Guide Modèle relationnel

Employees (<u>firstname</u>, salary, address, dept) */* firstname est l'identifiant */*
/ <n, s, a, d> ∈ Employees ⟺ l'employé identifié par son nom n gagne un salaire s. Il habite à l'adresse a et est affecté au rayon d. */*
Leaderships (<u>boss</u>, <u>dept</u>) */* 2 identifiants : boss et dept */*
/ <b, d> ∈ Leaderships ⟺ l'employé b est responsable du rayon d. */*
Contraintes d'intégrité référentielle :
Leaderships[boss, dept] ⊆ Employees[firstname, dept]

Employees

firstname	salary	address	dept
John	120	Randwick	Toys
Mary	130	Wollongong	Furniture
Peter	110	Randwick	Garden
Tom	120	Botany Bay	Toys

Leaderships

boss	dept
John	Toys
Mary	Furniture
Peter	Garden

Modèle relationnel	- Une base de données relationnelle est constituée d'un ensemble de relations (tables) - Chaque relation contient les données relatives à des entités de même nature (ex. employés d'une entreprise) - Chaque ligne (n-uplet) d'une relation reprend les données relatives à une entité - Les lignes d'une relation sont distinctes - Chaque colonne d'une relation décrit une propriété commune des entités (attributs) - On évite de stocker les informations qui peuvent être calculées (ex. *age*)
Concepts importants	- Un **domaine** est un ensemble de valeurs atomiques (ex. *entiers > 100*) - Un **attribut** indique le rôle joué par un domaine dans une relation. (ex. *domaine(salary) = entiers > 100*) - La structure de la relation (**schéma**) est donnée par son nom et par un ensemble d'attributs (ex. *Employees (firstname, salary, address, dept))* - La **spécification** d'une relation est un prédicat utile pour comprendre le schéma de la relation et le documenter.
Contraintes	- **Domaine** : contraint les types des attributs (ex. *le salaire doit être un entier plus grand que 100*) - **Identification** : définit une contrainte d'unicité (clé) (ex. *un employé est identifié par son prénom*) - **Intégrité référentielle** : définit une restriction d'un ensemble de possible valeurs d'une relation par rapport à une autre. (ex. *tous les chefs doivent être des employés*) - **Autres** : autres contraintes (ex. *le salaire des employés ne peut pas diminuer lors d'une mise à jour*)
Opérations de base	**PROJECTION** (SELECT en SQL) **Employees** **SELECTION** (WHERE en SQL) firstname / salary / address / dept John / 120 / Randwick / Toys Mary / 130 / Wollongong / Furniture Peter / 110 / Randwick / Garden Tom / 120 / Botany Bay / Toys

