Formulaire de BDD

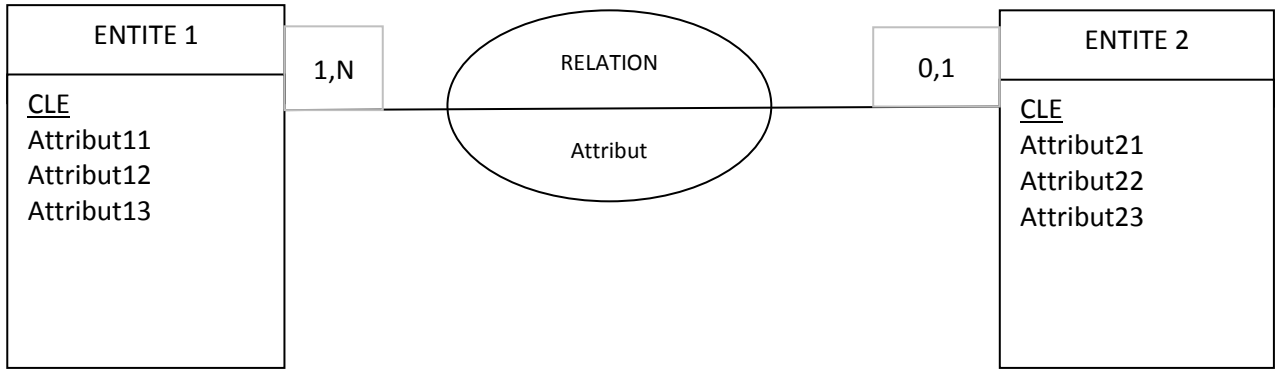
I. Le dictionnaire de données

Le dictionnaire de données permet de recenser toutes les informations utiles au système considéré. Le dictionnaire peut-être modalisé par le tableau suivant :

Libellé de la propriété	Nom du champ	Type	Dimension	Contrainte
Code Client	**Code Client**	**Number**	**3**	**Unique**
Nom Client	**Nom**	**String**	**30**	**-**
Prenom Client	**Prenom**	**String**	**30**	**-**
Adresse Client	**Adresse**	**String**	**50**	**-**
Référence Location	**Ref location**	**Number**	**10**	**Unique**
Date Location	**Date debut**	**Date**	**8**	**-**
Date Retour	**Date fin**	**Date**	**8**	**-**
Code Produit	**Code produit**	**Number**	**10**	**Unique**
Modèle produit	**Modèle**	**String**	**30**	**-**
Coût journalier produit	**Coût journalier**	**Number**	**2**	**-**
Caution Produit	**Caution**	**Number**	**3**	**-**

II. Le MCD

Le MCD permet de représenter les données sous forme de schéma. Il faut dans un premier construire les entités (chacune possède une clé qui permet de repérer de façon unique une donnée), puis ajouter les relations qui existent entre les entités. En plus de cela, il convient d'ajouter la cardinalité qui régit les relations.



III. Le MLD

- **Traduction MCD en MLD**

Règle 1 : Toute entité est représentée par une relation. Chaque attribut de l'entité devient un attribut de la relation. L'identifiant est conservé en tant que clé de la relation.

Règle 2 : Toute association qui associe plus de deux entités (ternaire et au-delà) est représentée par une relation.

Règle 3 : Toute association binaire dont les cardinalités maximales sont n de chaque côté est une relation (relation dont les attributs sont les attributs clefs des entités qu'elle relie ainsi que les éventuels attributs propres à l'association).

Règle 4 : Une association de type père - fils, cardinalité maximum à n d'un côté et à 1 de l'autre, n'est pas représentée par une relation. On indique les attributs clefs de l'entité père (côté (.,n)) dans le fils (côté (.,1)).

- **Règles de normalisation**

Dépendance fonctionnelle élémentaire :

Une DF élémentaire est une DF de la forme X → A où A est un attribut unique n'appartenant pas à X et où il n'existe pas X' inclus au sens strict dans X (i.e. X' Ì X) tel que X' → A.

