Troisième Partie

Schémas

Partie III
Schémas

- Formes normales
  - Définitions
  - 1FN, 2FN, 3FN
  - Autres formes normales
- Entités et associations
  - Définitions
  - Types et instances
  - Degré des associations
  - Cardinalités
  - Transformation
- SQL DDL
  - Création de tables
  - Clés
  - Contraintes d'intégrité
  - Contraintes actives
  - Vues et schémas externes

Formes normales
Introduction

- Un schéma peut être défini par un ensemble d'attributs (le « dictionnaire des données »)

  Ex : `idcours, titre, type, volh, salle, capacite`

- On peut construire une base de données en définissant une unique relation (dite « universelle »)

Idcours	***Titre***	***Type***	***volh***	***Salle***	***cap***
POO	*Prog..*	*TP*	*18*	*T20*	*20*
BDD	*Bases..*	*CM*	*21*	*C31*	*24*
BDD	*Bases..*	*TP*	*14*	*T20*	*21*

`Cours(idcours,titre,type,volh,salle,cap)`

Les Propriétés d'une relation ?
Il *n'existe pas de lignes de la table identiques.*
L'ordre des lignes est quelconque.
L'ordre des colonnes de la table est quelconque.
Chaque ligne doit pouvoir être identifiée de façon unique.

Mais on s'expose à plusieurs anomalies potentielles

Formes normales
Introduction (suite)

- Anomalie de modification : --> Incohérence

Idcours	Titre	Type	volh	Salle	cap
POO	Prog..	TP	18	T20	**20**
BDD	Bases..	CM	21	C31	24
BDD	Bases..	TP	14	T20	**21**

- Anomalie d'insertion:

Idcours	Titre	Type	volh	Salle	cap
POO	Prog..	TP	18	T20	20
BDD	Bases..	CM	21	C31	24
BDD	Bases..	TP	14	T20	21
WEB	Archi..	CM	14	**?**	**?**

- Anomalie de suppression:

Idcours	Titre	Type	volh	Salle	cap
POO	Prog..	TP	18	T20	20
BDD	Bases..	CM	21	T20	21

La salle C31 a disparu avec le cours

Formes normales
Introduction (suite)

- Certaines dépendances entre les données permettent :
  - d'éviter les problèmes de mise à jour
  - de compacter l'espace nécessaire
  - d'optimiser les accès

  Il faut pour cela décomposer la relation universelle

- Y a-t-il une décomposition idéale ?

- Si oui comment la construire ?

Formes normales
Définition

- Une dépendance fonctionnelle entre deux ensembles d'attributs X et Y, notée X -> Y, signifie :

  si $(x1,y1) \in R$ et $(x1,y2) \in R$; alors $y1 = y2$

  Autres formulations :
  - il existe une fonction qui permet de calculer Y à partir de X
  - la valeur de Y est parfaitement déterminée lorsqu'on connaît celle de X
  - On dit aussi que X détermine Y.

- X est **la source** de la DF et Y est **le but**

- Exemple de DF (sur ex. précédent) :
  - `idcours -> titre`
  - `salle -> cap`
  - `idcours, type -> volh, salle`

Formes normales
Définition

- **DF simple** ***:*** On dira qu'une DF est simple si sa source n'est composée que d'un seul attribut.
  - Une DF simple caractérise une *entité (table)* dont la «*source*» est la clé et dont les propriétés sont constituées par le «*but*» de la DF.
- **DF composée** ***:*** On dira qu'une DF est composée si sa source est composée par la réunion de plusieurs attributs.
  - Une DF composée caractérise une *association entre entités (tables)* dont la source est la clé et dont les propriétés sont constituées par le but de la DF.
    - ✓ Exemple
      ```
      idcours, type -> volh, sal
      ```
  - Il ne doit pas y avoir d'attributs superflus dans la source d'une DF composée.
    ```
    idcours, type, sal -> volh
    ```

Formes normales
Définition (suite)

- Les dépendances fonctionnelles ont certaines propriétés
  - si X ->Y, alors XZ ->YZ (augmentation)
  - si X ->Y et Y ->Z alors X ->Z (transitivité)
  - XY->X (et XY ->Y) (évidence)

  Ce sont les axiomes d'Armstrong

  On peut aussi déduire :
  - si X ->Y et X ->Z alors X ->YZ (union)
  - si X ->YZ alors X ->Y et X ->Z (décomposition)
  - si X ->Y et YZ ->W alors XZ ->W (pseudo-trans.)
  - si X ->Y et Z ->W alors XZ ->YW (accumulation)

et beaucoup d'autres

Formes normales
Définition (suite)

- Une clé est un ensemble d'attributs qui déterminent l'ensemble des attributs de la relation

  `K` est une clé de **`R`** ssi
  1. pour tout attribut de R on la DF `K -> A`

- Une clé est une clé *minimale*

  `K` est une *clé minimale* ssi
  1. **`K`** est une *clé*
  2. et aucun sous ensemble strict de K n'est une *clé* de R

- Les formes normales imposent des conditions sur les dépendances fonctionnelles entre les attributs de la clé et les autres

Formes normales
1FN, 2FN, 3FN

- La première forme normale est une définition. Une relation est en 1NF si:
  - elle a une clé (éventuellement totale)
  - tous les attributs sont atomiques
  - Les attributs sont constants dans le temps

