
Bases de données

-

Modèle relationnel

-

SQL

Table des matières

I - Modèle relationnel (Base de données).....	4
1. Généralités sur les bases relationnelles.....	4
2. Modèle relationnel (approche théorique).....	5
2.1. Les 12 règles de Codd.....	6
2.2. Cardinalité de relation.....	7
2.3. Principales "Formes Normales".....	8

2.4. L'algèbre relationnelle et la logique du SQL.....	10
2.4.a. Sélection.....	10
2.4.b. Projection.....	10
2.4.c. Produit cartésien.....	11
2.4.d. Jointure.....	12
2.4.e. Union.....	13
2.4.f. Différence (rare).....	13
2.4.g. Intersection (partie commune).....	14

II - SQL..... 15

1. Présentation de SQL.....	15
2. DDL (Data Definition Language) / SQL.....	15
2.1. Suppression & création de base de données:.....	15
2.2. Suppression & Création de tables:.....	16
3. DCL (Data Control Language) / SQL.....	17
3.1. Affectation de privilèges (droits accordés).....	17
4. DQL (Data Query Language) / SQL.....	17
4.1. Interrogations (SELECT).....	17
4.2. Création de nouvelles tables à partir d'une requête:.....	23
4.3. Subselect (sous select).....	24
5. DML (Data Manipulation Language) / SQL.....	25
5.1. Insertions d'enregistrements.....	25
5.2. Modifications d'enregistrements.....	26
5.3. Suppressions d'enregistrements.....	27
6. Vues.....	28

III - Bases de données relationnelles..... 29

1. Principaux SGBDR (RDBMS).....	29
2. Oracle.....	29
3. MySQL.....	30
3.1. Exemples de scripts pour créer une base "MySQL".....	30
4. DB2.....	31
4.1. Exemples de scripts DB2.....	31

IV - Contraintes d'intégrités et procédures stockées..... 33

1. Contraintes d'intégrités.....	33
1.1. contraintes d'intégrités classiques.....	33
1.2. éventuelles contraintes d'intégrités en mode "différé".....	33
2. Transactions (COMMIT , ROLLBACK, ...).....	34
3. Procédures stockées.....	34
3.1. présentation des procédures stockées.....	34
3.2. déclenchement explicite depuis jdbc/java.....	35

3.3. éventuels déclenchements automatiques via "trigger".....	35
---	----

V - Annexe – Modélisation base de données.....	37
---	-----------

1. différences cardinalités/multiplicités.....	37
2. Modélisation d'un schéma relationnel.....	38
2.1. Visualisation d'une structure existante.....	38
2.2. Modélisation d'un nouveau schéma relationnel.....	39

VI - Annexe – Bibliographie, Liens WEB + TP.....	43
---	-----------

1. Bibliographie et liens vers sites "internet".....	43
2. TP.....	43
2.1. Installation des logiciels et de l'environnement de Tp.....	43
2.2. Prise en main de MySqlWorkbench.....	44
2.3. Séries de TPs autour de bo = base école d'équitation.....	44

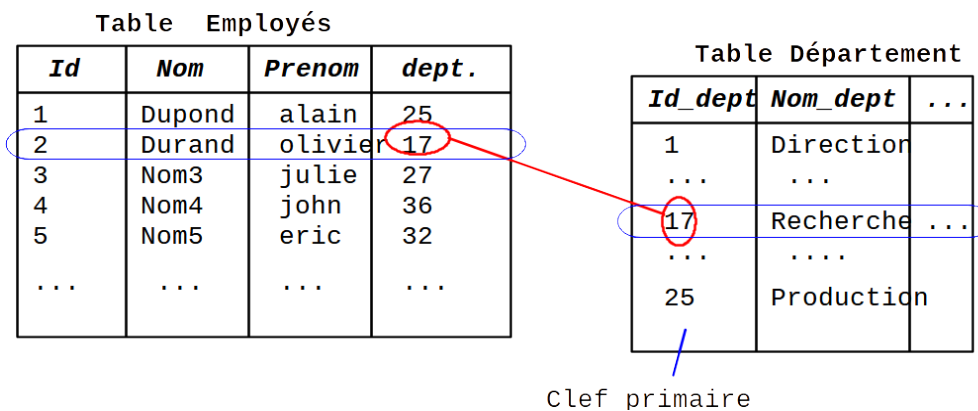
I - Modèle relationnel (Base de données)

1. Généralités sur les bases relationnelles

Une **base de données relationnelle** est essentiellement un **ensemble de tables**.
Chaque table représente une association entre différents attributs.

Les **colonnes** d'une table sont quelquefois appelées des champs (field) . La **structure** d'une table est ainsi définie par la liste de ses colonnes.

Les **lignes** d'une table sont appelées des **enregistrements** (record). Le **contenu** d'une table est de cette façon défini par la liste de ses enregistrements.



Certaines **tables** sont quelquefois **reliées** entre elles par des **relations de jointure** (par exemple: Employés.dept = Département.id_dept).

Sigles :

SGBDR : Système de Gestion de Base de Données Relationnelles

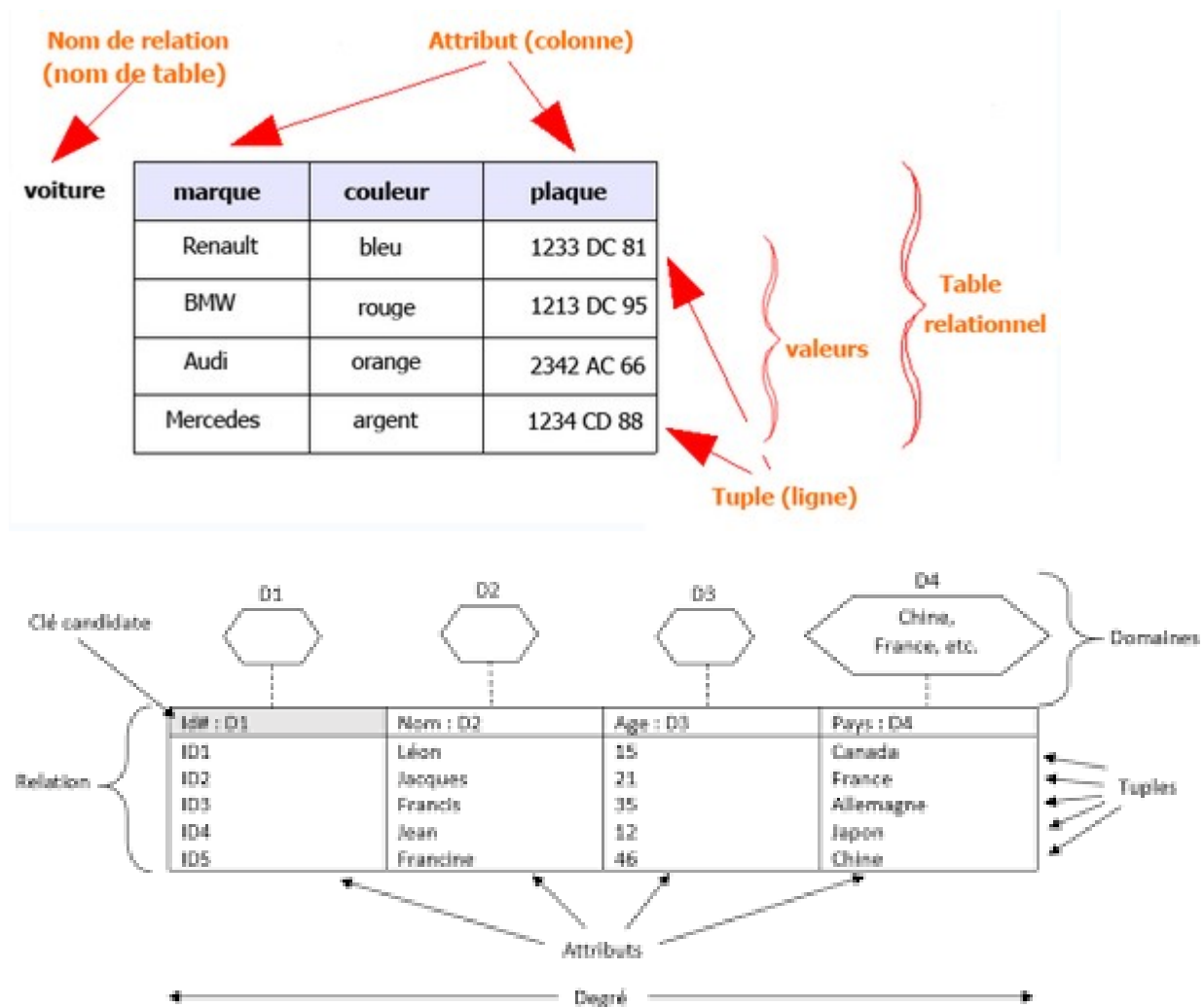
RDBMS : Relational DataBase Management System

2. Modèle relationnel (approche théorique)

Edgar Frank « Ted » Codd (23 août 1923 - 18 avril 2003) est un informaticien [britannique](#). Il est considéré comme l'inventeur du [modèle relationnel](#) des [SGBDR](#).

https://fr.wikipedia.org/wiki/Mod%C3%A8le_relationnel

[https://fr.wikipedia.org/wiki/Forme_normale_\(bases_de_donn%C3%A9es_relationnelles\)](https://fr.wikipedia.org/wiki/Forme_normale_(bases_de_donn%C3%A9es_relationnelles))



Vocabulaire	Signification
Degré	Nombre d'attribut dans une relation
Table Relationnelle	Entête = ensemble d'attributs (nom et domaine/type) , corps = ensemble des uplets (enregistrements/lignes)
Domaine (Type)	Le domaine représente un ensemble fini de valeur possible pour un attribut donné auquel on définit aussi un ensemble d'opérateurs pouvant être appliqués aux valeurs du domaine