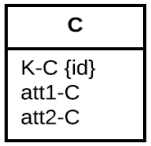
Aide-mémoire SQL

PROJECTION D'ATTRIBUTS	
SELECT	SELECT Table.attribut / SELECT attribut / SELECT *
FROM	FROM Table1
DISTINCT	SELECT DISTINCT attribut (enlève les possibles doublons)
SELECTION DE LIGNES	
WHERE	WHERE attribut
LIKE	WHERE attribut LIKE chaîne de caractères (ou expression régulière)
AND	AND attribut = Valeur
OR	OR attribut = Valeur
= <> < > => <=	attribut *operateur* Valeur
IS [NOT] NULL	WHERE attribut IS [NOT] NULL
TRI	
ORDER BY … ASC ou DESC	ORDER BY attribut1 DESC, attribut2 ASC
OPERATEURS ENSEMBLISTES	
UNION	Requête UNION Requête *(besoin de schémas compatibles)**
INTERSECT	Requête INTERSECT Requête *(besoin de schémas compatibles)**
EXCEPT (MINUS)	Requête EXCEPT Requête *(besoin de schémas compatibles)**
SOUS-REQUETES D'INTERSECTION	
IN	WHERE (attribut1, attribut2, …) IN (Requête) *(besoin de schémas compatibles)**
NOT IN	WHERE (attribut1, attribut2, …) NOT IN (Requête) *(besoin de schémas compatibles)**
JOINTURES	
JOIN ON	FROM Table1 JOIN Table2 ON (Table1.attribut1 = Table2.attribut2)
JOIN ON (même table → besoin de renommer)	FROM Table1 TX JOIN Table1 TY ON (TX.attribut1 = TY.attribut1 AND TX.attribut2 <> TY.attribut2)
JOIN USING	FROM Table1 JOIN Table2 USING (attribut)
~~NATURAL JOIN~~ (à éviter)	~~FROM Table1 NATURAL JOIN Table2~~
CALCULS	
COUNT	SELECT COUNT (Table.attribut) AS nomAlias
SUM	SELECT SUM (Table.attribut) AS nomAlias
AVG	SELECT AVG (Table.attribut) AS nomAlias
MIN	SELECT MIN (Table.attribut) AS nomAlias
MAX	SELECT MAX (Table.attribut) AS nomAlias
REGROUPEMENT	
GROUP BY	GROUP BY Table.attribut
HAVING	GROUP BY Table.attribut HAVING Table.attribut = Valeur
SOUS-REQUETES INTERMEDIAIRES	
WITH AS	WITH NomSous-requête AS (Requête) SELECT attribut FROM Table JOIN NomSous-requête USING (attribut);
CREATION ET SUPPRESSION	
CREATE	CREATE TABLE Table1 (Attributs + Contraintes);
DROP	DROP TABLE Table1 ;
MISE À JOUR	
INSERT	INSERT INTO Table (attribut1, attribut2) VALUES (valeur1, valeur2) ;
UPDATE	UPDATE Table1 SET attribut = Nouvelle valeur ;
DELETE	DELETE FROM Table1 WHERE condition ;
*Schémas compatibles	Même nombre d'attributs et types similaires

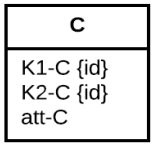
UML Class Diagram* to Relational Model Rules + Example

*Adaptation for DBs

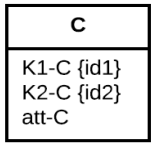
Classes



C(K-C, att1-C, att2-C);



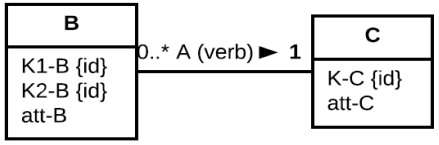
C(K1-C, K2-C, att-C);
(Composed key)



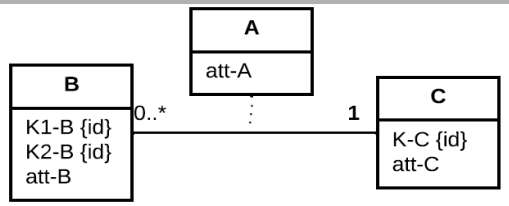
C(K1-C, K2-C, att-C);
(Two candidate keys)

Associations & Association Classes

1 to Many

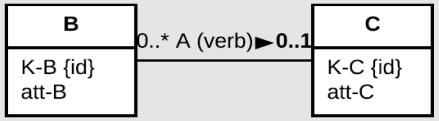


B(K1-B, K2-B, att-B, **K-C**); **C**(K-C, att-C);



B(K1-B, K2-B, att-B, att-A, **K-C**); **C**(K-C, att-C);

0..1 to Many



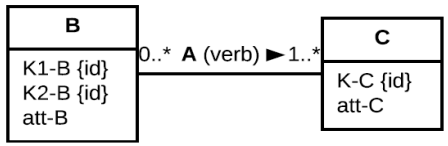
K-C could be NULL **B**(K-B, att-B, **K-C**); **C**(K-C, att-C);

OR

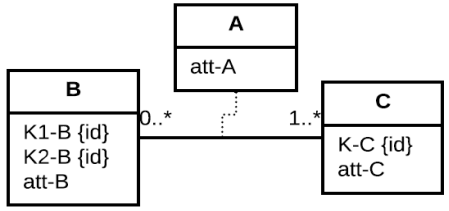
K-C NOT NULL **B**(K-B, att-B); **A**(**K-B**, **K-C**); **C**(K-C, att-C);

this option is generally better

Many to Many

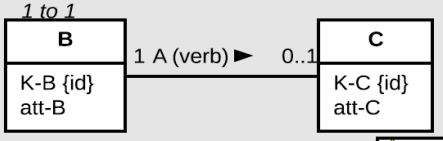


B(K1-B, K2-B, att-B); **C**(K-C, att-C);
A (K1-B, K2-B, K-C)