- Un attribut *atomique* ne peut prendre que des valeurs « simples » : pas de liste ou d'ensemble, ou de structures de taille variable (arbres, graphes, etc.)
- Pas de structuration interne :

Nom	*Prenom*	*adresse*			*ddn*
		Rue	*Cp*	*Ville*	
…	…	…	…	…	…

devient

nom	*Prenom*	*Rue*	*Cp*	*Ville*	*ddn*
…	…	…	…	…	…

Formes normales
1FN, 2FN, 3FN

- Si on veut représenter un ensemble de valeurs : on multiplie les n-uplets

Etud	*Cours*	*notes*
Jean	*BDD*	*{12,14,11}*

->

Etud	*Cours*	*note*
Jean	*BDD*	*12*
Jean	*BDD*	*14*
Jean	*BDD*	*11*

- Si on veut conserver l'ordre, on ajoute un entier :

Etud	*Cours*	*notes*
Jean	*BDD*	*{12,14,11}*

->

Etud	*Cours*	*n*	*note*
Jean	*BDD*	*1*	*12*
Jean	*BDD*	*2*	*14*
Jean	*BDD*	*3*	*11*

- La notion de « type simple » varie d'un SGBD à l'autre

Formes normales
1FN, 2FN, 3FN (suite)

- Un ensemble de relations est en deuxième forme normale si et seulement si :
  - elle est en première forme normale
  - aucun attribut ne dépend d'une partie stricte d'une clé

- Exemple : inscription des étudiants aux cours

idetud	*cours*	*Nom*
1	*BDD*	*Dupont*
1	*Prog*	*Dupont*

On a *idetud* -> Nom. Donc on scinde en :

Idetud	*Nom*
1	*Dupont*

Idetud	*cours*
1	*BDD*
1	*Prog*

On a éliminé la redondance, et le risque de supprimer un étudiant avec son inscription

Formes normales

1FN, 2FN, 3FN (suite)

- Pour des raisons de cohérence de la base de données, il faut ajouter une « contrainte d'intégrité » :

- Une contrainte d'intégrité fonctionnelle (en abrégé : CIF) se définit par le fait que l'une des entités participant à l'association est complètement déterminée par la connaissance d'une ou plusieurs autres entités participant dans cette même association.

  Exemple:
  avec `representant(idrepr,nom)` et `commande = (idcom,date,idrep)`.
  Signifie : les répresentants doivent être dans la table *`representant`* avant d'entrer dans la table *`commande`*.

- Dans la pratique, on utilisera une clé étrangère : une colonne dont la valeur doit faire référence à la clé primaire d'une relation.

Formes normales
1FN, 2FN, 3FN (suite)

- Un ensemble de relations est en troisième forme normale si et seulement si :
  - elle est en deuxième forme normale
  - les attributs non-clé dépendent directement d'une clé

- Exemple :

étudiant	*bourse*	*tarif*
Jean	*Exc*	*100*

La clé est *etudiant*. Mais on a *bourse -> tarif* .

Donc on scinde en :

étudiant	*bourse*
Jean	*Exc*

bourse	*tarif*
Exc	*100*

Formes normales

1FN, 2FN, 3FN (suite)

- Pour des raisons de cohérence, on définit également une contrainte d'intégrité.

  Exemple:
  avec *Statut* = (*etudiant, bourse*) et *Tarifs* = (*bourse, tarif*),
  et donc *Statut.bourse* sera une *clé étrangère*

- 2FN et 3FN s'occupent des dépendances des attributs non-clés Y, c'est-à-dire des dépendances de la forme :
  - X ->Y, avec :
    - ✓ X partie stricte d'une clé (2FN)
    - ✓ Y non-clé (3FN)
- En règle générale, on peut construire directement des relations en 3FN.

Formes normales
1FN, 2FN, 3FN (suite)

- Un ensemble de relations est en forme normale de Boyce-Codd si et seulement si :
  - elle est en troisième forme normale
  - toutes les dépendances sont de la forme K -> A où K est la clé et A un attribut non clé

- Exemple :

numetud	*secsoc*	*tuteur*
100	*16701..*	*Pierre*
101	*26702..*	*Pierre*
102	*26703..*	*Alain*

Il y a deux clés candidates : (*numetud, tuteur*) et (*secsoc, tuteur*).
Mais on a numetud -> secsoc (et aussi secsoc -> numetud).

Formes normales
1FN, 2FN, 3FN (suite)

- Donc on scinde en :

numetud	*tuteur*
100	*Pierre*
…	*…*

numetud	*secsoc*
100	*16701..*
…	*…*

- Ici le problème se pose de deux façons distinctes :
  - si on considère la clé (*numetud, tuteur*), alors *numetud -> secsoc* pose problème
  - si on considère la clé (*secsoc, tuteur*), alors *secsoc -> numetud* pose problème

- En fait, les deux attributs *numetud* et *secsoc* sont « équivalents » (car numetud <-> secsoc)
  Le plus simple est peut-être d'en supprimer un...

- Remarque : la FNBC s'occupe des dépendances des attributs appartenant aux clés.

Formes normales
Autres formes normales

- La quatrième forme normale n'utilise pas les dépendances fonctionnelles, mais les dépendances multi-valuées.