2.1. Les 12 règles de Codd

Règles	Descriptions
R1 : Unicité de représentation	Toute l'information dans la base de données est représentée d'une et une seule manière, à savoir par des valeurs dans des champs de colonnes de tables
R2 : Garantie d'accès	Toutes les données doivent être accessibles sans ambiguïté. Cette règle est essentiellement un ajustement de la condition fondamentale pour des clefs primaires. Elle indique que chaque valeur scalaire individuelle dans la base de données doit être logiquement accessible en indiquant le nom de la table contenant, le nom de la colonne contenant et la valeur principale primaire de la rangée contenant
R3 : Traitement des valeurs nulles	Le système de gestion de bases de données doit permettre à chaque champ de demeurer nul (ou vide). Spécifiquement, il doit soutenir une représentation "d'information manquante et d'information inapplicable" qui est systématique, distincte de toutes les valeurs régulières (par exemple, "distincte de zéro ou tous autres nombres," dans le cas des valeurs numériques), et ce indépendamment du type de données. Cela implique également que de telles représentations doivent être gérées par le système de gestion de bases de données d'une manière systématique
R4 : Catalogue lui-même relationnel	Le système doit supporter un catalogue en ligne, intégré, relationnel, accessible aux utilisateurs autorisés au moyen de leur langage d'interrogation régulier. Les utilisateurs doivent donc pouvoir accéder à la structure de la base de données (catalogue) employant le même langage d'interrogation qu'ils emploient pour accéder aux données de la base de données
R5 : Sous-langage de données	Le système doit soutenir au moins un langage relationnel qui: <ol style="list-style-type: none"> 1. a une syntaxe linéaire 2. peut être employé interactivement et dans des programmes d'application, 3. supporte des opérations de définition d'informations supplémentaires (incluant des définitions de vues), de manipulation de données (mise à jour aussi bien que la récupération), de contraintes de sécurité et d'intégrité, et des opérations de gestion de transaction (commencer, valider et annuler une transaction)
R6 : Mise à jour des vues	Toutes les vues pouvant théoriquement être mises à jour doivent pouvoir l'être par le système
R7 : Insertion, mise à jour et effacement de haut niveau	Le système doit supporter les opérations par lot d'insertion, de mise à jour et de suppression. Ceci signifie que des données peuvent être extraites d'une base de données relationnelle dans des ensembles constitués par des données issues de plusieurs uplets et/ou de multiples table. Cette règle explique que

	l'insertion, la mise à jour, et les opérations d'effacement devraient être supportées aussi bien pour des lots d'uplets issues de plusieurs tables que juste pour un uplet unique issu d'une table unique
R8 : Indépendance physique	Les modifications au niveau physique (comment les données sont stockées, si dans les rangées ou les listes liées, etc.) ne nécessitent pas un changement d'une application basée sur les structures.
R9 : Indépendance logique	Les changements au niveau logique (tables, colonnes, rangées, etc) ne doivent pas exiger un changement dans l'application basée sur les structures. L'indépendance de données logiques est plus difficile à atteindre que l'indépendance de données physiques
R10 : Indépendance d'intégrité	Des contraintes d'intégrité doivent être indiquées séparément des programmes d'application et être stockées dans le catalogue. Il doit être possible de changer de telles contraintes au fur et à mesure sans affecter inutilement les applications existantes
R11 : Indépendance de distribution	La distribution des parties de la base de données à diverses localisations doit être invisible aux utilisateurs de la base de données. Les applications existantes doivent continuer à fonctionner avec succès : <ol style="list-style-type: none"> 1. quand une version distribuée du système de gestion de bases de données est d'abord présentée ; et 2. quand des données existantes sont redistribués dans le système.
R12 : non-subversion	Si le système fournit une interface avec langage de bas niveau , cette interface ne doit pas permettre de contourner le système (par exemple pour ajouter une contrainte relationnelle de sécurité ou d'intégrité) : ces contraintes doivent être exprimées dans le langage relationnel de haut niveau.

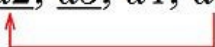
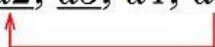

2.2. Cardinalité de relation

Cardinalités	Descriptions (d'une relation entre tables A et B)
Relation 1:1	un uplet de la table A se rapporte seulement à un uplet de la table B
Relation 1:N	n uplet de la table A peut se rapporter à plusieurs uplets de la table B, et un uplet de la table B seulement à un uplet de la table A.
Relation N:N	un uplet de la table A peut se rapporter à plusieurs uplets de la table B et un uplet de la table B peut se rapporter à plusieurs uplets de la table A. Une relation N:N peut donc être décomposées en deux relations 1:N

2.3. Principales "Formes Normales"

NB : Les formes normales sont à considérées comme des "conseils", des "bonnes pratiques générales" qu'il faut néanmoins contrebalancer par des aspects pragmatiques (simplicité, efficacité) quelquefois antagonistes.

NB : les formes normales sont imbriquées : $2FN = 1FN + \dots$, $3FN = 2FN + \dots$

Formes Normales	Principes
1FN : première forme normale <p><i>pas de ça :</i></p> <p>$R(\underline{a1}, \underline{a2}, \underline{a3}, a4, a5, a6)$</p> 	<p>Une relation (ayant par définition un identifiant) est en première forme normale si tous les attributs possèdent tous une valeur sémantiquement atomique.</p> <p>Une autre définition serait : un attribut est dit « atomique » si aucune subdivision de la donnée initiale n'apporte une information supplémentaire ou complémentaire</p> <p>Contre exemple : une adresse complète ou bien un numéro de sécurité sociale ne sont pas des infos atomiques (elles peuvent être décomposées)</p>
2FN : deuxième forme normale <p><i>pas de ça :</i></p> <p>$R(\underline{a1}, \underline{a2}, \underline{a3}, a4, a5, a6)$</p> 	<p>Les attributs d'une relation sont divisés en deux groupes : le premier groupe est composé de l'identifiant (un ou plusieurs attributs). Le deuxième groupe est composé des autres attributs (éventuellement vide). La deuxième forme normale stipule que tout attribut du deuxième groupe ne peut pas dépendre d'un sous-ensemble (strict) d'attribut(s) du premier groupe. En d'autres termes : « Un attribut non identifiant ne dépend pas d'une partie de l'identifiant mais de tout l'identifiant. »</p> <p>Corollaire : une relation ayant un identifiant formé d'un seul attribut est donc en deuxième forme normale.</p> <p><i>Le non-respect de la 2FN entraîne une redondance des données qui encombrement alors inutilement la mémoire et l'espace disque.</i></p>
3FN : troisième forme normale <p><i>pas de ça :</i></p> <p>$R(\underline{a1}, \underline{a2}, \underline{a3}, a4, a5, a6)$</p> 	<p>La troisième forme normale stipule que tout attribut du deuxième groupe ne peut pas dépendre d'un sous-ensemble (strict et excluant l'attribut considéré) d'autres attribut(s) du second groupe. En d'autres termes : « Un attribut non identifiant ne dépend pas d'un ou plusieurs attributs ne participant pas à l'identifiant ». Dit encore autrement : « Tous les attributs non identifiants doivent dépendre <i>directement</i> de l'identifiant, au sens où il n'y a aucun attribut non identifiant dépendant de l'identifiant par dépendances <i>transitives</i> par l'intermédiaire d'autres attributs non identifiants ».</p>
FNBC : Forme normale de Boyce-Codd (variante améliorée de 3FN)	<p>tous les attributs non identifiants (hormis les <u>clefs candidates</u> qui sont neutres et ne doivent pas être considérées) ne sont pas source de dépendance fonctionnelle (DF) pour une partie de l'identifiant.</p>

I - Modèle relationnel (Base de données)

	Le non-respect de la 2FN, 3FN et la FNBC entraîne de la redondance, une même donnée étant répétée un nombre considérable de fois.
4FN, 5FN,

Exemples de réajustements pour respecter les formes normales :

Sans respect de 1FN		Avec respect 1FN	
		Produit	Fournisseur
Produit	Fournisseur	téléviseur	Video SA
téléviseur	Video SA , Hitec LTD	téléviseur	Hitec LTD

Sans respect de 2FN			Avec respect 2FN	
			<u>Produit (pk)</u>	<u>Fournisseur (pk)</u>
<u>Produit (pk)</u>	<u>Fournisseur (pk)</u>	Adresse_Fournisseur	téléviseur	Video SA
téléviseur	Video SA	1 rue Elle	téléviseur	Hitec LTD
Écran plat	Video SA	1 rue Elle	Écran plat	Video SA
téléviseur	Hitec Ltd	12 avenue A	Et seconde table	
			<u>Fournisseur (pk)</u>	Adresse Fournisseur
			Video SA	1 rue Elle
			Hitec Ltd	12 avenue A

Sans respect de 3FN				Avec respect 3FN		
<u>Fournisseur (pk)</u>	Adresse_Fournisseur	Ville	Pays	<u>Fournisseur (pk)</u>	Adresse_Fournisseur	Ville
Video SA	1 rue Elle	Paris	France	Video SA	1 rue Elle	Paris
Hitec Ltd	12 avenue A	Londre	UK	Hitec Ltd	12 avenue A	Londre
				et nouvelle table		
				<u>Ville (pk)</u>	Pays	
				Paris	France	
				Londre	UK	

Ceci permet d'éviter une redondance des données mais au prix d'une certaine complexité ...

2.4. L'algèbre relationnelle et la logique du SQL

2.4.a. Sélection

SQL - Sélection

CODE	DEPARTEMENT	CODE_RESP
A00	INFORMATIQUE	00010
B01	PRODUCTION	00020
D01	ETUDES	00030
E10	COMPTABILITE	00040
E12	CONTENTIEUX	00045
F10	MICRO	00050

Départements dont le code commence par la lettre E

CODE	DEPARTEMENT	CODE_RESP
E10	COMPTABILITE	00040
E12	CONTENTIEUX	00045

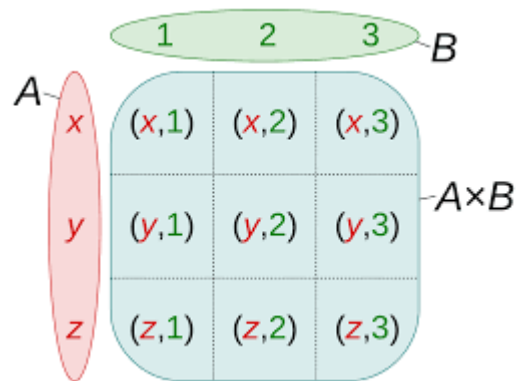
2.4.b. Projection

SQL - Projection

CODE	DEPARTEMENT	CODE_RESP
A00	INFORMATIQUE	00010
B01	PRODUCTION	00020
D01	ETUDES	00030
E10	COMPTABILITE	00040
E12	CONTENTIEUX	00045
F10	MICRO	00050

DEPARTEMENT	CODE
INFORMATIQUE	A00
PRODUCTION	B01
ETUDES	D01
COMPTABILITE	E10
CONTENTIEUX	E12
MICRO	F10

2.4.c. Produit cartésien



Mathématiquement :

Produit cartésien de deux tables :

Exemple :

Etudiants		Epreuves	
n°étudiant	nom	libellé épreuve	coefficient
101	DUPONT	Informatique	2
102	MARTIN	Mathématiques	3
		Gestion financière	5