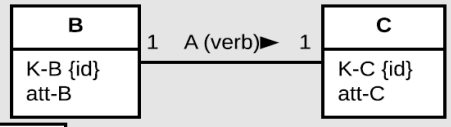
B(K1-B, K2-B, att-B); **C**(K-C, att-C);
A (K1-B, K2-B, K-C, att-A)

1 to 1



B(K-B, att-B); **C**(K-C, att-C, K-B)

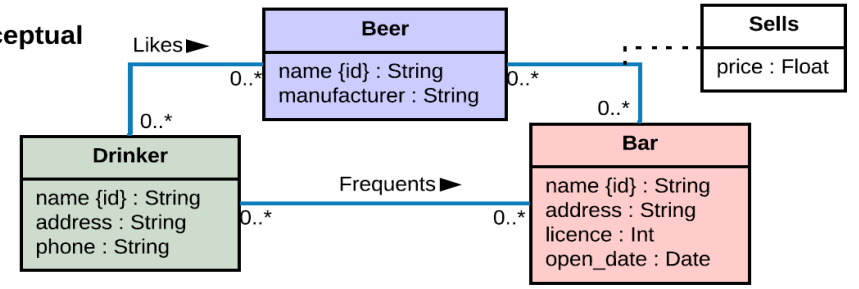
⚠ Not composed key but two candidate keys



B(K-B, att-B); **C**(K-C, att-C, K-B)
OR
B(K-B, att-B, K-C); **C**(K-C, att-C);

+Constraints **+Constraints**

Conceptual



Logical

Beers (beer_name, beer_manufacturer)

Drinkers (drinker_name, drinker_address, drinker_phone)

Bars (bar_name, bar_address, bar_licence, bar_open_date)

Likes (drinker_name, beer_name)

Likes[drinker_name] ⊆ Drinkers[drinker_name]

Likes[beer_name] ⊆ Beers[beer_name]

Frequents (drinker_name, bar_name)

Frequents[drinker_name] ⊆ Drinkers[drinker_name]

Frequents[bar_name] ⊆ Bars[bar_name]

Sells (bar_name, beer_name, sells_price)

sells_price > 0

Sells[beer_name] ⊆ Beers[beer_name]

Sells[bar_name] ⊆ Bars[bar_name]

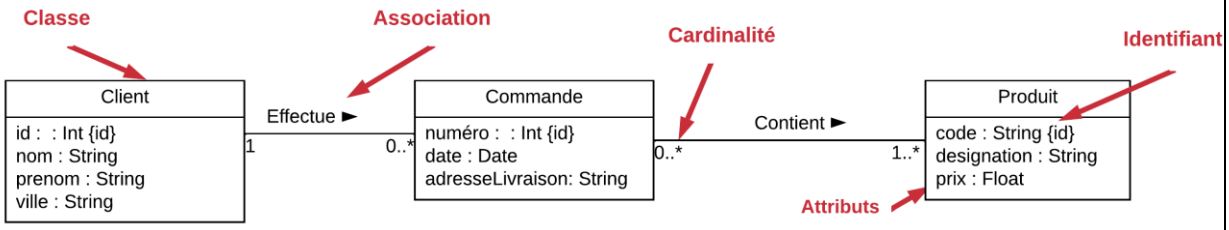
SQL Implementation

```
-- SQLite Syntax
CREATE TABLE Beers (
  beer_name TEXT NOT NULL,
  -- NOT NULL not necessary for PK attributes in some systems such as ORACLE
  beer_manufacturer TEXT,
  CONSTRAINT pk_beers_c0 PRIMARY KEY (beer_name)
);

CREATE TABLE Bars (
      bar_name TEXT NOT NULL,
      bar_address TEXT,
      bar_licence DATE,
      bar_open_date DATE,
      CONSTRAINT pk_bars_c0 PRIMARY KEY (bar_name)
);

CREATE TABLE Sells(
  bar_name TEXT NOT NULL,
  beer_name TEXT NOT NULL,
  sells_price REAL NOT NULL,
  CONSTRAINT pk_sells_c0 PRIMARY KEY (bar_name,beer_name),
  CONSTRAINT fk_sells_c1 FOREIGN KEY (bar_name) REFERENCES Bars(bar_name),
  CONSTRAINT fk_sells_c2 FOREIGN KEY (beer_name) REFERENCES Beers(beer_name),
  CONSTRAINT ck_sells_c3 CHECK (sells_price > 0)
);
```

