |
| birthMonth | int(11) | YES | | NULL | |
| birthDay | int(11) | YES | | NULL | |
| birthCountry | varchar(50) | YES | | NULL | |
| birthState | varchar(2) | YES | | NULL | |
| birthCity | varchar(50) | YES | | NULL | |
| deathYear | int(11) | YES | | NULL | |
| deathMonth | int(11) | YES | | NULL | |
| deathDay | int(11) | YES | | NULL | |
| deathCountry | varchar(50) | YES | | NULL | |
| deathState | varchar(2) | YES | | NULL | |
| deathCity | varchar(50) | YES | | NULL | |
| nameFirst | varchar(50) | YES | | NULL | |
| nameLast | varchar(50) | YES | | NULL | |
| nameGiven | varchar(255) | YES | | NULL | |
| weight | int(11) | YES | | NULL | |
| height | double | YES | | NULL | |
| bats | varchar(1) | YES | | NULL | |
| throws | varchar(1) | YES | | NULL | |
| debut | datetime | YES | | NULL | |
| finalGame | datetime | YES | | NULL | |
| retroID | varchar(9) | YES | | NULL | |
| bbrefID | varchar(9) | YES | | NULL | |

# La table Teams:

l'année de l'équipe pour ces stats

G: nombre de parties jouées cette année là

L'identificate pour l'équipe.

La clé principale de Teams est composée de trois attributs: `yearID`, `lgID`, `teamID`

W: nombre de parties gagnées cette année là

L: nombre de parties perdues cette année là

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| yearID | int(11) | NO | PRI | 0 | |
| lgID | varchar(2) | NO | PRI | | |
| teamID | varchar(3) | NO | PRI | | |
| franchID | varchar(3) | YES | | NULL | |
| divID | varchar(1) | YES | | NULL | |
| Rank | int(11) | YES | | NULL | |
| G | int(11) | YES | | NULL | |
| Ghome | int(11) | YES | | NULL | |
| W | int(11) | YES | | NULL | |
| L | int(11) | YES | | NULL | |
| Di[illegible] | varchar(1) | YES | | NULL | |
| [illegible]cWin | varchar(1) | YES | | NULL | |
| LgWin | varchar(1) | YES | | NULL | |
| WSWin | varchar(1) | YES | | NULL | |
| R | int(11) | YES | | NULL | |
| AB | int(11) | YES | | NULL | |
| H | int(11) | YES | | NULL | |
| 2B | int(11) | YES | | NULL | |
| 3B | int(11) | YES | | NULL | |
| HR | int(11) | YES | | NULL | |
| BB | int(11) | YES | | NULL | |
| SO | int(11) | YES | | NULL | |
| SB | int(11) | YES | | NULL | |
| CS | int(11) | YES | | NULL | |
| HBP | int(11) | YES | | NULL | |
| SF | int(11) | YES | | NULL | |
| RA | int(11) | YES | | NULL | |
| ER | int(11) | YES | | NULL | |
| ERA | double | YES | | NULL | |
| CG | int(11) | YES | | NULL | |
| SHO | int(11) | YES | | NULL | |
| SV | int(11) | YES | | NULL | |
| IPouts | int(11) | YES | | NULL | |
| HA | int(11) | YES | | NULL | |
| HRA | int(11) | YES | | NULL | |
| BBA | int(11) | YES | | NULL | |
| SOA | int(11) | YES | | NULL | |
| E | int(11) | YES | | NULL | |
| DP | int(11) | YES | | NULL | |
| FP | double | YES | | NULL | |
| name | varchar(50) | YES | | NULL | |
| park | varchar(255) | YES | | NULL | |
| attendance | int(11) | YES | | NULL | |
| BPF | int(11) | YES | | NULL | |
| PPF | int(11) | YES | | NULL | |

# La table Salaries:

```
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| yearID   | int(11)    | NO   | PRI | 0       |       |
| teamID   | varchar(3) | NO   | PRI |         |       |
| lgID     | varchar(2) | NO   | PRI |         |       |
| playerID | varchar(9) | NO   | PRI |         |       |
| salary   | double     | YES  |     | NULL    |       |
+----------+------------+------+-----+---------+-------+
```

La table Salaries encode une relation entre les joueurs (Master) et les équipes (Teams) qui les embauchent pour une année donnée. La clé pour cette relation est une clé pour Teams (`teamID`+`yearID`+`lgID`), une clé pour Master (`playerID`).

