Université Hadj Lakhdar Batna

Faculté des sciences économiques, Commerciales et de gestion

Système de Gestion de Base de Données

SOUS DELPHI

L'enseignante :Saidani nabila

Plan du cours

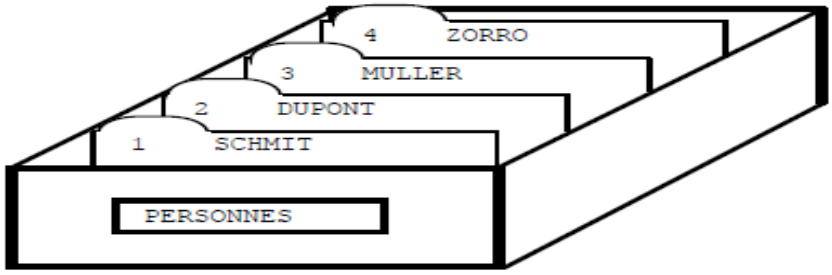
1. Les systèmes de gestion de base de données : Notions de base
2. DELPHI

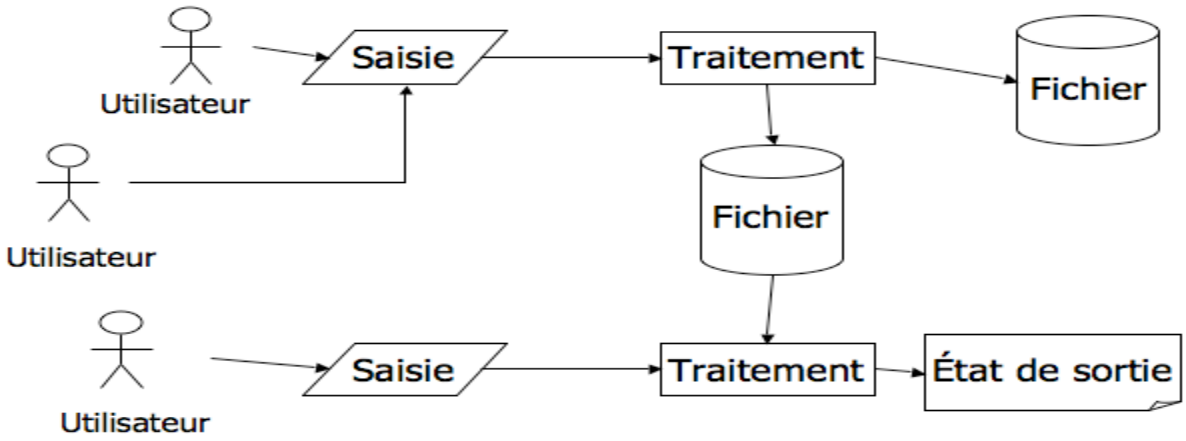
Les systèmes de gestion de base de données

Notions De Base

1.1 Organisation en fichiers

On peut gérer des données à l'aide d'une boîte à fiches. Ici on gère p. ex. les données de différentes personnes.

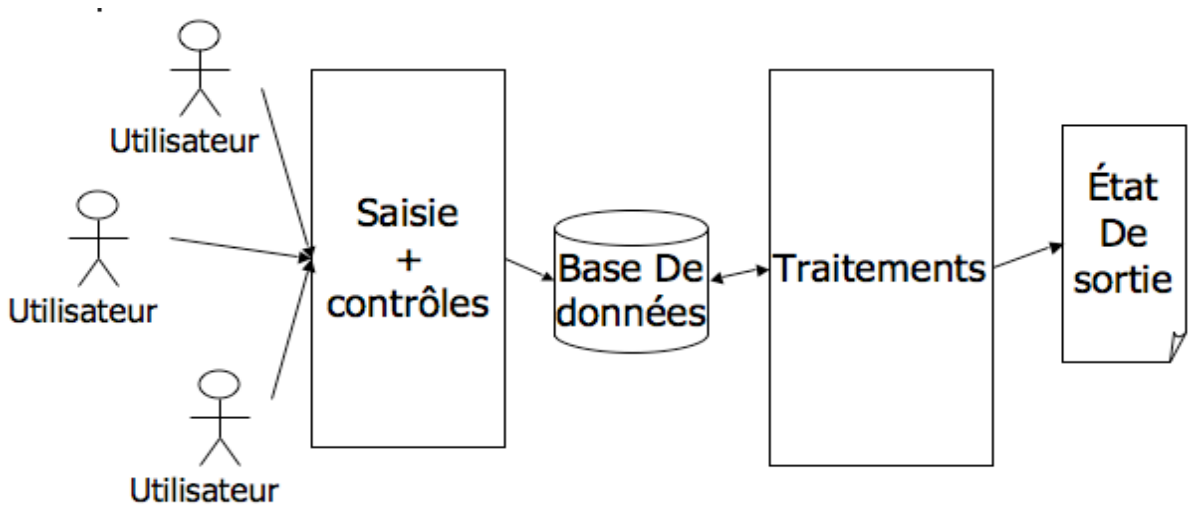




1.2. Limites de l'organisation en fichiers :

- Particularisation de la saisie et des traitements en fonction des fichiers ⇨ un ou plusieurs programmes par fichier.
- Contrôle différé des données ⇨ augmentation des délais et du risque d'erreurs.
- Particularisation des fichiers en fonction des traitements ⇨ grande redondance des données.

1.3. Organisation en base de données :



1.4. Avantages de l'organisation en base de données :

- Uniformisation de la saisie et standardisation des traitements (ex: tous les résultats de consultation sous forme de liste et de tableaux)
- Contrôle immédiat de la validité des données.
- Partage de données entre plusieurs traitements ⇨ limitation de la redondance des Données.