Examen = PRODUIT (Etudiants, Epreuves)

n°étudiant	nom	libellé épreuve	coefficient
101	DUPONT	Informatique	2
101	DUPONT	Mathématiques	3
101	DUPONT	Gestion financière	5
102	MARTIN	Informatique	2
102	MARTIN	Mathématiques	3
102	MARTIN	Gestion financière	5

2.4.d. Jointure

SQL - Jointure

CODE	LIBELLE	MAT	NOM	SERVICE
A01	INFORMATIQUE	0001	DUVAL	A01
B01	MARKETING	0002	DUBOIS	B01
D01	COMMERCIAL	0003	DURAND	A01

NOM	LIBELLE
DUVAL	INFORMATIQUE
DUBOIS	MARKETING
DURAND	INFORMATIQUE

Jointure en tant que sous partie du produit cartésien :

VENTE As V

IdCli	IdPro	Date	Qte
X	P	1/1/98	1
Y	Q	2/1/98	1
Z	P	3/1/98	1

X

PRODUIT As P

IdPro	Désignation	Prix
P	PS	100
Q	Mac	100

VENTE x PRODUIT (V.IdPro=P.IdPro)



Idcli	IdPro	Date	Qte	Désignation	Prix
X	P	1/1/98	1	PS	100
Y	Q	2/1/98	1	Mac	100
Z	P	3/1/98	1	PS	100

2.4.e. Union

SQL - Union

TABLE DES EMPLOYES TRAVAILLANT A PARIS		
MAT	NOM	DATE DE NAISSANCE
0001	DUVAL	19720101
0003	DUBOIS	19750201
0005	DURAND	19780315
TABLE DES EMPLOYES TRAVAILLANT A LONDRES		
MAT	NOM	DATE DE NAISSANCE
0002	WILSON	19760619
0006	FORD	19720820

LISTE DES EMPLOYES		
MAT	NOM	DATE DE NAISSANCE
0001	DUVAL	19720101
0002	WILSON	19760619
0003	DUBOIS	19750201
0005	DURAND	19780315
0006	FORD	19720820

2.4.f. Différence (rare)

SQL - Différence

CODE	LIBELLE	MAT	NOM	SERVICE
A01	INFORMATIQUE	0001	DUVAL	A01
B01	MARKETING	0002	DUBOIS	B01
D01	COMMERCIAL	0003	DURAND	A01

CODE	LIBELLE
D01	COMMERCIAL

2.4.g. Intersection (partie commune)

SQL - Intersection

SERVICE	LIBELLE	MAT	NOM	SERVICE
A01	INFORMATIQUE	0001	DUVAL	A01
B01	MARKETING	0002	DUBOIS	B01
D01	COMMERCIAL	0003	DURAND	A01

SERVICE
A01
B01

II - SQL

1. Présentation de SQL

Le langage **SQL** (Structured Query Language) est utilisé pour interroger, mettre à jour et gérer des bases de données relationnelles.

SQL n'est pas un véritable langage de programmation (procédural ou événementiel). Il n'a pas été créé pour écrire entièrement une application; Ce n'est qu'un langage de définition et de manipulation de données.

SQL est un langage ensembliste : Il manipule des ensembles d'enregistrements.

Attention : Il existe plusieurs "dialectes" SQL: ANSI-SQL-89 , ANSI-SQL-92 , ACCESS-SQL ,

Par exemple , le fait de pouvoir utiliser des noms de champ contenant des espaces est une fonctionnalité spécifique à certaines bases de données . (Il faut pour cela les encadrer par des crochets avec ACCESS_SQL (ex: SELECT [Year Published], Title FROM Titles) et utiliser des quotes inversées avec MySQL (ex: SELECT `Year Published` , Title FROM Titles))

Les instructions ANSI-SQL peuvent être divisées en plusieurs catégories:

DQL (Data Query Language)	SELECT ...FROM ... WHERE ...GROUP BY ...HAVING ... ORDER BY
DML (Data Manipulation Language)	INSERT , UPDATE, DELETE
TPL (Transaction Processing Language)	BEGIN TRANSACTION , COMMIT,ROLLBACK
DDL (Data Definition Language)	CREATE TABLE, CREATE INDEX, DROP TABLE,...
CCL (Cursor Control Language)	DECLARE CURSOR, FETCH INTO,...
DCL (Data Control Language)	GRANT , REVOKE (gestion des permissions d'accès)

2. DDL (Data Definition Langage) / SQL

2.1. Suppression & création de base de données:

DROP DATABASE *IF EXISTS* xxxdb;

CREATE DATABASE xxxdb;

USE xxxdb;

NB: cette syntaxe valable pour MySql doit peut être être adaptée à d'autres SGBDR .

2.2. Suppression & Création de tables:

Exemples:

```
DROP TABLE IF EXISTS T_Annonce;
CREATE TABLE T_Annonce (
  id INTEGER NOT NULL,
  texte VARCHAR ( 512 ),
  codePostal VARCHAR ( 12 ),
  dateParution DATE format "yyyy-mm-dd",
  onLine SMALLINT,
  rubriqueId INTEGER NOT NULL,
  auteurUserName VARCHAR ( 30 ) NOT NULL,
  CONSTRAINT PK_T_Annonce PRIMARY KEY (rubriqueId, id),
);
```

```
DROP TABLE IF EXISTS T_Rubrique;
CREATE TABLE T_Rubrique (
  id INTEGER NOT NULL PRIMARY KEY,
  label VARCHAR ( 50 ) NOT NULL,
);
```

```
CREATE INDEX TC_I_Annonce1 ON T_Annonce (rubriqueId);
```

```
ALTER TABLE T_Annonce
ADD CONSTRAINT FK_T_Annonce3
FOREIGN KEY (rubriqueId) REFERENCES T_Rubrique (id) ;
```


3. DCL (Data Control Language) / SQL

3.1. Affectation de privilèges (droits accordés)

```
# GRANT ALL PRIVILEGES
GRANT SELECT,INSERT,UPDATE,DELETE
  ON devisedb.*
  TO mydbuser@%'
  IDENTIFIED BY 'mypwd';
FLUSH PRIVILEGES;
```

4. DQL (Data Query Language) / SQL

4.1. Interrogations (SELECT)

```
SELECT fieldlist      FROM tablenames      [WHERE searchcondition]
[GROUP BY fieldlist  [HAVING searchconditions] ] [ORDER BY fieldlist ]
```

Exemples:

1. **SELECT** LastName, FirstName **FROM** Employees **WHERE** Salary > 21000
2. **SELECT** Titles.Title, Dept, Author **FROM** Titles, Authors **WHERE** Titles.AU_ID = Authors.AU_ID
3. **SELECT** Publishers.* **FROM** Publishers
4. **SELECT** Year_Published **AS** Year **FROM** Titles
5. **SELECT** Count(*), Avg(Salary) , Max(Salary) **FROM** Employees
6. **SELECT** Last_Name, First_Name, City **FROM** Employees **WHERE** City **In** ('Interlaken', 'New York', 'Frankfurt')
7. **SELECT** **DISTINCT** Last_Name **FROM** Employees **WHERE** Last_Name = 'Smith';
8. **SELECT** Last_Name, Salary **FROM** Employees **WHERE** Salary **Between** 20000 **And** 30000
9. **SELECT** table1.champA,, Table2.champA, Table2.champB **FROM** Table1, Table2
10. **WHERE** Table1.champA = Table2.champA
11. **SELECT** Product_Name, Sum(Units_in_Stock)
12. **FROM** Products **GROUP BY** Product_Name
13. **SELECT** Department, Count(Department) **FROM** Employees **GROUP BY** Department **HAVING** Count(Department) > 100
14. **SELECT** Last_Name, First_Name **FROM** Employees **ORDER BY** Last_Name

Clause SELECT :

SELECT spécifie quelles sont les colonnes à récupérer . Cette instruction est généralement accompagnée d'une clause FROM indiquant les tables qui contiennent ces colonnes et d'une clause WHERE qui précise quels sont les enregistrements à récupérer.

Notes:

- SELECT est habituellement le premier mot d'une instruction SQL.

Les noms des champs/colonnes doivent être séparés par des virgules et doivent être mentionnés dans le même ordre que le résultat souhaité.

- Si le nom d'un champ apparaît dans plus d'une table listée(s) dans la clause FROM, alors ce nom de champ doit être préfixé par le nom de la table et l'opérateur de portée . (point). Dans l'exemple suivant, le champ AU_ID est à la fois dans la table Authors et Titles et l'instruction SQL sélectionne le champ Title de la table Titles et le champ Author de la table Authors.

```
SELECT Titles.Title, Dept, Author FROM Titles, Authors
WHERE Titles.AU_ID = Authors.AU_ID
```

- On peut utiliser un astérisque (*) pour sélectionner tous les champs d'une table. L'exemple suivant sélectionne tous les champs de la table Publishers:

```
SELECT Publishers.* FROM Publishers
```

- Il est en outre possible d'utiliser le mot clé AS pour créer un alias pour un nom de champ. L'exemple suivant utilise l'alias Year.

```
SELECT Year_Published AS Year FROM Titles
```

<i>SQL statement</i>	<i>Description</i>
SELECT Last Name, First Name FROM Employees	Selects the Last_Name and First_Name fields of all records in the Employees table.
SELECT Employees.* FROM Employees	Selects all fields from the Employees table.
SELECT Orders.Order_ID, Product_ID, Unit_Price FROM Orders, Order_Details	Because the Order_ID field appears in both the Orders and Order_Details tables, this statement specifies that Order_ID be retrieved from the Orders table. Product_ID and Price appear only in the Order_Details table, so the name of the table doesn't have to be specified.
SELECT Count ("Postal_Code") AS Tally FROM Customer	Counts the number of records that have an entry in the Postal_Code field and places the title Tally at the top of the column.
SELECT Last_Name, Salary * 1.1 AS Proposed FROM Employees	Shows what the salary would be if each employee received a 10 percent raise. It does not change the original salary amounts.
SELECT Last_Name AS Name, Salary FROM Employees	Places the title Name at the top of the Last_Name column. The title Salary appears at the top of the Salary column.
SELECT Count (*), Avg (Salary) , Max (Salary) FROM Employees	Shows the number of employees and the average and maximum salaries.
SELECT Last_Name, 'has a salary of', Salary FROM Employees	For each record, shows the Last_Name and Salary in the first and last fields. It displays "has a salary of" in the middle field of each record.