- Une dépendance multi-valuée indique l'indépendance de deux ensembles d'attributs non clés.

Exemple :

Nom	*Diplôme*	*langage*
Jean	*Licence*	*C*
Jean	*Licence*	*Java*
Jean	*Master*	*C*
Jean	*Master*	*java*

- Ici *diplome* et *langage* sont indépendants : on écrit
  - nom->->diplome (ou nom->->langage, ce qui est équivalent)

- Si on ajoute un langage à Jean, il faut insérer deux lignes *((Jean,Licence,Python) et (Jean,Master,Python))*

Formes normales
Autres formes normales

- Dans ce cas ,on scinde la relation en

Nom	*Diplôme*
Jean	*Licence*
Jean	*Master*

Nom	*langage*
Jean	*C*
Jean	*Java*

- La cinquième forme normale vise à supprimer toutes les dépendances « *de jointure* » :
  peut-on scinder une relation et la reconstruire en faisant une jointure sur les relations résultant de la scission ? Cette forme normale ne s'applique que dans de **rare cas**...

- Il existe même une sixième forme normale...

Entités et associations

- La théorie des formes normales est une démarche de conception :
  - énumérer tous les attributs présents
  - scinder des relations jusqu'à obtenir la FN désirée

- Ce n'est pas satisfaisant pour plusieurs raisons :
  - en général, trop d'attributs à manipuler
  - trop formelle : l'intuition permet de gagner du temps

- Le besoin de discuter/documenter/communiquer un schéma appelle des méthodes plus « graphiques »

- Il existe différents « diagrammes » de conception : Merise/MCD, Data Flow Diagrams, UML/Class etc. tous sensiblement équivalents.

  Nous appellerons « entités-associations » ces diagrammes.

Entités et associations
Définitions

- On appelle entité un type d'objet à représenter, et instance d'entité un objet particulier.

  Exemple : *Jean* est une instance de l'entité *Etudiant*

- On appelle attribut une variable servant à décrire une entité

  Exemple : *Jean* a un numéro d'étudiant, une *adresse*

- On appelle clé un (ensemble d') attribut(s) permettant d'identifier les instances d'une entité

  Exemple : le numéro d'étudiant est propre à chaque étudiant

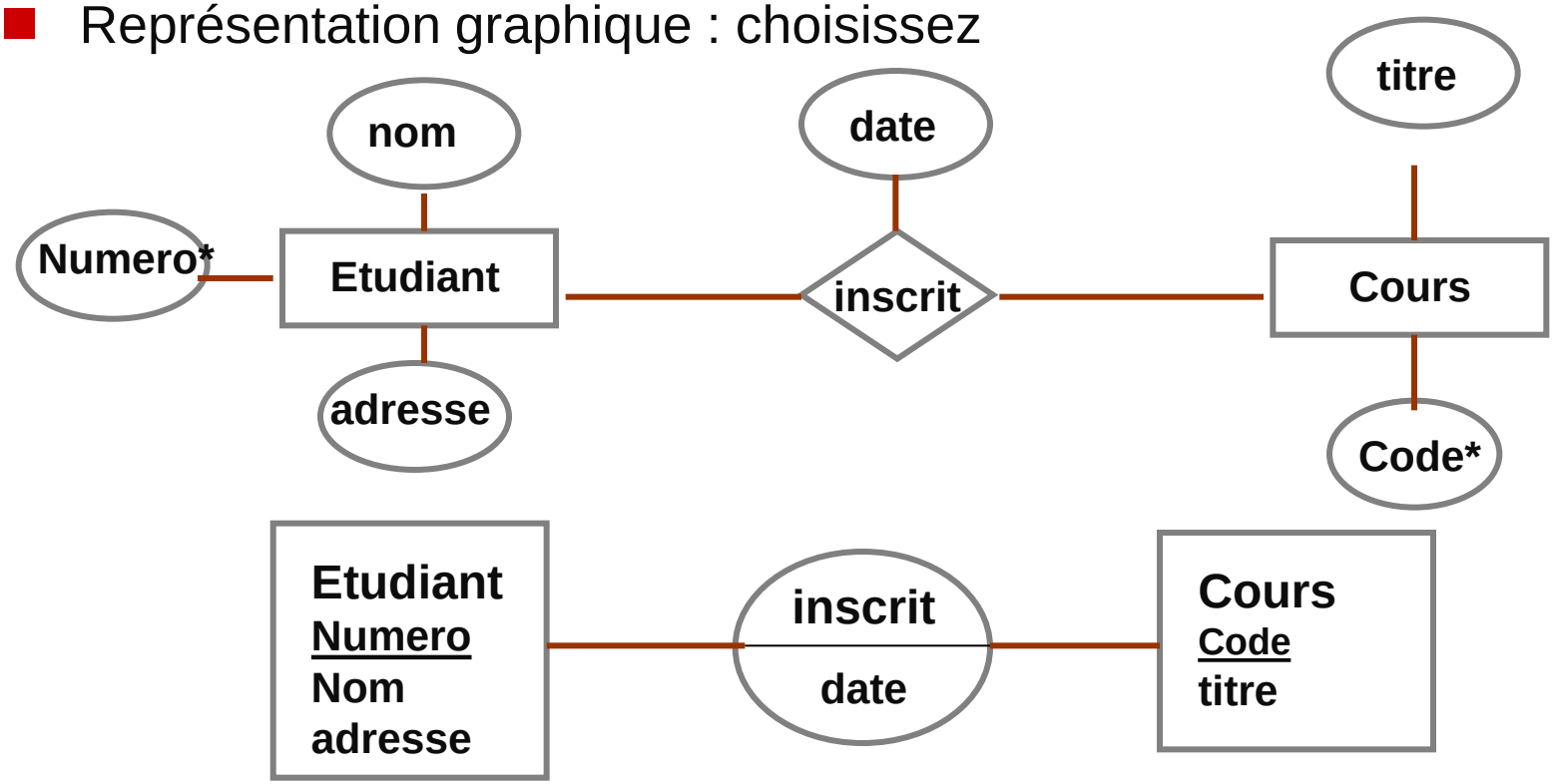
- On appelle association un lien entre une, deux ou plusieurs entités, éventuellement porteur d'attributs