Dépendance fonctionnelle directe :

Une DF X → A est une DF directe s'il n'existe aucun attribut B tel que l'on puisse avoir X → B et B → A.

Norme 1FN : Une relation est en 1FN si, et seulement si, tout attribut contient une valeur atomique (non multiple, non composée).

Norme 2FN : Une relation est en 2FN si, et seulement si elle est en 1FN et si toutes les dépendances fonctionnelles entre la clé et les autres attributs sont élémentaires.

Norme 3FN : Une relation est en 3FN si, et seulement si elle est en 2FN et si toutes les dépendances fonctionnelles entre la clé et les autres attributs sont directes.

Norme BCNF : Une relation est en BCNF si, et seulement si elle est en 3FN et si les seules dépendances fonctionnelles élémentaires sont celles dans lesquelles une clé détermine un attribut (et non l'inverse).

IV. Les types et les contraintes en SQL

Voici une liste des principaux types :

- **Type alpha-numérique : char(n fixe),varchar2(n max),long**
- **Type numérique : number(p,s),integer,smallint,float**
- **Type gestion du temps : Date, TimeStamp**

Voici une liste des principales contraintes de colonnes (précédé du mot clé **CONSTRAINT**) :

- **Nullité de colonne : NULL / NOT NULL**
- **Unicité de valeur dans la colonne : UNIQUE**
- **Clé primaire : PRIMARY KEY**
- **Vérification : CHECK**
- **Cle étrangère : REFERENCES**

Attribuer une valeur par défaut : **DEFAULT**

De la mêmemanière que l'on définie des contraintes de colonne, on peut définir des contraintes qui concernent des regroupement de colonne. Aucune différence d'utilisation, si ce n'est qu'il faut préciser les colonnes concernées à la fin.
De manière générale, on donne un nom à chaque contrainte qui explique ce qu'elle fait.

V. Le LDD (définition)

- **Create**

CREATE TABLE <*nom*> (<*définition colonne*> | <*définition contrainte*>,...)
[<*spécification stockage*>]
[<*données provenant d'une requête*>]

- **Drop**

DROP TABLE <nom_table>

- **Alter**

ALTER TABLE <nom_table>
[<add>][<modify>][<drop>]

VI. Le LMD (manipulation)

- **Insert**

```
INSERT INTO <nom_table>
VALUES (<expr>[,<expr>...]); // Attention à l'ordre des colonnes

INSERT INTO <nom_table> (<col1> [, … ])
VALUES (<expr1> [, … ]);
```

- **Update**

```
UPDATE <nom_table>
SET <col>=<expr>[,<col>=<expr>…]
[WHERE <predicat>]
```

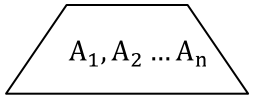
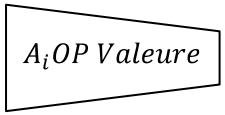
- **Delete**

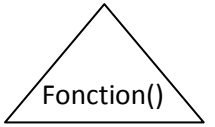
```
DELETE FROM <tab> [WHERE <predicat>]
```

- **Select**

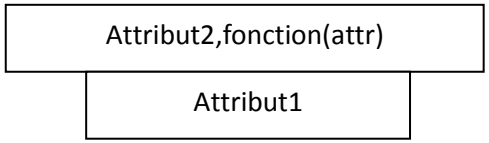
```
SELECT <col1> [, … ] FROM <tab> [WHERE <predicat>]
 //  '*' selectionne toutes les colonnes
```