Prédicats ALL, DISTINCT, DISTINCTROW :

Ces prédicats (optionnels) peuvent être utilisés dans les clauses **SELECT** ou **SELECT...INTO**. Lorsqu'ils sont utilisés, les prédicats **DISTINCT** et **DISTINCTROW** sont appliqués au résultat de la requête après toutes les autres clauses de l'instruction SQL.

Le prédicat **ALL** indique que le résultat doit comporter toutes les lignes respectant les conditions indiquée dans la clause **WHERE**, même s'il y a des lignes identiques.

Le prédicat **DISTINCT** doit être mentionné pour ne récupérer qu'un seul élément d'un ensemble de lignes (enregistrements) identiques. Exemple:

```
SELECT DISTINCT Last_Name FROM Employees
WHERE Last_Name = 'Smith';
```

Clause FROM :

FROM spécifie les tables ou requêtes qui contiennent les champs inclus dans l'instruction **SELECT**.

Notes:

- La clause **FROM** est absolument nécessaire et doit suivre **SELECT**.
- L'ordre des noms des tables n'est pas important.

Clause WHERE:

La clause **WHERE** est à utiliser pour déterminer quels seront les enregistrements qui apparaîtront dans le résultat de l'instruction **SELECT**.

Les enregistrements sont sélectionnés suivant la liste de conditions de la clause **WHERE**.

On peut par exemple sélectionner tous les employés du département des Ventes (**WHERE Dept = 'Sales'**) ou tous les clients dont l'âge est compris entre 18 et 30 ans (**WHERE Age BETWEEN 18 And 30**).

Notes:

- La clause **WHERE** est optionnelle, mais elle doit suivre **FROM** lorsqu'elle est mentionnée. Si n'y a pas de clause **WHERE**, tous les enregistrements sont sélectionnés.
- Si la requête comporte plus d'une table (clause **FROM**), et s'il n'y a pas de clause **WHERE**, le serveur retourne alors un "produit cartésien" de toutes les tables:

<u>Table1:</u>	<u>Table2:</u>	<u>Table1 x Table2:</u>
A B	1 2	A B 1 2
C D	3 4	A B 3 4
		C D 1 2
		C D 3 4

- Par exemple, l'instruction SQL suivante sélectionne tous les livres qui ont été publiés après 1991:

```
SELECT Year_Published , Title FROM Titles WHERE Year_Published > 1991
```

<i>SQL statement</i>	<i>Description</i>
SELECT Last_Name, First_Name FROM Employees WHERE Last_Name = 'King';	Selects the Last_Name and First_Name fields of each record in which the last name is King.
SELECT Last_Name, First_Name FROM Employees WHERE Last_Name Like 'S%';	Selects the Last_Name and First_Name fields for employees whose last names begin with the letter S.
SELECT Last_Name, Salary FROM Employees WHERE Salary Between 20000 And 30000;	Selects employees whose salaries are between \$20,000 and \$30,000, inclusive.
SELECT Last_Name, Salary FROM Employees WHERE Last_Name Between 'Lon' And 'Tol';	Selects employees whose last names fall in alphabetical order between Lon and Tol, inclusive. It doesn't retrieve Tolstoy because Tolstoy follows Tol and therefore is outside the specified range.
SELECT Last_Name, First_Name, City FROM Employees WHERE City In ('Interlaken', 'New York', 'Frankfurt');	Selects employees who live in Interlaken, New York, or Frankfurt.
SELECT Title, Year_Published, Author FROM Titles, Authors WHERE Titles.AU_ID = Authors.AU_ID AND Title LIKE 'A%'	Selects title, year, and author for all titles that start with the letter "A"

Compléments sur les jointures:

*Le type le plus fréquent de jointure est l'équi-jointure (INNER JOIN) .
On peut créer une telle jointure on peut utiliser l'ordre SQL suivant:*

```
SELECT table1.champA, ....., Table2.champA,Table2.champB
FROM Table1,Table2
WHERE Table1.champA = Table2.champA
```

La syntaxe SQL-92 amenant au même résultat est la suivante:

```
SELECT table1.champA, ....., Table2.champA,Table2.champB
FROM Table1 INNER JOIN Table2
ON Table1.champA = Table2.champA
```

Clause GROUP BY :

La clause GROUP BY combine en un simple enregistrement tous les enregistrements qui ont des valeurs identiques pour une liste précisée de champs. Une valeur "résumé" est créée pour chaque enregistrement résultant d'une combinaison si une fonction statistique telle que **Sum** ou **Count** est spécifiée dans la partie SELECT.

Si l'instruction SQL comporte une clause WHERE , les enregistrements sont alors regroupés après que leur soit appliqué le filtrage résultant des conditions de la clause WHERE .

Notes:

- La clause GROUP BY est optionnelle mais si présente, elle doit suivre les clauses FROM et WHERE.
- Aucune valeur "résumé" se sera calculée si il n'y a pas de fonction statistique dans la partie SELECT.
- Les valeurs "Null" des champs figurant dans la clause GROUP BY sont regroupées et ne sont pas ignorées. Néanmoins, les valeurs "Null" ne sont pas évaluées dans les fonctions statistiques.
- Il faut utiliser la clause WHERE pour exclure les lignes que l'on ne veut pas regrouper.
- Un champ figurant dans la clause GROUP BY peut faire référence à n'importe quel champ appartenant à l'une des tables figurant dans la clause FROM , même si ce champ n'est pas présent dans la liste des champs de la partie SELECT; Il faut cependant fournir une partie SELECT comportant des fonctions statistiques.
- Tous les champs indiqués dans l'instruction SQL doivent soit être présents dans la clause GROUP BY , soit être utilisés par une fonction statistique de la partie SELECT.

exemple: **SELECT** Product_Name, **Sum**(Units_in_Stock)
 FROM Products **GROUP BY** Product_Name

Clause HAVING :

La clause HAVING sert à spécifier quels sont les enregistrements combinés à afficher.

Notes:

- La clause HAVING est optionnelle mais doit suivre la clause GROUP BY si elle est présente.
- HAVING est semblable à WHERE. La clause WHERE détermine quels sont les enregistrements à sélectionner. Une fois que les enregistrements ont été regroupés par la clause GROUP BY, la clause HAVING détermine quels sont les enregistrements à afficher (récupérer).
- Etant donné que le filtrage spécifié par HAVING intervient après le regroupement, les conditions exprimées dans cette clause peuvent utiliser des valeurs statistiques issues du regroupement (c'est d'ailleurs tout l'intérêt de cette clause).

Exemples:

<i>SQL statement</i>	<i>Description</i>
----------------------	--------------------

SELECT Title, Count (Title) FROM Employees WHERE Department = 'Production' GROUP BY Title HAVING Count (Title) > 50;	Displays the job titles in the Production department assigned to more than 50 employees
SELECT Department, Count (Department) FROM Employees GROUP BY Department HAVING Count (Department) > 100;	Displays departments with more than 100 employees

Clause ORDER BY :

La clause ORDER BY tri les données (enregistrements) à récupérer (à afficher) suivant l'ordre indiqué .

Notes:

- La clause ORDER BY est optionnelle. Si elle est omise, les enregistrements sont récupérés dans l'ordre de création (non triés).
- Par défaut, le tri se fait par ordre croissant (ASCending : (A-Z, 0-9)). Il est néanmoins possible de mentionner le mot clé ASC à la fin de chaque colonne que l'on souhaite récupérer dans l'ordre croissant.

Les 2 exemples suivants trient les noms des employés dans l'ordre croissant:

```
SELECT Last_Name, First_Name FROM Employees ORDER BY Last_Name
```

```
SELECT Last_Name, First_Name FROM Employees
ORDER BY Last_Name ASC
```

- Pour inverser l'ordre du tri (Z-A, 9-0), Il faut ajouter le mot clé **DESC** à la fin de chaque colonne que l'on souhaite récupérer dans l'ordre **décroissant**. L'exemple suivant sélectionne les salaires et les tri dans l'ordre décroissant:

```
SELECT Last_Name, Salary FROM Employees
ORDER BY Salary DESC, Last_Name;
```

- La clause ORDER BY est habituellement la dernière d'une instruction SQL .

Autres exemples:

<i>SQL statement</i>	<i>Description</i>
SELECT Last_Name, First_Name FROM Employees ORDER BY Last_Name DESC	Sorts the records by last name in descending order (Z-A)
SELECT Category_ID, Product_Name, Unit_Price FROM Products ORDER BY Category_ID, Product_Name	Sorts by category ID first, then by product name

Fonctions statistiques du SQL :

<i>Fonctions SQL</i>	<i>Description</i>
Avg (expr)	Moyenne arithmétique (ANSI-SQL)
Count (expr)	Nombre d'élément (ANSI-SQL)
First (expr), Last (expr)	Premier,Dernier élément (ACCESS-SQL)
Min (expr), Max (expr)	Minimum,Maximum (ANSI-SQL)
StDev (expr), StDevP (expr)	Ecart type (ACCESS-SQL)
Sum (expr)	Somme (ANSI-SQL)
Var (expr), VarP (expr)	Variance (ACCESS-SQL)

4.2. Création de nouvelles tables à partir d'une requête:

Instruction SELECT...INTO:

L'instruction SELECT...INTO sert à créer de nouvelles tables à partir d'une requête sur des tables existantes.

Les utilisations classiques de cette instruction sont l'archivage d'enregistrements, la création de copies de sauvegarde ou de copies destinées à l'exportation de données et la création d'un rapport basé sur l'état des données à un instant bien précis. On peut par exemple produire un rapport mensuel sur les ventes régionales en lançant la même instruction SELECT ... INTO chaque mois.

Notes:

- L'instruction SELECT...INTO a la syntaxe suivante:

SELECT *fieldlist* **INTO** *newtablename*

- L'argument *fieldlist* est la liste des champs qui seront copiés dans la nouvelle table. Si *fieldlist* contient plus d'un champ, ceux-ci doivent être séparés par des virgules.
- L'argument *newtablename* est le nom de la table qui sera créée en exécutant la requête. Si le nom que l'on indique pour la nouvelle table est le même que celui d'une table existante, alors la structure et les données de l'ancienne table seront remplacés par celles de la nouvelle.
- On peut avoir besoin de définir un index primaire pour la nouvelle table. Lorsque l'on crée une nouvelle table par SELECT...INTO, les champs de celle-ci héritent simplement du type et de la taille des champs des tables sources, mais aucune autre propriété n'est transférée.
- Pour rajouter des enregistrements dans une table existante, il vaut mieux utiliser l'instruction INSERT INTO.