  Exemple : un étudiant peut **s'inscrire** à plusieurs cours

Entités et associations
Définitions (suite)

- Intuition (qui vaut ce qu'elle vaut) : dans une description en langage naturel
  - les entités sont des noms
  - les associations sont des verbes

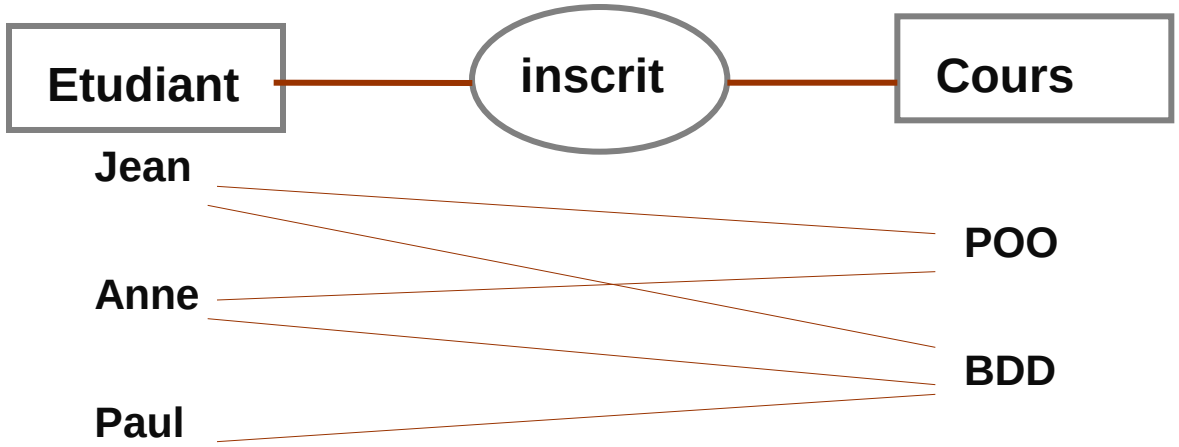
- Représentation graphique : choisissez



Entités et associations

Types et instances

- Il est important de faire la distinction entre types (d'entité, d'association) et instances
- Le type d'entité, d'association est descriptif : c'est une structure de données
- L'instance est une réalisation particulière d'un type d'entité : elle doit comporter une valeur pour chaque attribut.
- Pour les associations : l'instance de l'association est liée à autant d'instances d'entités que nécessaire :



Les 5 instances de l'association (+ attributs)

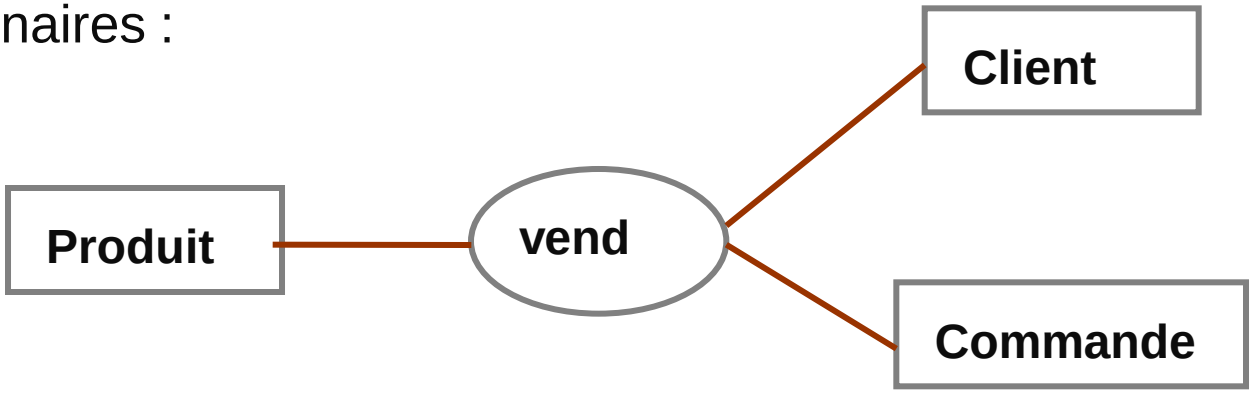
Entités et associations

Degré des associations

- Le degré (ou dimension) d'une association est le nombre d'entités qu'elle lie.
- Le cas le plus fréquent est celui des associations binaires :