1.5. Définitions

1.5.1 Base De Données

- Base De Données (BDD) : ensemble structuré de données enregistrées avec un minimum de redondance.
- Base De Données (BDD) est un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations avec le minimum de redondance afin d'en faciliter l'exploitation (ajout, mise à jour, recherche de données).

1.5.2 Système de Gestion de Bases de Données (SGBD)

SGBD est un ensemble de logiciels permettant aux utilisateurs de définir, créer, maintenir, contrôler et accéder à la BDD.

1.6 Exemples de grandes applications

Les Bases de Données sont nécessaires à tous les domaines d'activité industrie, commerce, services, recherche scientifique.

- ❖ Systèmes de compagnies aériennes
- ❖ Systèmes bancaires, d'assurance, commerciaux
- ❖ Bases de données scientifiques, techniques
- ❖ Biologie
- ❖ Astronomie
- ❖ Produits industriels
- ❖ Bases de données bibliographiques

et, de plus en plus, interactions entre applications de divers domaines

santé, transports, tourisme, …

Exemples (1)

Une BDD pour une compagnie aérienne.

Pour supporter les réservations :

quelles informations doivent être stockées ?

quels types d'interrogations sont souhaités ?

- Les données :

les appareils

les vols

les aéroports

les réservations

les achats

- Les types d'interrogations :

quels sont les vols au départ de X et arrivant à Y le 15 mars 2004 ?

quels sont les prix de ces vols ?

combien de passagers ont voyagé sur le vol 1234 du 15 mars 2004 ?

Exemples (2)

SGBD relationnel : les données sont stockées dans des tables

Exemple :

Vols	n°vol	compagnie	type_avion
	123	Air France	Boeing 747
	234	Alitalia	Airbus A340
	…	….	…

Requête SQL: Donner le type d'avion du vol 123

```
SELECT  type_avion
FROM    Vols
WHERE   n°vol = 123 ;
```

1.7. Différents types de BDD

- Bases hiérarchiques (schéma arborescent)
- Bases réseaux (id. mais plus rapides) 20%
- Bases relationnelles (tables et algèbre relationnel) 75%
- Bases déductives (tables et logique du 1er ordre)
- Bases objets (formalisme objet d'héritage).

1.8. Objectifs et avantages des SGBD

- ✓ **Indépendance physique** : Pouvoir modifier les structures de stockage ou les index sans que cela ait de répercussion au niveau des applications.
- ✓ **Manipulation facile des données** : un utilisateur non informaticien doit pouvoir manipuler simplement les données.

- ✓ **Administration facile des données** : un SGBD doit fournir des outils pour décrire les données, permettre le suivi de ces structures et autoriser leur évolution (administrateur de BDD)
- ✓ **Efficacité des accès aux données** : garantie d'un bon *débit* (nb. de transactions /s) et d'un bon *temps de réponse* (temps moyen d'attente pour une transaction)
- ✓ **Non redondance des données** : éviter la duplication d'informations pour diminuer le volume de stockage et éviter les incohérences.
- ✓ **Cohérence des données** : ex: l'âge d'une personne est un nombre positif. Le SGBD veille à ce que les applications respectent cette règle (*contrainte d'intégrité*).
- ✓ **Partage des données** : utilisation simultanée des données par différentes applications.
- ✓ **Sécurité des données** : les données doivent être protégées contre les accès non autorisés ou en cas de panne.

1.9. Fonctions des SGBD

- Description des données: *Langage de Description des données* (LDD)
- Recherche des données (langage de manipulation de données LMD)
- Mise à jour des données (langage de manipulation de données LMD)
- Transformation des données (langage de manipulation de données LMD)
- Contrôle de l'intégrité des données.
- Gestion de transactions et sécurité.

1.10. Quelques SGBD connus et utilisés :

Il existe de nombreux systèmes de gestion de bases de données, en voici une liste non exhaustive :

- ACCESS : plate-forme Windows, mono -poste, licence commerciale
- SQL SERVER : plate-forme Windows, mode client/serveur, licence commerciale

- ORACLE : plate-forme Windows et Linux, mode client/serveur, licence commerciale
- SYBASE : plate-forme Windows et Linux, mode client/serveur, licence commerciale
- POSTGRESQL : plate-forme Windows et Linux, mode client/serveur, licence libre
- MYSQL : plate-forme Windows et Linux, mode client/serveur, licence libre.

2-Le modèle E/A

2.1. Analyse et conception de BD

Conception d'une BDD

Une BDD peut stocker une quantité considérable d'informations, celles-ci peuvent évoluer dans le temps et peuvent subir des mises à jours.

Vu l'importance d'une BDD, sa création doit être réfléchie. La meilleure façon de la concevoir et d'anticiper tous les besoins en information et à long terme. Toutefois, une BDD peut être conçue par

approche successive.

Avant de se lancer dans la création d'une BDD, on devrait prendre quelques instants pour structurer nos besoins en information. On propose les étapes d'analyse suivantes :

- **Première étape** : définir l'objet ou le domaine d'étude et décrire tous les paramètres le décrivant.
- **Deuxième étape** : faire un découpage des informations jusqu'au plus petit élément significatif. Le découpage doit permettre de répartir les informations selon leurs natures et leurs dépendances.
- **Troisième étape** : structurer la BDD en des tables.
- **Quatrième étape** : déterminer les relations qui peuvent exister entre les tables
- **Le résultat** de la conception est : le schéma de la BDD

Exemple : suivi des notes des étudiants dans un établissement d'enseignement

Réalisation d'une BDD

Une fois l'analyse et la conception de la BDD terminées, on doit passer à sa réalisation à l'aide d'un SGBD tel que ACCESS.