==> à vérifier, avec MySQL plutôt ==> insert into combiné avec Select

Exemples:

<i>SQL statement</i>	<i>Description</i>
SELECT Employees.* INTO Emp_Backup FROM Employees;	Selects all records in the Employees table and copies them into a new table named Emp Backup.
SELECT Employees.* INTO Trainees FROM Employees WHERE Title = 'Trainee';	Creates a new table that contains only employee records that have the title Trainee.
SELECT Employees.* INTO Employees FROM Employees;	Makes a copy of the Employees table.
SELECT Employees.*, Salary INTO Trainees FROM Employees, Payroll, Employees INNER JOIN Payroll ON Employees.Employee_ID = Payroll.Employee_ID WHERE Title = 'Trainee';	Creates a new table that contains employee and payroll data for all trainees. The Employees and Payroll tables have a one-to-one relationship. The new table contains all of the data from the Employees table, plus the Salary field from the Payroll table.

4.3. Subselect (sous select)

....

5. DML (Data Manipulation Language) / SQL

5.1. Insertions d'enregistrements

```
INSERT INTO tableX [ (Champ1,Champ2, ....) ] VALUES ('ValTexte1',ValNum2, ...)
```

Exemples:

- **INSERT INTO** T_Rubrique (id,label) **VALUES** (2 , "Automobile")
- **INSERT INTO** T_Annonce **VALUES** (1,"livre avec images, facile à lire, 3E","75000","**NOW**",1,1,"PowerUser")

L'instruction INSERT INTO :

L'ordre INSERT INTO sert à ajouter de nouveaux enregistrements dans une table.

Notes:

- Aucun enregistrement ne sera rajouté si le champ correspondant à la clé primaire est vide ou contient une valeur déjà existante dans la table.

Exemples:

<i>SQL statement</i>	<i>Description</i>
INSERT INTO Customers SELECT New_Customers.* FROM New_Customers	Selects all records in the New_Customers table and adds them to the Customers table
INSERT INTO Employees SELECT Trainees.* FROM Trainees WHERE Hire_Date < Now() - 30	Selects all trainees who were hired more than 30 days ago and adds their records to the Employees table

5.2. Modifications d'enregistrements

```
UPDATE TableY SET Champ1 = ValeurNum1 , Champ2 = 'ValeurTexte2'
WHERE Condition
```

Exemples:

- **UPDATE** Orders **SET** Freight = Freight * 1.03 **WHERE** Ship_Country = 'UK'
- **UPDATE** Orders **SET** Order_Amount = Order_Amount * 1.1, Freight = Freight * 1.03 **WHERE** Ship_Country = 'UK'

L'instruction UPDATE :

L'instruction **UPDATE** permet de modifier des enregistrements d'une ou de plusieurs tables .On peut par exemple, réduire le prix de tous les boissons de 10 % ou bien augmenter les frais d'envois des vêtements de 3 %.

Notes:

- **UPDATE** est tout spécialement utile pour changer plusieurs enregistrements (pouvant éventuellement appartenir à plusieurs tables).
- Le mot clé **SET** permet de spécifier les nouvelles valeurs. Dans l'exemple suivant, la nouvelle valeur du champ "Freight" est positionné à la valeur existante augmentée de 3%:

```
UPDATE Orders SET Freight = Freight * 1.03
WHERE Ship_Country = 'UK'
```

- On peut changer plusieurs champs en même temps. L'exemple suivant augmente la valeur du champ [Order Amount] de 10 % et celle du champ "Freight" de 3% :

```
UPDATE Orders
SET Order_Amount = Order_Amount * 1.1, Freight = Freight * 1.03
WHERE Ship_Country = 'UK'
```

- Aucun enregistrement n'est retourné lorsque l'on exécute un ordre UPDATE.

Exemples:

<i>SQL statement</i>	<i>Description</i>
UPDATE Employees SET Reports_To = 5 WHERE Reports_To = 2;	Changes values in the Reports_To field to 5 for all employee records that currently have Reports_To values of 2.
UPDATE Products SET Unit_Price = Unit_Price * 1.1 WHERE Supplier_ID = 8 AND Discontinued = No;	Increases the Unit_Price for all nondiscontinued products from supplier 8 by 10 percent.
UPDATE Products, Suppliers, Suppliers INNER JOIN Products ON Suppliers.Supplier_ID = Products.Supplier_ID SET Unit_Price = Unit_Price * .95 WHERE Company_Name = 'Tokyo Traders' AND Discontinued = No;	Reduces the Unit_Price for all nondiscontinued products supplied by Tokyo Traders by 5 percent. The Products and Suppliers tables have a one-to-one relationship.

5.3. Suppressions d'enregistrements

DELETE FROM Table WHERE Condition
--

Exemple:

- **DELETE FROM** Employees **WHERE** Title = 'Trainee'

L'instruction DELETE :

L'instruction DELETE est à utiliser dans une requête action pour demander l'effacement des enregistrements spécifiés (par la clause WHERE) et appartenant aux tables mentionnées dans la clause FROM.

Notes:

- L'instruction DELETE est à utiliser tout spécialement pour effacer plusieurs enregistrements (qui peuvent éventuellement être dans des tables différentes).
- Pour effacer tous les enregistrements d'une table, il est plus efficace d'effacer la table elle même que d'effectuer une requête d'effacement globale. La structure de la table est cependant perdue si on efface la table. Par contre, l'ordre DELETE ne fait qu'enlever les enregistrements ; La structure de la table ainsi que toutes ses propriétés, (champs,attributs,index) demeurent intacts.
- Pour n'effacer qu'une partie des enregistrements d'une table , il faut inclure une clause WHERE dans l'ordre DELETE. Seuls les enregistrements en accord avec les conditions de la clause WHERE seront effacés.
- Aucun enregistrement n'est retourné lorsque l'on exécute une instruction DELETE.
- On peut utiliser DELETE pour effacer les enregistrements d'une simple table ou bien d'un ensemble de tables reliées entre elles par des relations 1-1. Pour effacer les enregistrements de tables reliées par une relation 1-n , il faut exécuter 2 ordres DELETE.
- L'exécution d'une requête DELETE efface des enregistrements entiers et non pas seulement les champs spécifiés dans la requête. Pour n'effacer qu'un champ bien précis, il faut utiliser un ordre UPDATE qui changera la valeur de ce champ en une valeur "Null".

Remarque importante: Une fois que les enregistrements ont été effacés à la suite d'un ordre DELETE, il n'est plus possible d'annuler l'opération.

Pour savoir quels seront les enregistrements qui seront effacés , on peut examiner le résultat d'une sélection utilisant les mêmes critères que la future requête d'effacement.

Il est également conseiller de faire régulièrement des copies de sauvegarde (pour les tables) . Ainsi , si l'on efface involontairement certains enregistrements , on pourra les récupérer en restaurant la copie de sauvegarde.

Exemples:

<i>SQL statement</i>	<i>Description</i>
DELETE FROM Employees WHERE Title = 'Trainee';	Deletes all records for employees whose title is Trainee. When the FROM clause includes only one table, you don't have to list the table name in the DELETE statement.
DELETE FROM Employees, Payroll, Employees INNER JOIN Payroll ON Employees.Employee_ID = Payroll.Employee_ID WHERE Title = 'Trainee';	Deletes all records for employees whose title is Trainee and who also have a record in the Payroll table. The Employees and Payroll tables have a one-to-one relationship.

6. Vues

CREATE VIEW vue_xxx **AS** select

select * from vue_xxx;

....

III - Bases de données relationnelles

1. Principaux SGBDR (RDBMS)

<i>SGBDR</i>	<i>Caractéristiques</i>
Oracle	Un des SGBDR les plus appréciés du marché (mais un peu cher). Bonnes performances (Unix, Windows, ...). Adapté à de gros volumes.
MySQL (open source + suppléments) et variante MariaDB (open source)	Base "open source" idéale pour des bases de tailles moyennes. Le support des transactions est maintenant correctement assuré au sein des versions récentes . <u>NB</u> : MariaDB s'installe facilement sur un PC Windows
Sybase	
SQLServer (Microsoft)	
DB2 (IBM)	SGBDR sérieux développé par IBM ==> tout un tas de variantes en fonctions des divers O.S. d' IBM
Informix	
PostgreSQL (open source)	SGBDR open source très élaboré (gère de gros volume de données de manière performante)
Access (Microsoft)	Mini base de données / pour petits volumes / bureautique

2. Oracle

....

```

set MYORACLE_HOME=c:\oracle
REM set MYORACLE_BIN=%MYORACLE_HOME%\ora81\bin
set MYORACLE_BIN=%MYORACLE_HOME%\product\10.1.0\Db_1\bin
REM sqlplus USERNAME/password
%MYORACLE_BIN%\sqlplus SYS/admin@ORCL as sysdba
                                < create_user_and_grant_priv.sql
pause

```

%MYORACLE_BIN%\sqlplus mydbuser/mypwd@ORCL < CreateBankDBPart_Oracle.sql

3. MySQL

3.1. Exemples de scripts pour créer une base "MySQL"

1. télécharger le serveur MySQL depuis l'url <http://dev.mysql.com/>
1. installer le logiciel
2. lancer MySQL/bin/WinMySQLAdmin.exe ou MySQL/bin/MySQLInstanceConfig.exe pour paramétrer le serveur et faire en sorte qu'il puisse démarrer comme un service
3. Lancer ensuite les scripts suivants:

set_env_mysql.bat

```
set MYSQL_HOME=C:\Prog\DB\MySQL\MySQL_Server_4.1
set MYSQL_BIN=%MYSQL_HOME%\bin
```

1 lancer_pwd_root.bat

```
call set_env_mysql.bat
REM Fixer le mot de passe de l'administrateur "root" de mysql (ex: "root")
%MYSQL_BIN%\mysql -h localhost -u root -p < update\_root\_user\_with\_root\_pwd.txt
pause
```

[update_root_user_with_root_pwd.txt](#)

```
USE mysql;
UPDATE user SET Password=PASSWORD('root') WHERE user='root';      FLUSH PRIVILEGES;
```

2 lancer_delete_NoPassword.bat

```
call set_env_mysql.bat
%MYSQL_BIN%\mysql -h localhost -u root -p < delete\_no\_password.txt
pause
```

[delete_no_password.txt](#)

```
USE mysql;      DELETE FROM user WHERE User="";      FLUSH PRIVILEGES;
```

3a_create_devisedb.bat

```
call set_env_mysql.bat
%MYSQL_BIN%\mysql -h localhost -u root -p < create\_devisedb.txt
pause
```