- On peut avoir des associations unaires :

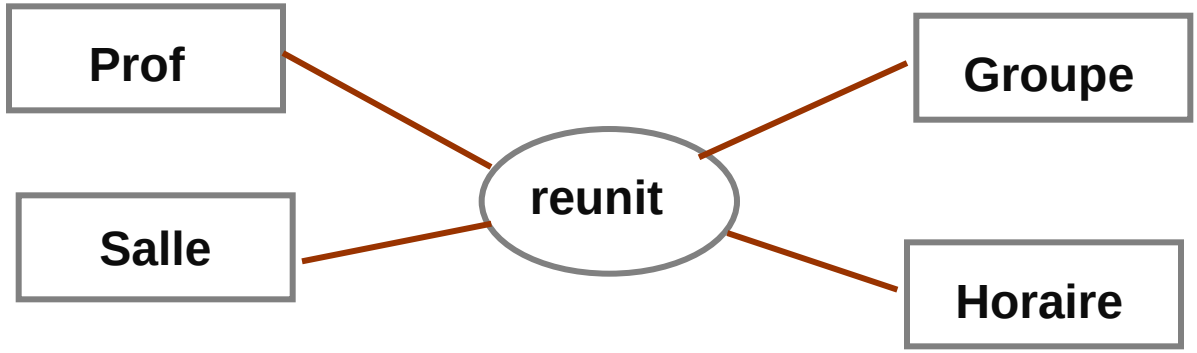
- Ou ternaires :



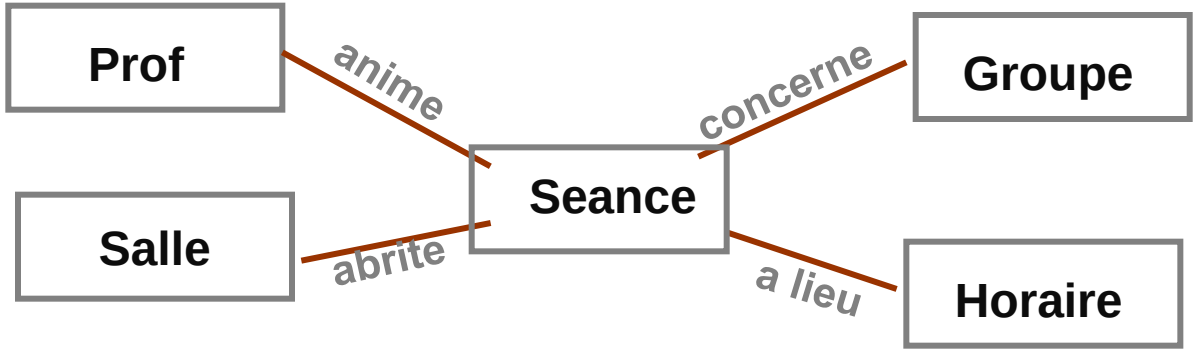
Entités et associations

Degré des associations (suite)

- Voire quaternaires :

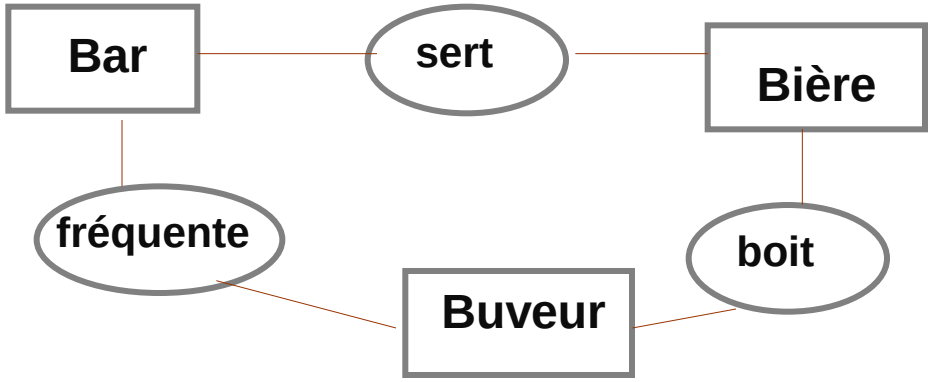


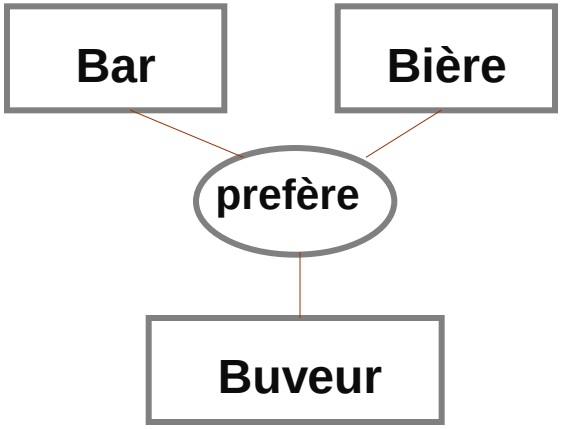
- La réification consiste à utiliser une entité plutôt qu'une association de degré > 2



Entités et associations
Degré des associations (suite)

- Interprétation :





Nuance :

- « un buveur préfère certaines bières dans certains bars »
- le modèle à 3 associations ne peut pas exprimer cela.