Cela nécessite les étapes :

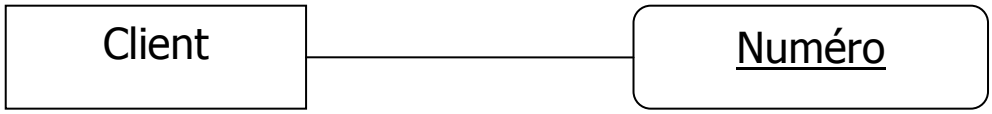
- Création des tables de la BDD : la création une table passe par deux étapes :
  *Création de sa structure (définition des champs)

  *Saisie des informations (création des enregistrements)
- Définition des relations entre tables
- Interrogation de la BDD : Une BDD ne doit pas être sans intérêt, elle doit restituer des informations à la demande des utilisateurs. Une demande constitue une interrogation (requête) de la BDD.
- Établissement des états à imprimer : l'utilisateur d'un BDD a toujours besoin d'afficher et d'imprimer des listings (états) contenants des données obtenues à partir des tables ou des requêtes.

2.2. Le formalisme E/A

E/A signifie entité/Association

- Modèle conceptuel des années 1970
- Il utilise une représentation graphique
- Le modèle E/A est un outil permettant d'analyser les situations du monde réel Entités et attributs.
- **Entité:** objet (abstrait ou concret) qui peut être distingué d'un autre objet
- **Classe d'entités:** ensemble d'entités similaires
- **Attribut:** propriété caractéristique d'une entité
- **Identifiant / clé :** une clé ou un identifiant est un attribut ou un ensemble d'attributs dont les valeurs identifient de façon unique les occurrences d'un type d'entité. Ex: le numéro de sécurité sociale est une clé.

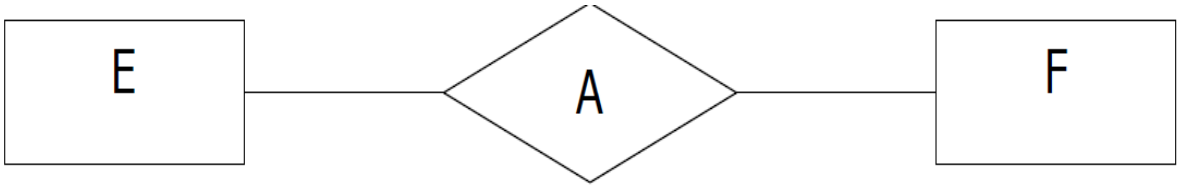


2.3. Association

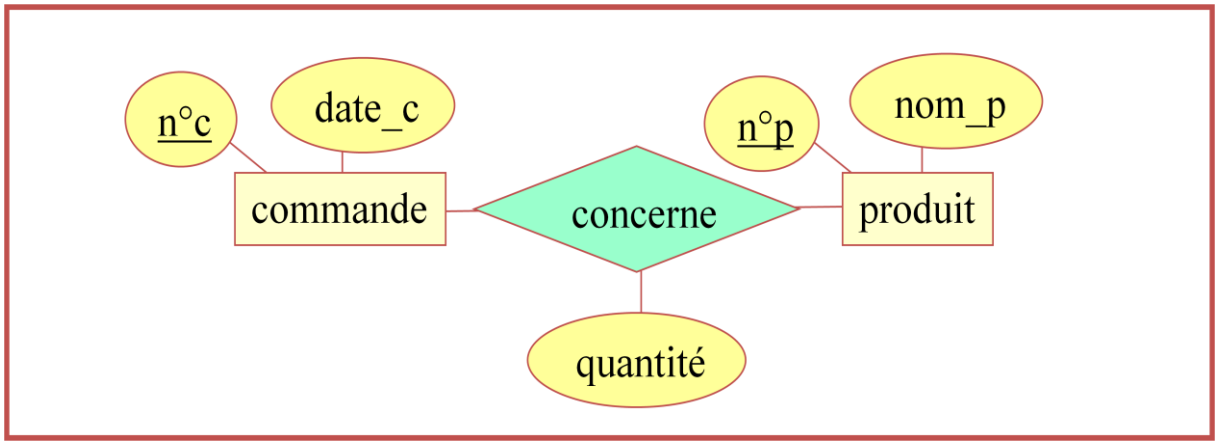
Association : regroupement d'entités traduisant une réalité c-a-d une liaison entre des entités.