[create_devisedb.txt](#)

```
#DROP DATABASE devisedb;
CREATE DATABASE devisedb;
```

```
USE devisedb;
CREATE TABLE DEVISE(MONNAIE VARCHAR(64) NOT NULL PRIMARY KEY,DCHANGE DOUBLE);

INSERT INTO DEVISE VALUES('Dollar',1.0);   INSERT INTO DEVISE VALUES('Euro',1.05);
INSERT INTO DEVISE VALUES('Livre',0.7);   INSERT INTO DEVISE VALUES('Yen',2.1);
show tables;
```

4a lancer grant_priv_devisedb.bat

```
call set_env_mysql.bat
%MYSQL_BIN%\mysql -h localhost -u root -p < grant_priv_on_devisedb.txt
pause
```

grant_priv_on_devisedb.txt

```
# GRANT ALL PRIVILEGES
GRANT SELECT,INSERT,UPDATE,DELETE
ON devisedb.*
TO mydbuser@%'
IDENTIFIED BY 'mypwd';
FLUSH PRIVILEGES;
```

4. DB2

4.1. Exemples de scripts DB2

```
set DB2_HOME=c:\prog\IBM\SQLLIB
set DB2_BIN=%DB2_HOME%\bin

%DB2_BIN%\db2cmd _createDeviseDB.bat
```

```
set DB2_ADM_USER=db2admin
set DB2_ADM_PASSWD=passwdDB2
set DB2_USER=JAVAUSER
set DB2_PASSWD=Java02
REM db2admin/passwdDB2 et JAVAUSER/Java02 sont 2 comptes de l' O.S. local
set DB2INSTANCE=DB2
set DATABASE_NAME=DeviseDB
set LOG=createDeviseDB.log

echo "Attaching to DB2..."
db2 -z%LOG% ATTACH TO %DB2INSTANCE% user %DB2_ADM_USER% using
%DB2_ADM_PASSWD%
echo "Dropping the old database..."
db2 -z%LOG% DROP DATABASE %DATABASE_NAME%
echo "Creating the new database..."
db2 -z%LOG% CREATE DATABASE %DATABASE_NAME%
echo "Connecting to the new database..."
db2 -z%LOG% CONNECT TO %DATABASE_NAME% user "%DB2_ADM_USER%" using
"%DB2_ADM_PASSWD%"
```

```
echo "Granting authorities on the new database to the new user..."
db2 -z%LOG% GRANT CONNECT, CREATETAB ON DATABASE TO USER
%DB2_USER%
echo "Disconnecting from the SAMPLE database..."
db2 -z%LOG% DISCONNECT CURRENT
exit

....
db2 -z%LOG% -f devise_db.sql
...
```

drop table DEVISE

create table DEVISE(MONNAIE varchar(64) not null ,DCHANGE decimal(10,2) , constraint
pk_devise primary key (MONNAIE))

INSERT INTO DEVISE VALUES('Dollar',1.0)
INSERT INTO DEVISE VALUES('Euro',1.05)
INSERT INTO DEVISE VALUES('Livre',0.7)
INSERT INTO DEVISE VALUES('Yen',2.1)

IV - Contraintes d'intégrités et procédures stockées

1. Contraintes d'intégrités

1.1. contraintes d'intégrités classiques

Une clef primaire peut éventuellement être indiquée de cette façon:

```
ALTER TABLE T_YYY ADD PRIMARY KEY (ID_YYY);
```

De façon à garantir une certaine cohérence entre les enregistrements de différentes tables, il est souvent utile de préciser qu'une clef étrangère d'un enregistrement d'une table doit absolument référencer un enregistrement valide d'une autre table .

Cette contrainte d'intégrité élémentaire s'exprime via la syntaxe sql suivante:

```
ALTER TABLE T_XXX  
ADD CONSTRAINT XXX_ref_valid_YYY  
FOREIGN KEY (REF_ID_YYY) REFERENCES T_YYY (ID_YYY);
```

avec: REF_ID_YYY = nom de la colonne de type "clef étrangère" (FK) au sein de T_XXX
ID_YYY = clef primaire de T_YYY

1.2. éventuelles contraintes d'intégrités en mode "différé"

Bien qu'utiles pour sauvegarder la cohérence de l'ensemble des données d'une base, les contraintes d'intégrités classiques sont quelquefois peu pratiques car elles imposent un ordre dans les créations et les suppressions des enregistrements:

Un enregistrement XXX devant absolument référencer un enregistrement YYY ne pourra être créé (via insert into) qu'après YYY et inversement YYY ne pourra être supprimé que s'il n'est plus référencé par XXX .

Un éventuel mode "de vérification différée des contraintes d'intégrités" (s'il est supporté par la base de données) peut s'avérer pratique pour ne pas imposer un ordre entre différentes instructions d'une même transaction.

2. Transactions (COMMIT, ROLLBACK, ...)

Une transaction permet de délimiter un ensemble d'instructions SQL qui seront toutes validées ou toutes annulées (mais surtout pas effectuées à moitié).

BEGIN ou **START TRANSACTION;**

...

....

COMMIT;

ou **ROLLBACK;**

NB:

- MySQL est par défaut en mode AutoCommit=true (chaque ordre SQL qui s'est exécuté sans erreur est implicitement suivi d'un commit).
- L'instruction **START TRANSACTION** permet de désactiver temporairement ce mode AutoCommit=true (jusqu'au prochain commit ou rollback).
- Oracle est par défaut en mode AutoCommit=false et l'instruction **COMMIT;** doit être explicitée.

3. Procédures stockées

3.1. présentation des procédures stockées

Une procédure stockée est une séquence d'instructions SQL se présentant comme une procédure ou une fonction d'un langage de programmation.

Une procédure stockée peut ainsi avoir des paramètres d'entrées et peut éventuellement retourner des valeurs.

Identifiée par un nom et stockée / pré-compilée au sein d'un SGBDR, une procédure stockée peut soit :

- être automatiquement activée suite à des événements internes ("trigger" de la base).
- être explicitement déclenchée depuis des programmes clients.

Exemple MySQL:

```
delimiter $$
CREATE PROCEDURE crediter_compte(IN p_num_cpt INT, IN p_montant DOUBLE,
                                IN p_label VARCHAR(255))
BEGIN
  INSERT INTO OPERATION(NUM_CPT,LABEL,MOUVEMENT)
                                VALUES (p_num_cpt, p_label , p_montant);
  UPDATE COMPTE SET solde=solde+p_montant WHERE NUM_CPT=p_num_cpt ;
END;
$$
delimiter ;
```

CALL *crediter_compte*(1,50,"reception paiement yyy");

3.2. déclenchement explicite depuis jdbc/java

```
CallableStatement cstmt = cn.prepareCall( "{call crediter_compte(?,?,?)}" );
cstmt.setInt(1/*numéro du param*/, 1/*valeur*/);
cstmt.setDouble(2/*numéro du param*/,50.0 /*valeur*/);
cstmt.setString(3/*numéro du param*/,"reception paiement yyy" /*valeur*/);
cstmt.executeUpdate();
```

```
CallableStatement cstmt2 = cn.prepareCall( "{call fromNomProc(?)}" );
cstmt2.setString(1/*numéro du param*/,"Power User" /*valeur*/);
rs = cstmt2.executeQuery();
while( rs.next())
{
    System.out.print( rs.getObject(1) + "," );
    System.out.print( rs.getObject(2)+ "," );
}
rs.close();
```

3.3. éventuels déclenchements automatiques via "trigger"

```
CREATE TABLE MYSTATS (NOM_TABLE VARCHAR(255) NOT NULL PRIMARY KEY,
                      NB_HITS INTEGER);
```

```
INSERT INTO MYSTATS VALUES ('CLIENT' , 0 );
INSERT INTO MYSTATS VALUES ('Table2' , 0 );
```

```
CREATE TABLE CLIENT (NUM_CLIENT INTEGER
                     NOT NULL PRIMARY KEY AUTO_INCREMENT,
                     NOM VARCHAR(255),
                     PRENOM VARCHAR(255));
```

```
DELIMITER $$
CREATE TRIGGER test_hits BEFORE INSERT ON CLIENT
FOR EACH ROW BEGIN
    UPDATE MYSTATS SET NB_HITS=NB_HITS+1 WHERE NOM_TABLE='CLIENT';
END;
$$
DELIMITER ;
```

ANNEXES

V - Annexe – Modélisation base de données

1. différences cardinalités/multiplicités

L'une des principales difficultés du mapping objet-relationnel réside dans le besoin impératif de ne pas confondre "cardinalité" (Merise,relationnel) et multiplicité (UML).

Termes idéals:

Avec UML (objets) ==> **Classes/Types , associations et multiplicités.**

Avec Modèle relationnel (MCD , MLD) ==> **Entités , relations et cardinalités.**

Cardinalité:

La **cardinalité** (*exemple: (1,N)*) correspond au **nombre** minimum et maximum **de fois où une entité précise peut participer à une relation** .

Autrement dit , la **cardinalité** indique les **nombre** minimum et maximum **de fois où une entité précise peut être reliée** à une autre (assez souvent d'un autre type) **dans le cadre d'une certaine relation**. [NB: l'entité précise considérée correspond physiquement à un enregistrement d'une table relationnelle et est assimilable à une instance (objet précis)]

Exemple: dans la relation "1 Personne possède zéro , une ou plusieurs Voitures" , une entité de type "Personne" peut être reliée 0 à N fois à une entité de type "Voiture" et on indique donc une cardinalité de (0,N) du côté "Personne" dans les diagrammes relationnels (MCD,MLD).

Multiplicité (UML):

La **multiplicité** (*exemple: "0..1" ou "0..*"*) correspond au **nombre** minimum et maximum **d'élément(s) d'un certain type (d'une certaine "Classe") qui peuvent être conceptuellement associé(s)** à un autre élément (souvent d'un autre type) **dans le cadre d'une certaine association**.

Exemple: dans la l'association "1 Personne possède zéro , une ou plusieurs Voitures" , une seule entité **de type "Personne"** est généralement associée à une certaine Voiture (cas particulier d'une voiture pas encore achetée ==> associée à zéro propriétaire). On indique donc généralement une multiplicité de "1" ou "0..1" du côté "**Classe Personne**" dans les diagrammes de classes UML.

NB: **Classe** = ensemble des éléments ayant la même structure (attributs) et les mêmes comportements (méthodes) , **multiplicité** = expression du **nombre d'éléments dans le sous ensemble des éléments associés/associables à l'autre extrémité de l'association**.

Les notions de cardinalité et de multiplicité sont donc très différentes (quasiment inversées) .