Entités et associations
Cardinalités

- Les cardinalités d'une association indique le nombre de fois qu'une instance d'une entité associée peut intervenir dans une instance de l'association, au minimum et au maximum.

- « Un prof peut donner un ou plusieurs cours. Chaque cours est affecté à un unique prof. »

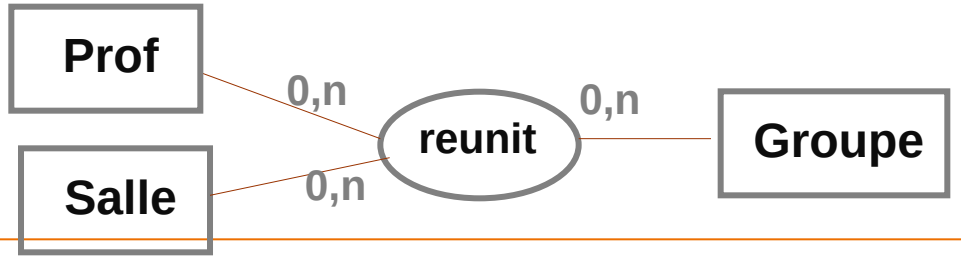
- Les cardinalités vont par deux : minimum et maximum
  - 0,1 : association optionnelle
  - 1,1 : association unique obligatoire
  - 0,n : un nombre quelconque
  - 1,n : un ou plusieurs

Entités et associations
Cardinalités (suite)

- La place des cardinalités varie selon les méthodes
- MERISE (modèle conceptuel des données) :

- UML (diagramme de classes – simplifié ici)

- Nous choisirons de placer les cardinalités près de l'association (voir transparent précédent)
- Dans le cas des associations de degré > 2 :



Entités et associations

Transformation

- Un modèle entités-associations peut être transformé en modèle relationnel automatiquement.

- Chaque entité donne lieu à une relation :
  - tous les attributs de l'entité sont transformés en attributs de la relation
  - normalement, la clé est parfaitement identifiée

  C'est à ce stade que se fait le choix des types d'attribut

- La transformation des associations dépend de leur cardinalité, mais:
  - chaque entité a déjà donné lieu à une relation
  - et cette relation a une clé définie

- La transformation des associations utilise la notion de *clé étrangère :* un (ou plusieurs) attributs permettant de localiser une ligne dans une autre table, donc de même type que la clé référencée

Entités et associations
Transformation (suite)

- Le mécanisme le plus simple consiste à définir une relation pour chaque association :

- donnera les relations suivantes :

```
Prof (cleprof, …)
Cours (clecours, …)
donne (cleprof , clecours, …)
```

- Dans *donne*, les attributs *clecours* et *cleprof* sont des clés étrangères (chacun).
  La clé de *donne* est le couple (*cleprof* , *clecours*).

- Il y a des contraintes d'intégrités à respecter : les valeurs de *cleprof* dans *donne* doivent apparaître dans la table *Prof*.

Entités et associations
Transformation (suite)

- Le problème est que ce principe conduit à créer beaucoup de tables.

- Selon les cardinalités, la création de table peut être évitée :

On peut construire le schéma :

Prof (cleprof,…)
Cours (clecours,…, cleprof)

(un cours est donné par un seul prof)

- Ici, pas de nouvelle table, une clé étrangère (*cleprof*).

Entités et associations
Transformation (suite)

- Le fait de procéder par augmentation (quand c'est possible) a deux avantages principaux :
  - moins de relations (donc simplicité du schéma relationnel)
  - à l'usage, moins de jointures à faire...
  - ... et plus faciles à optimiser

Par exemple : titre des cours données par Tournesol

- Premier cas (trois tables) :

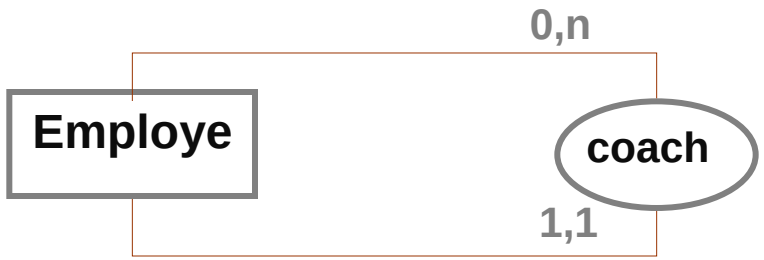
```
SELECT cours.titre FROM prof,cours,donne
  WHERE prof.nom = 'Tournesol'
      AND prof.cleprof = donne.cleprof
      AND donne.clecours = cours.clecours
```

- Second cas (deux tables) :

```
SELECT cours.titre FROM prof,cours
  WHERE prof.nom = 'Tournesol'
      AND prof.cleprof = cours.cleprof
```

Entités et associations
Transformation (suite)

- Le cas des associations unaires entre dans le cadre général :



(un employé coache entre *0* et *N* employés, chaque employé est coaché par exactement un employé)

Employe(idemp,…,idempcoach)

idempcoach est une clé étrangère vers... *Employe*

Entités et associations
Transformation (suite)

- Les associations 1–1 laissent le plus de liberté

peut donner lieu à

- *1 cadre(idc,…) et Portable(idp,…,idc)*
- *2 cadre(idc,…,idp) et Portable(idp,…)*
- *3 cadre(idc,…), Portable(idp,…) et dote(idc, idp)*
- *4 cadre(idc,…,idp,…) (fusion)*