# La table Managers

```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| playerID | varchar(10) | YES  |     | NULL    |       |
| yearID   | int(11)     | NO   | PRI | 0       |       |
| teamID   | varchar(3)  | NO   | PRI |         |       |
| lgID     | varchar(2)  | YES  |     | NULL    |       |
| inseason | int(11)     | NO   | PRI | 0       |       |
| G        | int(11)     | YES  |     | NULL    |       |
| W        | int(11)     | YES  |     | NULL    |       |
| L        | int(11)     | YES  |     | NULL    |       |
| rank     | int(11)     | YES  |     | NULL    |       |
| plyrMgr  | varchar(1)  | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

Notez qu'un manager a aussi un `playerID` défini dans la table Master. Autrement dit, la table Master contient plus que les joueurs, elle contient également les managers.

Pour avoir le nom complet d'un manager, il faudra donc consulter également la table Master.

# La table Appearances

Cette table permet de déterminer où les joueurs ont joué à chaque année:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| yearID | int(11) | NO | PRI | 0 | |
| teamID | varchar(3) | NO | PRI | | |
| lgID | varchar(2) | YES | | NULL | |
| playerID | varchar(9) | NO | PRI | | |
| G_all | int(11) | YES | | NULL | |
| GS | int(11) | YES | | NULL | |
| G_batting | int(11) | YES | | NULL | |
| G_defense | int(11) | YES | | NULL | |
| G_p | int(11) | YES | | NULL | |
| G_c | int(11) | YES | | NULL | |
| G_1b | int(11) | YES | | NULL | |
| G_2b | int(11) | YES | | NULL | |
| G_3b | int(11) | YES | | NULL | |
| G_ss | int(11) | YES | | NULL | |
| G_lf | int(11) | YES | | NULL | |
| G_cf | int(11) | YES | | NULL | |
| G_rf | int(11) | YES | | NULL | |
| G_of | int(11) | YES | | NULL | |
| G_dh | int(11) | YES | | NULL | |
| G_ph | int(11) | YES | | NULL | |
| G_pr | int(11) | YES | | NULL | |

# Les types d'attributs en SQL

- **CHAR**(*t*): Une chaîne de caractères de longueur *t*.
- **VARCHAR**(*max*): Une chaîne de caractères de longueur au plus *max*.
- **TEXT**: Un texte d'au plus 65535 caractères.
- **SET**: Un ensemble de valeur permise. Exemples: (M,F), (bleu, vert, rouge).
- **INT**(*t*): Un entier entre -2147483648 à 2147483647 si signé. Un entier entre 0 et 4294967295 sinon. Le nombre de chiffres maximum peut être indiqué par *t*.
- **double**, **real**: Un nombre réel d'un certain nombre de chiffres décimaux (double en contient deux fois plus que real).

# Les types d'attributs en SQL (II)

- **datetime**: Une date et un temps.
- **date**: Une date seulement.
- **time**: Un temps seulement.
- **year**: Une année en deux chiffres ou quatre.
- **bit**: Un bit qui vaut 0,1 ou NULL.
- **binary**(*n*): Une chaîne de bits de longueur *n*.
- etc,etc,etc…

# Créer un table en SQL

```
CREATE TABLE Salaries(
                    yearID:  int(11),
                    teamID: varchar(3),
                    lgID: varchar(2),
                   playerID: varchar(9),
                    salary: double );
```

Il n'y a pas de clé définie.

D'autres contraintes ne sont pas indiquées

Cette commande mySQL produit une table vide pour `Salaries`.

# Indiquer les contraintes en SQL

```
CREATE TABLE Salaries(
        yearID:    int(11),
        teamID:    varchar(3),
        lgID:      varchar(2),
        playerID: varchar(9),
        salary:    double NOT NULL,
        PRIMARY KEY (yearID, teamID, lgID, playerID)
);
```

# Autres contraintes

- `FOREIGN KEYS`: Un attribut (ou colonne) d'une table est une clé dans une autre table. Définir une colonne de ce type permet d'y accéder plus rapidement.
    - Permet de garder la base de données sous la bonne forme en s'assurant que lorsque qu'une `FOREIGN KEY` est détruite dans sa table alors cet enregistrement l'est aussi dans la table qui définie la `FOREIGN KEY`.
    - Permet de conserver la consistance de la base de données.
- `UNIQUE`: Un attribut (une colonne) `UNIQUE` est comme une clé sans en être une. Chaque enregistrement donne des valeurs uniques à ces attributs.
    - Permet de poser des contraintes de valeur unique mentionnée précédemment.

# Remplir les tables en SQL

- Il y a plusieurs façons de remplir les tables créées pour finalement obtenir une base de données:
  - Insérer une nouvelle ligne:
    - ```
      INSERT Salaries
      VALUES (1977, 'MON', 'NL', 'perezto01', 23500);
      ```
  - LIRE d'un fichier:
    - ```
      LOAD DATA LOCAL INFILE 'path/salaries.txt
       INTO TABLE Salaries;
      ```