Les associations similaires sont regroupées en classes d'associations

Les entités CLIENT et PRODUITS sont dites participantes à la relation COMMANDE



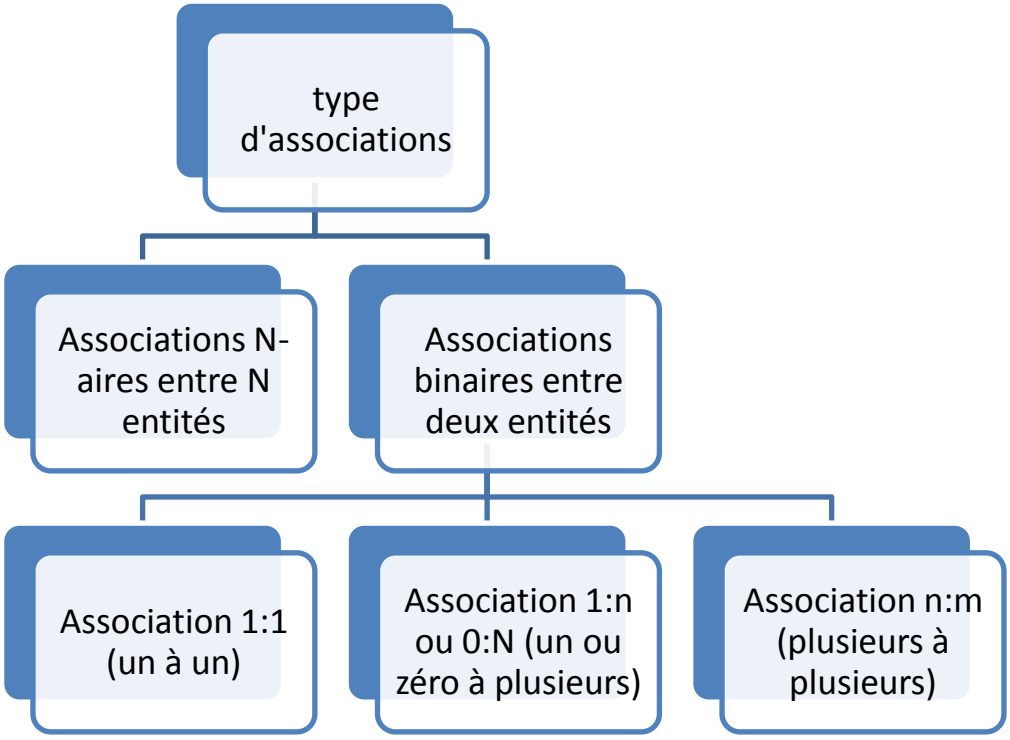
Attributs de relation



2.4. Types d'associations

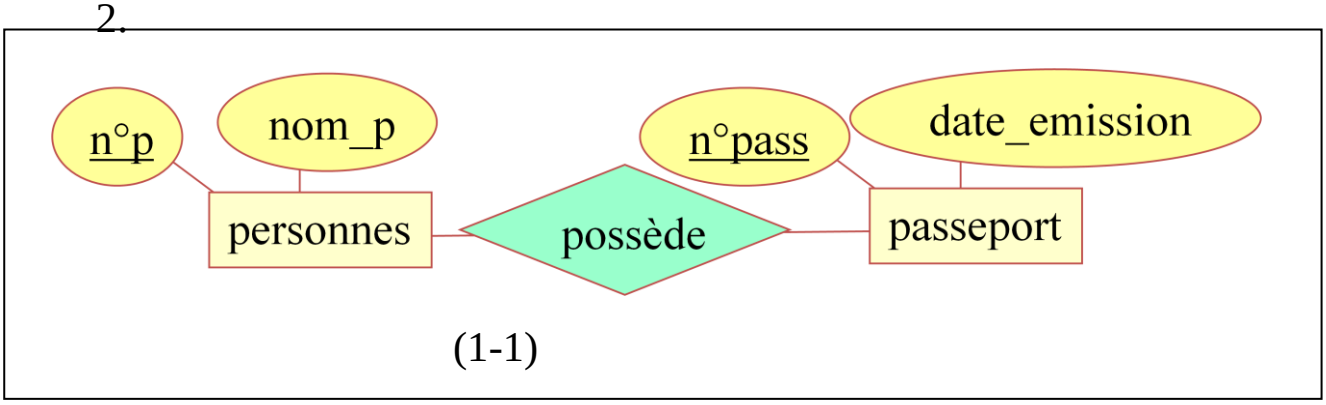
Le type d'association caractérise le nombre de liens entre entités:

- Associations N-aires entre N entités
- Associations binaires entre deux entités