2. Modélisation d'un schéma relationnel

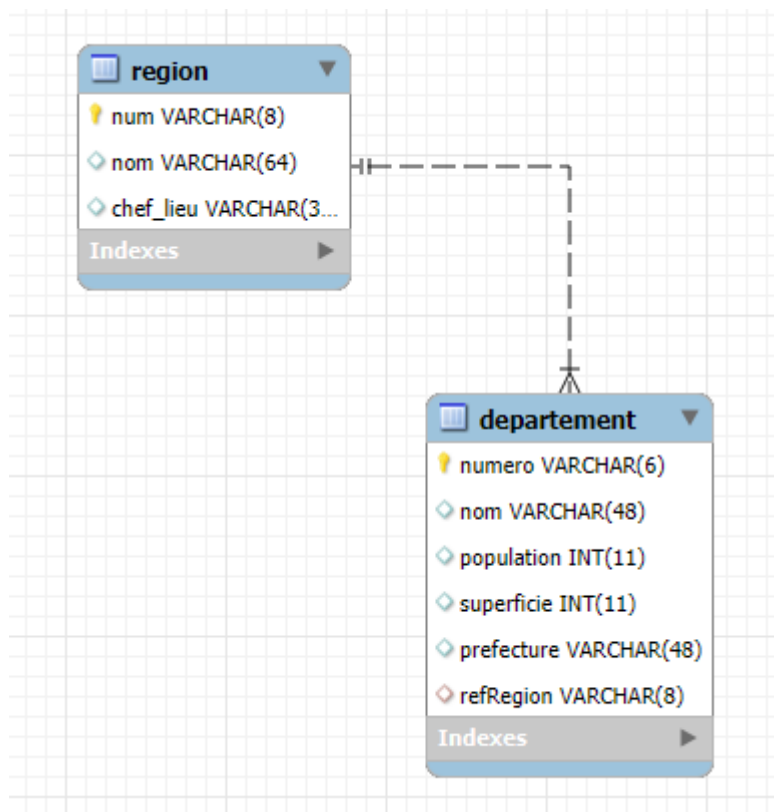
2.1. Visualisation d'une structure existante

Avec **MySQLWorkbench** :

menu "**Database ---> Reverse engineer ..**"

(NB: fonctionne partiellement si manque de contrainte référentielle)

exemple de résultat :

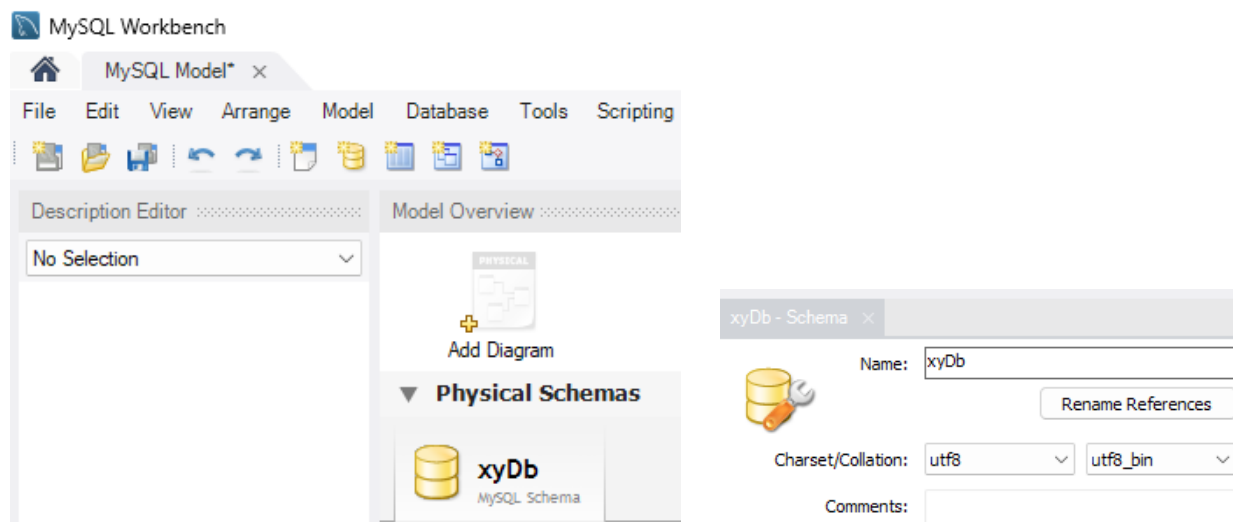


EER Diagram = Enhanced Entity-Relationship Diagram

2.2. Modélisation d'un nouveau schéma relationnel

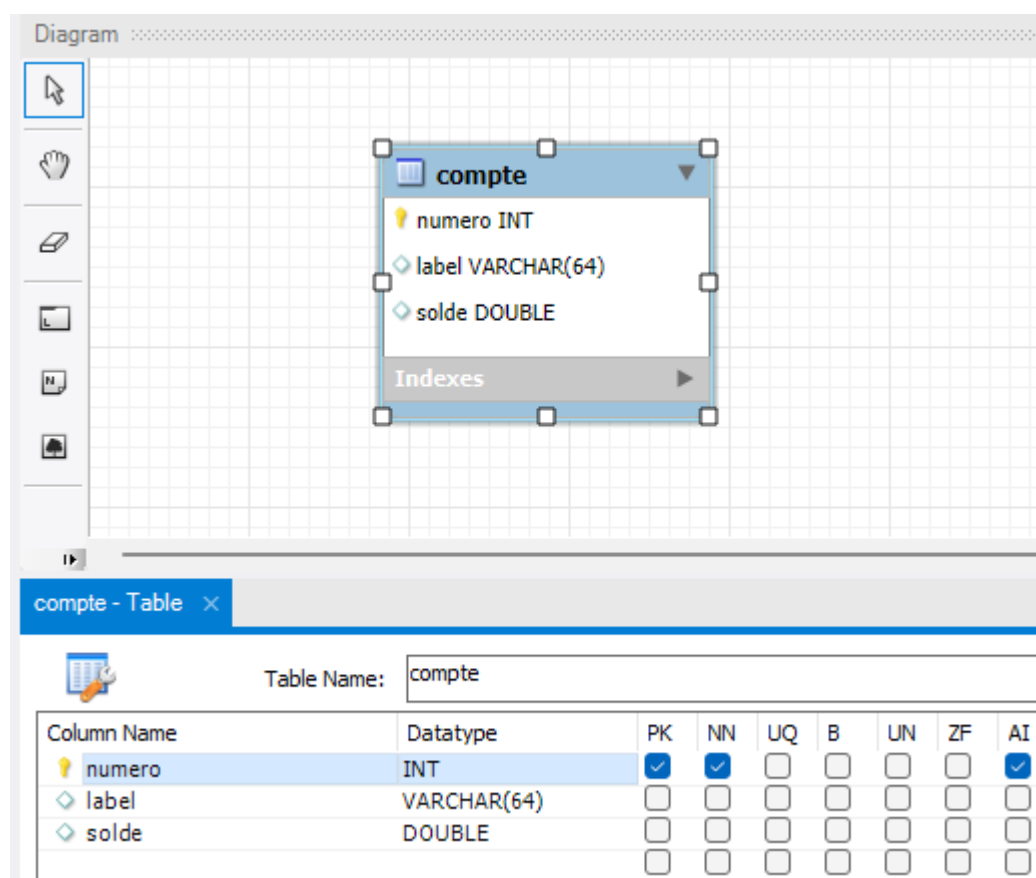
Avec **MySQLWorkbench** : menu "File ---> **New/Open Model ..**"

Commencer par (re)nommer le schéma (ex : xyDB) :



Double click sur "add diagram" pour ouvrir un nouvel onglet "EER Diagram"

Depuis la mini palette , ajouter une nouvelle table (ex : compte)
et paramétrer ses colonnes via l'éditeur de propriétés :

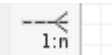
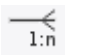


Cocher si besoin "AI" pour "AUTO INCREMENT" .

Edition des relations :

Type de relation	Caractéristiques
Identifying relationship <i>(en trait plein)</i>	When the existence of a row in a child table depends on a row in a parent table (cascade delete in reverse direction) The purpose of an identifying relationship is that the foreign key should NEVER CHANGE , because it is part of the primary key... <u>therefore</u> identifying!!!
non-identifying relationship <i>(en pointillé)</i>	when the primary key attributes of the parent <i>must not</i> become primary key attributes of the child. A non-identifying relationship can be optional or mandatory , which means the foreign key column allows NULL or disallows NULL, respectively

Pour créer une nouvelle relation **1:n** (avec création d'une nouvelle clef étrangère) , il faut :


1. cliquer sur l'icône  ou 
2. cliquer sur la table fille (là ou sera placée la clef étrangère , coté n en général)
3. cliquer sur la table parent (celle qui sera référencée , coté 1 en général)
4. renommer éventuellement la colonne clef étrangère générée

Même mode opératoire pour une nouvelle relation 1-1


Sachant qu'en version "identifying" (en trait plein) , la nouvelle colonne clef étrangère créée est quelquefois automatiquement considéré comme la clef primaire de la table fille

et que ce n'est pas de cas en version "non-identifying" (en pointillé) .