....

Eléments UML de base et traduction en modèle relationnel (niveau logique)

Niveau Conceptuel



Niveau Logique

Clients (id_client, nom_client, prenom_client, ville_client)

id_client	nom_client	prenom_client	ville_client
1	Bond	James	Londres
2		Nomi	Londres

ProduitsCommandes (Contient)
(numero_commande, code_produit)

numero_commande	code_produit
006	7F5735
007	7F5735
007	9H62DA

Produits (code_produit, designation_produit, prix_produit)

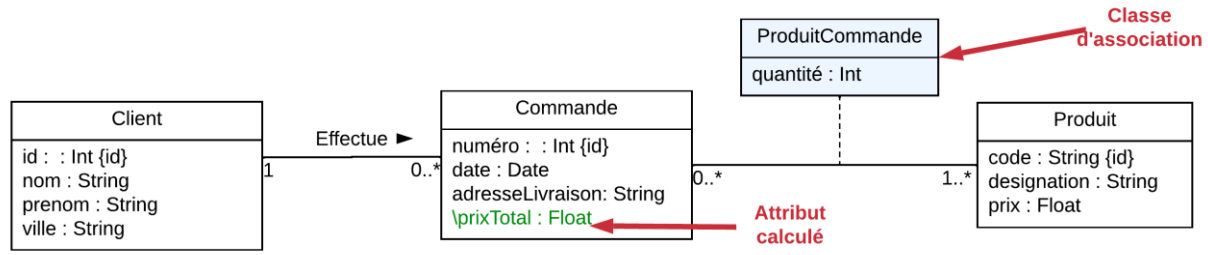
code_produit	designation_produit	prix_produit
7F5735	Pistolet	100
9H62DA	Gilet	50

Commandes (numero_commande, date_commande, adresse_livraison_commande, id_client)

numero_commande	date_commande	adresse_livraison_commande	id_client
006	2022-03-17	Londres	1
007	2022-03-21	Londres	1

Eléments UML plus avancés et traduction en modèle relationnel

Niveau Conceptuel



Niveau Logique

Clients
(id_client, nom_client, prenom_client, ville_client)

id_client	nom_client	prenom_client	ville_client
1	Bond	James	Londres
2		Nomi	Londres

ProduitsCommandes
(numero_commande, code_produit)

numero_commande	code_produit	quantite_produitcommande
006	7F5735	3
007	7F5735	4
007	9H62DA	2

Produits
(code_produit, designation_produit, prix_produit)

code_produit	designation_produit	prix_produit
7F5735	Pistolet	100
9H62DA	Gilet	50

Commandes_base
(numero_commande, date_commande, adresse_livraison_commande, id_client)

numero_commande	date_commande	adresse_livraison_commande	id_client
006	2022-03-17	Londres	1
007	2022-03-21	Londres	1

View Commandes
(numero_commande, date_commande, adresse_livraison_commande, id_client, prix_total_commande)

numero_commande	date_commande	adresse_livraison_commande	id_client	prix_total_commande
006	2022-03-17	Londres	1	300
007	2022-03-21	Londres	1	500

Points importants sur la traduction en relationnel :

- Une association « plusieurs à plusieurs » (many-to-many) est traduite en une table intermédiaire
- Une bonne pratique consiste à avoir un nom conceptuel (UML) et un nom logique (relationnel et SQL)
- Certains outils de modélisations UML ne supportent pas tous les éléments UML (ex. ids, attributs calculés, …)
- Une bonne conception UML produit un bon modèle relationnel (sans redondances)