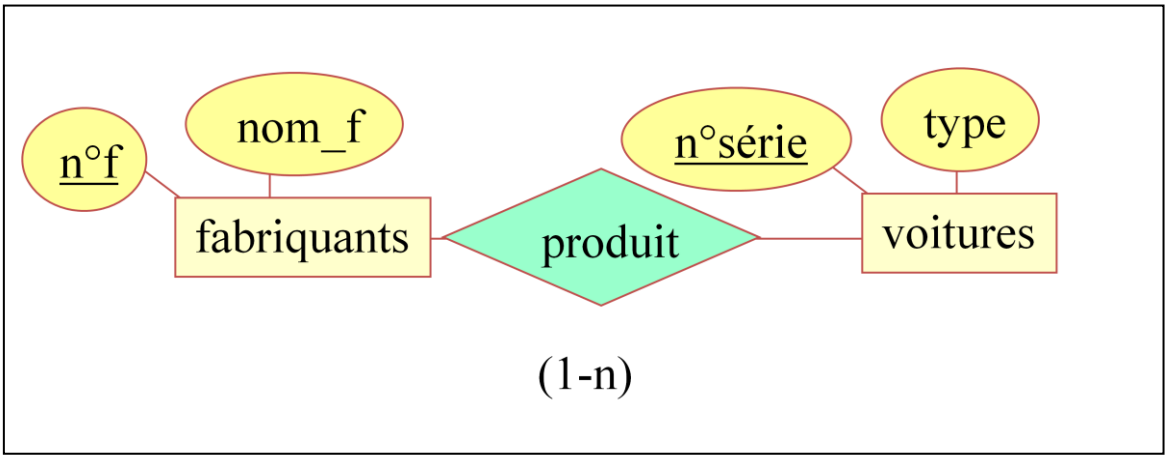


Exemples des associations binaires entre deux entités

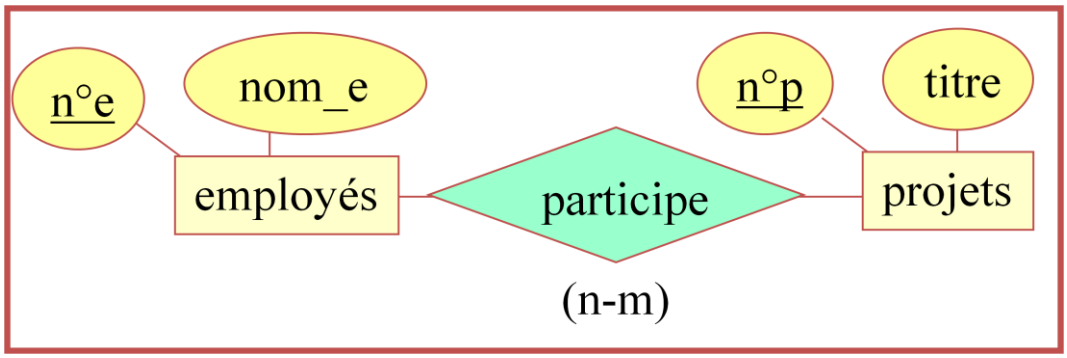
1. Association 1:1 (un à un)
2.



3. Association 1:n ou 0:N (un ou zéro à plusieurs)



4. Association n:m (plusieurs à plusieurs)

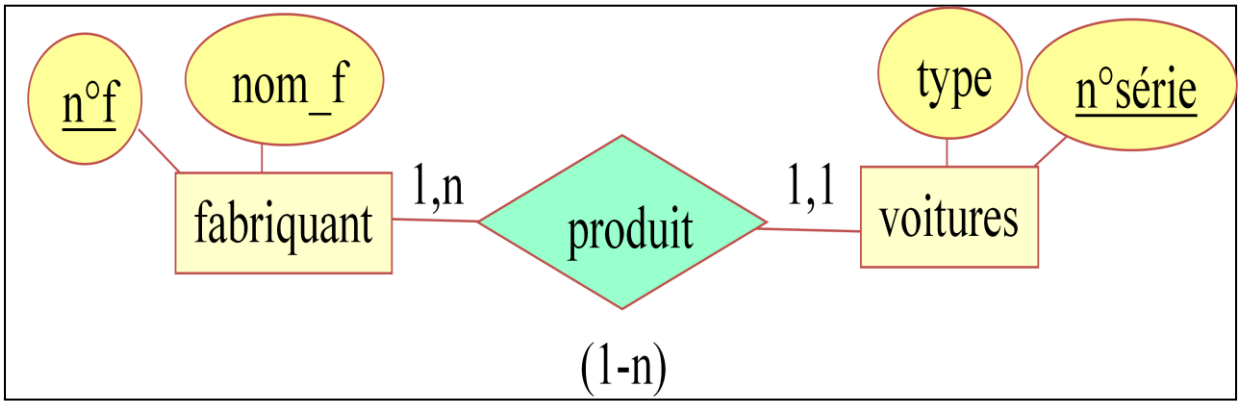


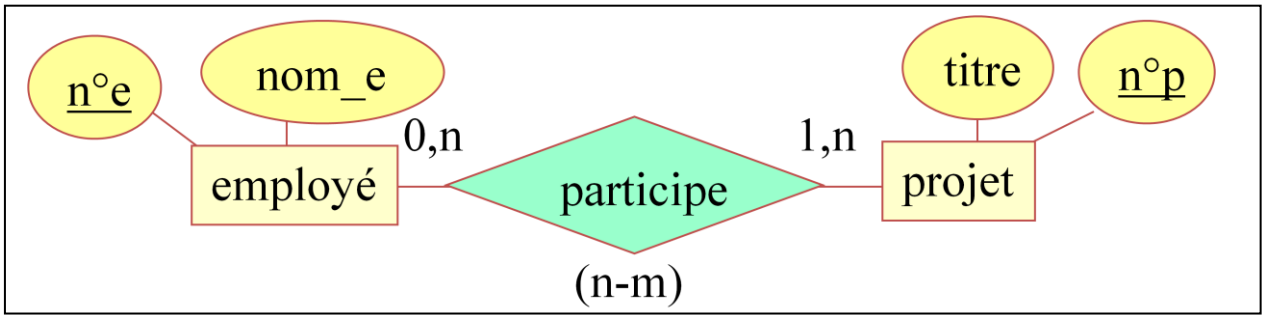
2.5. Cardinalité

La cardinalité d'un couple entité-association (E,A) est la données d'un couple d'entiers (m,M)