VII. Les opérations de base sur les selects : SQL et langage algébrique

Description	Syntaxe SQL	Algèbre relationnel	Schéma
Projection	`SELECT DISTINCT c1, c2 FROM ma_table`	$\pi_{C1,C2}\ (ma\_table)$	
Restriction	`SELECT * FROM ma_table WHERE c1 = 600;`	$\sigma_{C1=600}(ma\_table)$	
Comparaison chaînes	`SELECT prenom FROM personne WHERE nom LIKE '_X%'`	----------	
Concaténation	`SELECT first_name || ' ' || last_name FROM …`	----------	
Soustraction	`SUBSTR(chaîne, pos, long)`	----------	
Tri	`SELECT ... FROM ma_table [WHERE ...] ORDER BY COL1[ASC|DESC] [, COL2 [ASC|DESC]]`	$Tri(Relation, Ai\ \downarrow, Aj\ \uparrow)$	$A_i \downarrow, A_j$

Fonction	`SELECT fonction(un_attribut) FROM ma_table;`	$R2 = F_{fonction([attribut])}(R1)$ 	
Renommage attribut	`SELECT attribut1 AS exemple FROM ma_table`	$\boldsymbol{\rho_{(B_1,B_2\ldots)}R(A_1,A_2\ldots)}$	
Renommage relation	`SELECT T.attribut1 FROM ma_table T`	$\boldsymbol{\rho_E(R)}$	
Agrégat	`SELECT <col1>, <col2>, <fonction_agrégat>(<col3>) FROM ... WHERE … GROUP BY <col1>, <col2>;`	$R2 =_{attributs_1} F_{attributs_2,fonction(attr)}(R1)$ ($attributs_2$ ssensemble de $attributs_1$)	**Voir ci-dessous**
Predicat sur un groupe	`SELECT num_client, SUM(montant) FROM commandes GROUP BY num_client HAVING SUM(montant) > 1000;`	----------	
Sous-requêtes renvoyant une valeur	`SELECT * FROM table WHERE col1[,clo2…] = (SELECT col1[,col…] FROM table WHERE <predicat>);`	----------	
Sous-requêtes renvoyant plusieurs valeurs	`SELECT * FROM table WHERE col1[,clo2…] IN (SELECT col1[,col…] FROM table WHERE <predicat>);`	----------	

Agrégat :



Remarques :

- Voici une liste des opérateurs de comparaison pour le 'where' : = , <> , <= , >= , < , >
- Voici une liste des opérateurs logiques pour le 'where' : **AND** , **OR** , **NOT**
- Test sur la nullité d'un attribut : **IS NULL** , **IS NOT NULL**
- Voici une liste des meta-caractères existants en SQL : **_** (un seul caractère) , **%** (0 ou n caractères)
- Voici une liste des opérations sur les chaînes de caractère : **UPPER**(MAJASCULE) , **LOWER**(minuscule), **TO_NUMBER**(Chaîne → nombre), **TO_CHAR**(nb,masque(voir slide 13 BDD 6))
- Conversion d'une date : **TO_DATE**(chaîne,format) , **TO_CHAR**(date,formet) → voir slide 14 BDD 6
- Liste des opérateurs acceptés pour les sous-requêtes renvoyant une valeur : = , <> , <= , >= , < , >
- Liste des opérateurs acceptés pour les sous-requêtes renvoyant plusieurs valeurs : **IN** , **NOT IN** , **ALL** , **ANY** , = , <> , <= , >= , < , >

Remarque 2 : Lorsque l'on utilise une sous-requête, au lieu de faire une comparaison, on peut faire un test sur la liste : **SELECT** T1.ID **FROM** table2 T2 **WHERE EXISTS** | **NOT EXISTS** (**SELECT** T2.ID **FROM** table2 T2 **WHERE** T2.ID = T1.ID)

VIII. Les opérations ensemblistes et jointures : SQL et langage algébrique

1. Union

Langage Algébrique	*SQL*

$R_1 \cup R_2$	`SELECT T1.C1, T1.C2 FROM table1 T1`
	`UNION`
	`SELECT T2.C1, T2.C2 FROM table2 T2`

2. Intersection

Langage Algébrique	*SQL*

$R_1 \sqcap R_2$	`SELECT T1.C1, T1.C2 FROM table1 T1`
	`INTERSECT`
	`SELECT T2.C1, T2.C2 FROM table2 T2`