- Seule la dernière est totalement correcte, car :
  - *1 certains cadres peuvent ne pas être dotés*
  - *2 certains portables peuvent ne pas être attribués*
  - *3 les deux simultanément*

- Ces défauts existent dans tous les cas, par exemple : *suit(ide, idc)* ne garantit pas qu'un étudiant suive au moins un cours, ni qu'un cours soit suivi par au moins un étudiant

SQL DDL

- Lorsqu'on dispose d'un modèle relationnel, on peut le mettre en oeuvre à l'aide d'un SGBD relationnel

- SQL définit un Data Description Language (DDL) permettant de :
  - définir les relations, leurs attributs, les types de ces attributs
  - indiquer les clés (dites primaires)
  - indiquer les attributs qui constituent des clés étrangères

- SQL permet aussi de préciser une certaine classe de contraintes d'intégrité

SQL DDL
Création des tables

- On crée une table avec :

```
CREATE TABLE nom (
def-colonne [ , def-colonne ]*
[ , contrainte-de-table ]*
)
```

- Une colonne est définie par :

```
nom type [ contrainte-de-colonne ]*
```

(les colonnes doivent avoir un nom unique dans la table)

- On détruit une table avec :

DROP TABLE nom ;

Mais c'est très rare hors période de test

- On modifie une table avec :

ALTER TABLE nom ...

Mais les possibilités sont très variables selon les SGBD

SQL DDL
Création des tables (suite)

- Les types de données numériques SQL « standard » sont :

`BOOLEAN`	TRUE ou FALSE
`INTEGER`	entier (4 octets)
`SMALLINT`	entier (2 octets)
`REAL`	flottant (4 octets)
`DOUBLE RECISION`	flottant (8 octets)
`NUMERIC [ (`*p*`,`*s*`) ]`	précision arbitraire, avec *p* chiffres au total, dont *s* après la virgule

SQL DDL
Création des tables (suite)

- Les types de données textuelles sont :

`CHARACTER(n)`	chaîne de taille fixe (remplie avec des blancs si nécessaire).
`VARCHAR(n)`	chaîne de taille variable, au maximum n.

- Dates et heures sont représentées par :

`DATE`	date calendaire
`TIME`	heure (du jour)
`TIMESTAMP`	date et heure

SQL DDL
Création des tables (suite)

- Une seule règle : les types seront différents d'un SGBD à l'autre. RTFM.
- PostgreSQL propose aussi les types :

`Inet`	Adresse réseau + masque
`Macaddr`	Adresse MAC (6 octets)
`Point`	(x,y)
`Polygon`	$[(x_1,y_1),\ldots]$
…	

- Oracle propose également :

`BLOB`	Binary large object
`CLOB`	Character large object

SQL DDL
Création des tables (suite)

- Exemple :

```
CREATE TABLE etudiant (
    numet CHAR(10),
    nom VARCHAR(80),
    prenom VARCHAR(60),
    adresse1 VARCHAR(80),
    adresse2 VARCHAR(80),
    ddn DATE,
    photo BLOB)

CREATE TABLE note (
    numet CHAR(10),
    codecours VARCHAR(8),
    note NUMERIC(5,2),
    dobt DATE)
```

SQL DDL
Clés

- La clé (unique) d'une table est indiquée par PRIMARY KEY
- Si la clé est composée d'une unique colonne :

```
CREATE TABLE etudiant (
   numet CHAR(10) PRIMARY KEY,
   ...
   )
```

- Si la clé est composite :

```
CREATE TABLE suit (
   numet CHAR(10),
   codecours VARCHAR(8),
   PRIMARY KEY (numet,codecours)
   ...
   )
```

SQL DDL
Clés (suite)

- Une clé étrangère est indiquée par `REFERENCES`

```
CREATE TABLE cours (
   codecours VARCHAR(8) PRIMARY KEY,
   nprof VARCHAR(10)
     REFERENCES prof(numprof),
   ...
   )
```

Le nom de la clé étrangère peut être différent de la clé référencée.

En général on peut omettre le nom de la colonne après `REFERENCES` :

```
nprof VARCHAR(10) REFERENCES prof
```

SQL DDL
Clés (suite)

- Dans le cas composite

```
CREATE TABLE equivalence (
    codecours1 VARCHAR(8),
    codefac1 VARCHAR(4),
    codecours2 VARCHAR(8),
    codefac2 VARCHAR(4),
    PRIMARY KEY (codecours1,codefac1,
                 codecours2,codefac2),
    FOREIGN KEY (codecours1,codefac1)
      REFERENCES offre(cours,fac),
    FOREIGN KEY (codecours2,codefac2)
      REFERENCES offre(cours,fac)
)
```

SQL DDL
Contraintes d'intégrité

- Une contrainte d'intégrité est une contrainte que les données doivent respecter. Le SGBD vérifie (ou pas) ces contraintes. Le but est d'assurer la cohérence des données.