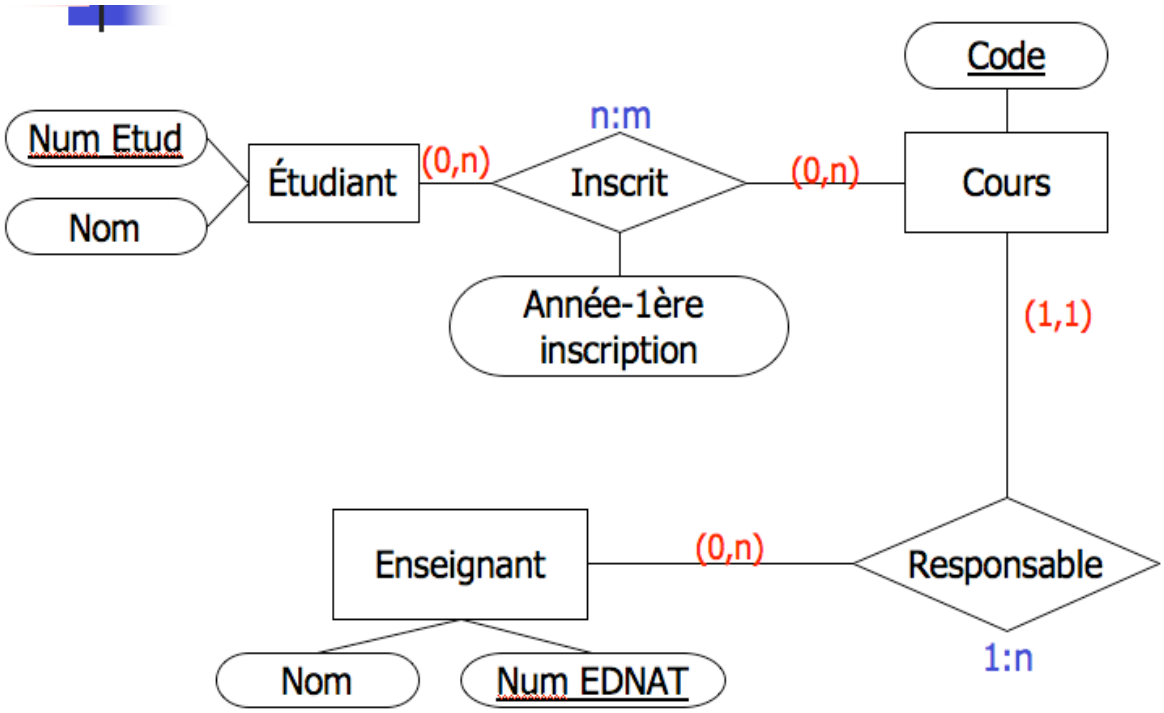
Elle est définie par une cardinalité minimum m et maximum M
m définit le nombre minimum d'associations de classe A pouvant exister pour une entité de classe E.

Exemples





2.6. Exemple de diagramme E/A



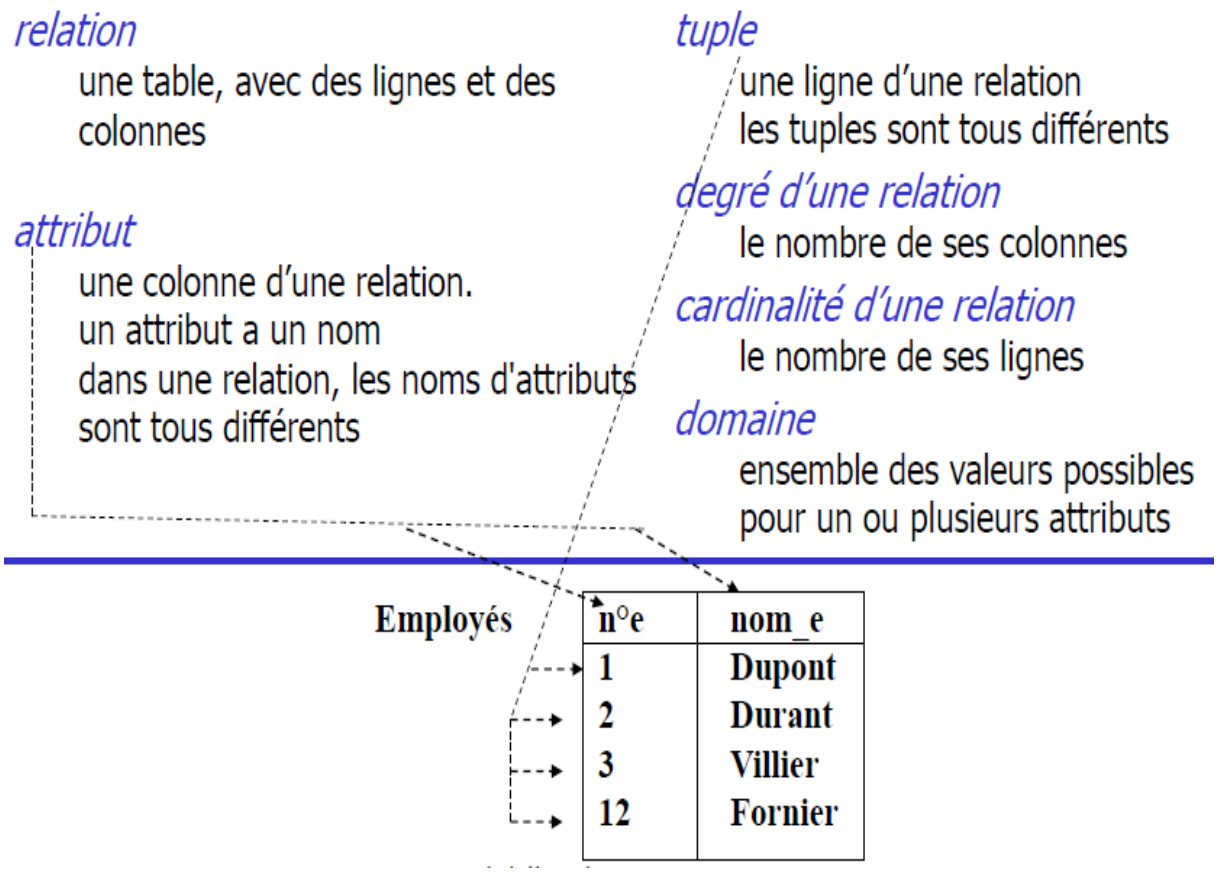
Synthèse

- Un modèle E/A contient les éléments suivants:
  - Entités
    - Propriétés (ou attributs)
    - Clés primaires
  - Relations
    - Éventuellement munies de propriétés (ou attributs)
  - Cardinalités
- Ces éléments sont TOUS OBLIGATOIRES