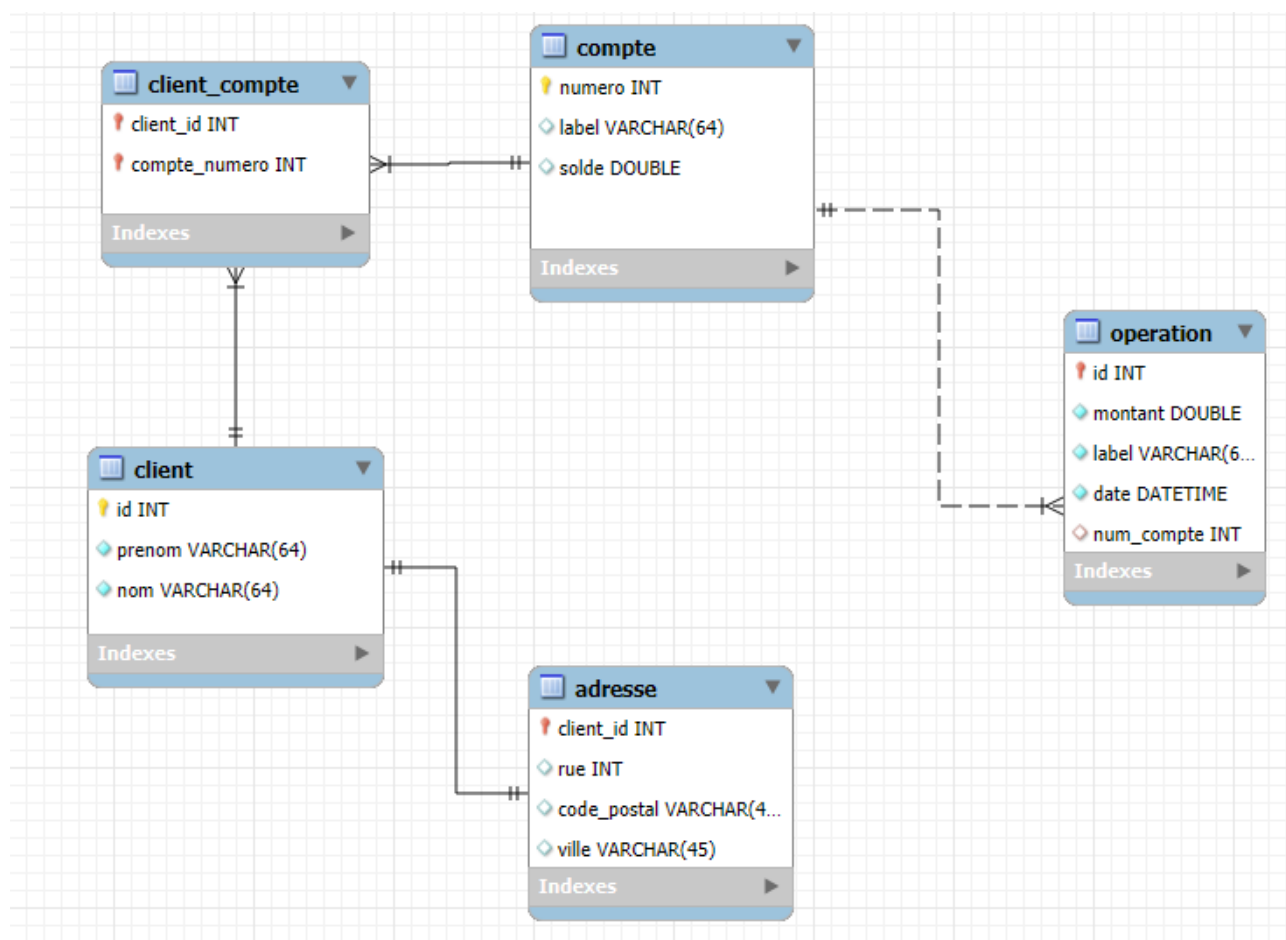
Pour créer une nouvelle relation **1:n** (en utilisant colonne existante de clef étrangère) , il faut :

1. cliquer sur l'icône 
2. cliquer au sein de la table fille sur une des colonnes de la clef étrangère , coté n en général)
3. cliquer dans la popup temporaire sur "**pick referenced column**"
4. cliquer au sein de la table parent (coté 1 en général) sur la colonne à référencer

Pour créer une nouvelle relation **n:m** (avec création d'une nouvelle table de jointure) , il faut :

1. cliquer sur l'icône 
2. cliquer sur la première table
3. cliquer sur la seconde table
4. renommer éventuellement la table intermédiaire générée

Exemple global :

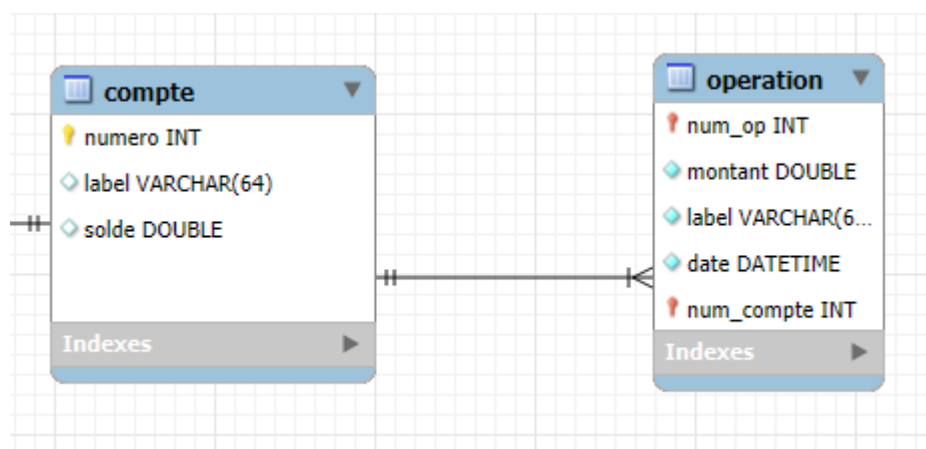


Chaque client comporte une adresse qui lui est propre (1:1)

1 client peut posséder plusieurs comptes bancaires et quelquefois un compte peut appartenir à plusieurs clients (compte joint, compte de copropriété, ...).

1 compte est associé à plusieurs opérations (au sens "traces sur opérations effectuées").

Variante possible où operation comporte une clef primaire composée (num_compte + num_op) :



Via le menu **File / Export / export as png , svg , ...** on peut exporter un diagramme au sein d'un fichier image . On peut également faire ceci via des captures d'écran partielles.

NB : le menu "**Model/relationship notation**" de **MySqlWorkbench** permet éventuellement un affichage au format UML .

Sauvegarde et exportation du schéma :

Au sein de l'outil **MySqlWorkbench** , l'éditeur de diagramme EER modifie indirectement le contenu d'un modèle que l'on peut sauvegarder au sein d'un fichier binaire xyDb.mwb

Via le menu "**Database ---> Forward engineer ..**" on peut générer/initialiser une nouvelle base de donnée (gérée par mysql/mariaDB) en fonction du contenu du modèle/schéma .

Via le menu **File / Export / Forward Engineer SQL Create Script** on peut exporter un modèle/schéma au sein d'un fichier SQL (comportant des instructions "CREATE TABLE" et des contraintes d'intégrité référentielles)

Contenu partiel du xyBankDb.sql généré :

```
...  
  
DROP TABLE IF EXISTS `xyBankDb`.`adresse` ;  
  
CREATE TABLE IF NOT EXISTS `xyBankDb`.`adresse` (  
  `client_id` INT NOT NULL,  
  `rue` INT NULL,  
  `code_postal` VARCHAR(45) NULL,  
  `ville` VARCHAR(45) NULL,  
  PRIMARY KEY (`client_id`),  
  CONSTRAINT `fk_adresse_client1`  
    FOREIGN KEY (`client_id`)  
    REFERENCES `xyBankDb`.`client` (`id`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION)  
ENGINE = InnoDB;  
  
...
```

VI - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

https://learnsql.fr/blog/memento-sql/	Mémento sur principales syntaxes SQL
https://sql.sh	Tuto et cours sur SQL
https://www.mysql.com/fr/	Site officiel de mySQL
https://mariadb.org/	Site officiel de mariaDB (open source)
https://www.postgresql.org/	Site officiel de PostgreSQL
https://www.sqlite.org/ ,	Base de données embarquée pour navigateur
https://www.h2database.com/	Base de données embarquée pour java
https://www.oracle.com/database/	Site officiel d'oracle (database)
https://fr.wikipedia.org/wiki/Structured_Query_Language	SQL wikipedia
https://fr.wikipedia.org/wiki/Mod%C3%A8le_relationnel	Modèle relationnel (wikipédia)
..	

2. TP

2.1. Installation des logiciels et de l'environnement de Tp

Sur un pc windows 64bits, installer les logiciels suivants :

MariaDB (en version 10.6 ou plus) , choisir **root/root** comme compte principal et accès distant possible

MySqlWorkbench

référentiel git pour les tps :

https://github.com/didier-mycontrib/sgbdr_sql

→ à cloner via "git clone" ou bien "code / download zip"

Préparation des bases de données pour les TPs :

- ajuster si besoin les fichiers **tp/init...DB.bat** (en adaptant la valeur de **MYSQL_HOME**)
- lancer les scripts **tp/init...DB.bat**
- visualiser les tables via l'outil HeidiSQL ou bien MySqlWorkbench

2.2. Prise en main de MySqlWorkbench

2.3. Séries de TPs autour de bo = base école d'équitation

Explications/illustrations au sein du fichier **docs/tp-sgbd-sql-cheval.ppt**

exos	objectifs
1	Sélectionner les villes des "lieu_concours" pour la région 'Ile de France'
2	Sélectionner/filtrer les concours ayant eu lieu en juillet 2006 pour des épreuves 'PRO %' avec un résultat compris entre 1 et 3
3	Sélectionner (sans doublon) les no_id des chevaux des concours de type = 'DR' ayant eu lieu entre le 1 ^{er} mars 2006 et le 30 juin 2006 .
4	Nom et date de naissance (au format année mois) des chevaux
5	Quantité de foin totale consommée en 2006 (en partant de la table ration_jour)
6	Afficher les numéros des chevaux et leurs travaux pour la journée du 4 janvier 2005. NB : on pourra extraire la sous partie date (sans heure) pour effectuer le filtrage par date on affichera le travail sous la forme 'Repos' si le temps_travail est NULL et au format 'Travail' concaténé avec son code_type sinon .
7	Afficher (via un regroupement par type_concours) les nombres de concours et le total et la moyenne des gains . Ceci seulement pour les concours de l'année 2005 .
8	Afficher la quantité de foin globalement consommée pour chaque mois
9	Via jointure(s) afficher les noms et libellés des robe et race des chevaux
10	Via jointure(s) afficher les quantités globales consommées en 2006 pour chaque type d'aliment (que l'on affichera via son libellé)
11	Afficher le nombre de chevaux pour chaque race (que l'on affichera via son libellé)
12	Via jointure(s) afficher les lieux , noms des chevaux , type_epreuve et cavaliers des concours effectués le 16 avril 2006 . Le résultat sera trié par nom de cavalier. On affichera "Dressage" pour DR , "Obstacle" pour "OB" et "complet" pour "CP" (c'est à dire le libellé d'un type de travail)
13	Afficher les noms et date de naissance des chevaux qui sont les plus jeunes ou bien les plus vieux
14	On affichera les années et les coûts des aliments qui n'ont pas augmenté d'une année sur l'autre pour un certain type précis d'aliment .
15	Pour l'année 2006, afficher le nom (puis , par union) les quantités de chaque aliment consommées par le cheval dont le complément de nom est 'Gagnant'
16,17,18	Effectuer librement quelques opérations de type "insert" , "update" , "delete" sur la table "cavalier" (sur de nouvelles lignes pour ne pas rendre incohérentes les autres tables)