- Dans la pratique, les directives :
  - `PRIMARY KEY`
  - `FOREIGN KEY ... REFERENCES ...`

  sont considérées comme des contraintes d'intégrité

- Objectif : décharger les applications clientes de la vérification de ces contraintes

SQL DDL
Contraintes d'intégrité (suite)

- La contrainte `NOT NULL` impose qu'une colonne ait une valeur (ce qui n'est pas le cas par défaut).

```
CREATE TABLE etudiant (
    numet CHAR(10) NOT NULL PRIMARY KEY,
    nom VARCHAR(80) NOT NULL,
    ...
    )
```

Consensus : `NOT NULL` devrait être active par défaut

- La contrainte `DEFAULT` fournit une valeur par défaut :

```
CREATE TABLE etudiant (
    ...
    boursier BOOLEAN DEFAULT FALSE
    )
```

N'empêche pas d'insérer `NULL` explicitement...

SQL DDL
Contraintes d'intégrité (suite)

- La contrainte `CHECK` permet de tester n'importe quelle condition.

```
CREATE TABLE employe (
    ...
    salaire NUMERIC(6,2),
    commission NUMERIC(6,2),
    CHECK ( commission <= salaire )
    )
```

- `CHECK` ne peut porter que sur les colonnes de la table. Elle est évaluée pour la ligne en cours d'insertion ou modification. Elle ne peut pas contenir de requêtes.

SQL DDL
Contraintes d'intégrité (suite)

- La contrainte `UNIQUE` permet de préciser qu'une (ou plusieurs) colonne(s) doivent prendre des valeurs toutes distinctes.

```
CREATE TABLE buddies (
    email VARCHAR(80)
      NOT NULL PRIMARY KEY,
    nickname VARCHAR(40)
      NOT NULL UNIQUE,
    ...
    )
```

- Une colonne `UNIQUE` n'est pas une clé : elle n'est pas référençable par une autre table.

- Éventuellement coûteux...

SQL DDL
Contraintes actives

- Les contraintes d'intégrité référentielle (REFERENCES) sont des contraintes structurelles, qui portent sur plusieurs tables en général.

- Exemple :

Prof

Numprof	**nom**	**…**
1	Pierre	…

Cours

numc	**titre**	**…**	**nprof**
17	PROG	…	1
42	BDD	…	3

- Oracle propose également :Que se passe-t-il si on supprime un prof qui assure un cours ?

  -> Une erreur, le SGBD refuse de briser l'intégrité référentielle.

SQL DDL
Contraintes actives (suite)

- Il est possible de paramétrer un comportement par défaut. Il y a deux possibilités :
  - supprimer toutes les lignes qui font référence à la ligne qui disparaît

```
CREATE TABLE cours (
    ...
    nprof VARCHAR(10)
       REFERENCES prof(numprof),
          ON DELETE CASCADE
    ...
    )
```

  - remplacer par NULL les références à la ligne qui disparaît

```
nprof VARCHAR(10)
    REFERENCES prof(numprof),
       ON DELETE SET NULL
```

SQL DDL
Contraintes actives (suite)

- Note : dans ce cas, ce n'est sûrement pas une bonne idée de supprimer automatiquement les cours dont le prof disparaît...

En règle générale :

- Des mécanismes indispensables pour ce qui concerne :
  - clés primaires
  - intégrité référentielle
  - `NOT NULL`

- Des mécanismes occasionnellement utiles pour :
  - valeurs par défaut
  - unicité
  - tests simples

- En général insuffisants pour implanter les contraintes d'intégrité dégagées pendant la conception

SQL DDL
Vues et schémas externes

- Une vue est un nom donnée à une requête :

```
CREATE VIEW nom [ ( col [ , col ]*) ]
    AS requete
    [ WITH CHECK OPTION ]
```

- Si les noms de colonne ne sont pas présents, ils sont déduits de la requête. Dans la requêtes les noms de colonnes peuvent être fixés avec AS.
- La vue s'utilise comme une table.
- La vue est instantanée : elle reflète les données actuelles.
- Les « schémas externes » prennent souvent la forme d'ensembles de vues.

SQL DDL
Vues et schémas externes (suite)

- Dans certains cas, on peut insérer/modifier des données *via* la vue dans les tables sous-jacentes : `WITH CHECK OPTION` impose que les données insérées/modifiées soient visibles dans la vue.

- On peut insérer/modifier/supprimer via une vue si :

  - la requête n'utilise pas `GROUP BY, DISTINCT, ORDER BY,` ou une fonction d'agrégation après `SELECT`
  - l'opération porte sur une seule table en cas de jointure dans la requête, et la vue contient suffisamment de colonnes
  - le résultat de la requête ne contient pas d'expressions, mais uniquement des valeurs de colonne (pour la table visée)

  (plus des limitations spécifiques aux SGBD)

SQL DDL
Vues et schémas externes (suite)

- Exemple :

```
CREATE VIEW roles AS SELECT
   UPPER(nome), nume, role, nump, nomp
   FROM employe, participe, projet
   WHERE ...
```

(insertion pour `participe`, modification sur `participe` et `projet` sauf colonne `role`, suppression sur les trois tables mais conditions restreintes aux colonnes présentes)

- On a souvent accès à l'utilisateur courant, la date, etc. :

```
CREATE VIEW myops AS SELECT
   d, texte FROM journal
   WHERE util = CURRENT_USER()
    AND CURRENT_DATE()-d < 15
```

Quatrième Partie

Transactions, Sécurité