3- Le modèle relationnel

3.1 Principes fondamentaux du modèle relationnel

3-1-1 Du modèle E/A au modèle relationnel



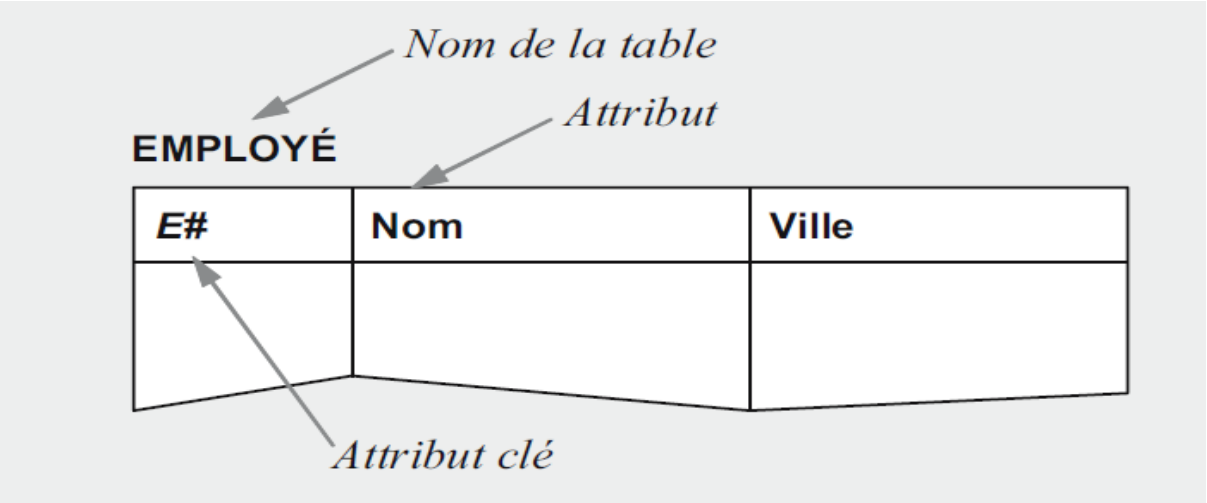
3.1.2 Table

Une table est un ensemble de données organisées en lignes et en colonnes.

Par exemple, la table de la figure 1 est conçue pour recueillir des informations sur les employés d'une entreprise. Elle porte un nom écrit en majuscule, EMPLOYÉ. Les noms de colonnes désignent les attributs des employés. Dans notre table, ce sont le numéro d'employé «E#», le nom de l'employé «Nom» et son domicile «Ville».

l'attribut E# permet d'identifier chaque employé de manière unique. En vertu de cette propriété, nous déclarons que le numéro de l'employé est une clé.

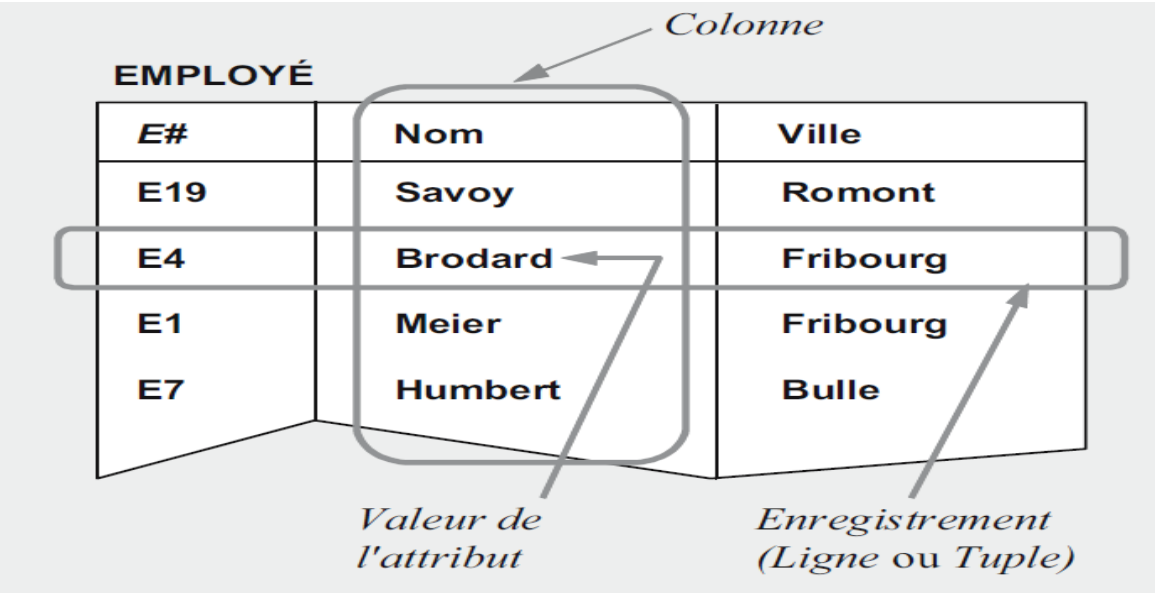
figure 1



- Les lignes sont appelées : Enregistrements
- Les colonnes sont appelées : Champs
- Une *clé d'identification* ou *clé* (*identification key*, en anglais) dans une table est un attribut ou une combinaison minimale d'attributs dont les valeurs clés permettent de désigner chaque enregistrement.

Introduisons maintenant les données du personnel dans la table EMPLOYÉ ligne par ligne, comme indiqué dans la figure 2

figure 2



3.1.3. Les propriétés d'une relation

1. Chaque table porte un nom unique.

2. À l'intérieur de la table, le nom de chaque attribut est unique et désigne une colonne avec des propriétés spécifiques.

3. Une table peut contenir un nombre quelconque d'attributs, l'ordre des colonnes dans la table est indifférent.

4. L'un des attributs ou une combinaison d'attributs identifie de façon unique chaque enregistrement dans la table et sera la clé primaire.