3. Différence

Langage Algébrique	*SQL*

$R_1 - R_2$	`SELECT T1.C1, T1.C2 FROM table1 T1`
	`MINUS`
	`SELECT T2.C1, T2.C2 FROM table2 T2`

4. Division

Langage Algébrique	*SQL*
$R_1 \div R_2$	`PAS D'EQUIVALENT => ON SE DEBROUILLE`

5. Produit Cartésien

- **Produit cartésien ou jointure**

Langage Algébrique	*SQL*
$R_1 \times R_2$	`SELECT * FROM table1 T1,table2 T2`

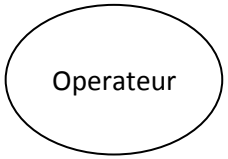
Le pivot est en fait une contrainte qui réduit les croisements possibles.

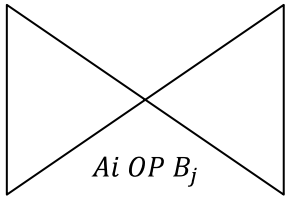
- **Jointure sans pivot**

Langage Algébrique	*SQL*
$R1 \bowtie_{A_i<OP>B_i} R2$	`SELECT * FROM table1 T1 [INNER] JOIN table2 T2 ON T1.Ai = T2.Bi`

Un cas particulier de jointure, est la jointure naturelle : elle se fait sur les attributs de même nom et même type. Le principal avantage est que les colonnes sur lesquelles sont faites les jointures ne sont affichées qu'une seule et unique fois : **`SELECT * FROM tab1 NATURAL JOIN tab2;`**

Pour tous les opérateurs ensemblistes et les jointures, les schema sont respectivement :





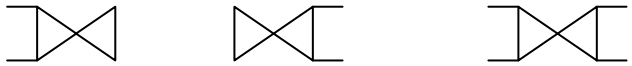
IX. Les joitures externes en SQL

```
SELECT ... FROM
<table gauche>
LEFT | RIGHT | FULL [OUTER] JOIN
<table droite>
[ LEFT | RIGHT | FULL [OUTER] JOIN
<table droite> … ]
ON
<condition de jointure>
WHERE … ;
```

Remarque : A la place des joitures externes, on peut utiliser les pivots.

`SELECT * FROM tab1 LEFT OUTER JOIN tab2 ON tab1.col11 = tab2.col21;`	⇔	`SELECT * FROM tab1,tab2 WHERE tab1.col1 = tab2.col1(+);`
`SELECT * FROM tab1 RIGHT OUTER JOIN tab2 ON tab1.col11 = tab2.col21;`	⇔	`SELECT * FROM tab1,tab2 WHERE tab1.col1(+) = tab2.col1;`

Attention : La clause WHERE est une condition sur une table et la clause ON est une condition de jointure. Les résultats sont complétements différents. Voici les schema respectivement des jointures LEFT, RIGHT et FULL :



X. Les vues, les index, les droits

○ Les vues

```
CREATE VIEW <nom_vue> [(<col1>,<col2>…)] AS
SELECT …
```

Où le select est le résultat de n'importe quelles opérations définies précédement. On peut en plus ajouter des vérifications d'intégrité dans le cas où un utilisateur voudrait ajouter une donnée dans cette table sans passer par le select => il pourrait y avoir des données qui ne correspondent pas aux conditions. Pour ce faire, on ajoute après chaque prédicat le mo clé `WITH CHECK OPTION CONSTRAINT <Nom_Contrainte>`.

```
DROP VIEW <nom_vue>
```

○ Les index

```
CREATE INDEX nom_index ON nom_table
(nom_col [ASC|DESC], nom_col…,);

DROP INDEX nom_ind;
```

○ Les privilèges

```
GRANT <privilège> ON <table> TO
<utilisateur> [with grant option]


REVOKE <privilège> ON <table> FROM
<utilisateur2>
```