5. Une table peut contenir un nombre quelconque d'enregistrements, l'ordre des enregistrements dans la table est indifférent.

Selon cette définition, dans un *modèle relationnel* (*relational model*, en anglais) chaque table est vue comme un *ensemble non ordonné d'enregistrements.*

3-1-4 . Le passage du modèle entité-association au schéma de base de données relationnelle

Nous abordons maintenant la traduction d'un modèle entitéassociation en un schéma de base de données relationnelle. Il s'agit essentiellement de définir une méthode qui permet de représenter les ensembles d'entités et de liens par des tables.

Règle 1 (ensemble d'entités)

Chaque ensemble de liens complexe-complexe doit être transformé en une table distincte. Elle contient au moins, comme clés étrangères, les clés d'identification des ensembles d'entités qui participent à la liaison. La clé primaire de la table est soit la clé d'identification formée par la concaténation des clés étrangères, soit

une autre clé candidate. D'autres attributs de l'ensemble de liens deviennent les attributs de la table.

Chaque ensemble d'entités doit être traduit en une table distincte, dotée d'une clé primaire qui peut être soit la clé correspondante de l'ensemble d'entités, soit une clé candidate. Les autres attributs de l'ensemble d'entités complètent les attributs de la table.

La définition d'une table (section 1.1) requiert une clé primaire unique. Lorsqu'il existe dans une table plusieurs clés candidates (candidate keys, en anglais) qui satisfont toutes aux critères d'unicité et de minimalité, c'est l'architecte de données qui décide du choix de l'une d'elles comme clé primaire.

Règle 2 (ensembles de liens)

Chaque ensemble de liens peut être traduit en une table distincte.

Les clés d'identification des ensembles d'entités participantes doivent y figurer comme clés étrangères. La clé primaire de cette table peut être soit une clé d'identification formée par la concaténation des clés étrangères, soit une autre clé candidate en créant par exemple une clé artificielle. Les autres attributs de l'ensemble de liens complètent les attributs de la table.

Règle 3 (Liaisons de type complexe-complexe)

Chaque ensemble de liens complexe-complexe doit être transformé en une table distincte. Elle contient au moins, comme clés étrangères, les clés d'identification des ensembles d'entités qui participent à la liaison. La clé primaire de la table est soit la clé

d'identification formée par la concaténation des clés étrangères, soit une autre clé candidate. D'autres attributs de l'ensemble de liens deviennent les attributs de la table.

Règle 4 (liaisons de type simple-complexe)

Un ensemble de liens simple-complexe peut s'exprimer dans les deux tables des ensembles d'entités participantes sans avoir besoin de créer une table distincte. Pour cela, dans la table qui participe à l'association simple (association de type 1 ou c), on définit une clé étrangère qui la relie à la table référencée et l'on ajoute éventuellement d'autres attributs de l'ensemble de liens considéré.

règle 4 nous permet de l'exprimer dans une table existante en ajoutant à celle-ci une clé étrangère qui fait référence à l'autre table dans l'association. Pour mettre en évidence une liaison, il convient de suffixer la clé d'identification correspondante par un *nom de rôle* qui permet de comprendre le rôle d'une clé spécifique dans une table étrangère.

Règle 5 (liaisons de type simple-simple)

Un ensemble de liens simple-simple *peut* s'exprimer dans les deux tables issues des ensembles d'entités participantes *sans avoir besoin de créer une table distincte*. L'une des deux tables est la table référencée dont la clé d'identification apparaît comme clé étrangère dans l'autre table.

De nouveau, il est important de désigner correctement la table référencée à laquelle on empruntera une clé candidate pour définir la clé étrangère dans l'autre table. En principe, on introduit la clé

étrangère dans la table qui participe à la liaison avec le type d'associations 1.

3.2 SQL, langage normalisé au niveau international

Comme nous l'avons déjà vu, un modèle relationnel représente les informations sous la forme tabulaire. À chaque table est associé un ensemble de tuples ou d'enregistrements de même type. Ce concept d'ensemble permet d'implémenter des opérations ensemblistes pour interroger et manipuler les bases de données.

Le plus important langage de requête et de manipulation des tables s'appelle Structured Query Language ou SQL en abrégé (voir figure3)

Les normes du langage SQL sont élaborées par deux organismes, l'ANSI (American National Standards Institute) et l'ISO

(International Organization for Standardization)

*SQL est utilisé dans de nombreux SGBD commercialisés :
• DB2, SQL/DS (gros ordinateurs sous systèmes d'exploitation MVS, VM/CMS)
• Ingres, Informix
• ORACLE (gros ordinateurs et micro-ordinateurs)
• DB4, PARADOX (micro-ordinateurs).

*En SQL

Table (table)désigne une collection de lignes ou n-uplets (généralisation de la notion de relation car plusieurs lignes peuvent être identiques)

Colonne (column) désigne un attribut

Ligne (row) désigne un n-uplet.

La commande SELECT

En SQL La commande SELECT appelée question ou interrogation ou requête (query) est fondamentale.

Forme générale

SELECT [ALL / DISTINCT] { liste <colonne/expression> / * }

FROM liste [<propriétaire> .] { <table> / <vue> } [<variable ligne>]

WHERE <condition> ;

- Instruction de base

SELECT [ALL / DISTINCT] { liste < colonne / expression > / * }

FROM liste { < table > / < vue > }

WHERE condition

- Visualisation de toutes les colonnes de la table T

SELECT * FROM T ;

- Visualisation des colonnes C1, C2 et T

SELECT C1, C2 FROM T;

- Visualisation d'une partie de T

SELECT * FROM T WHERE Condition ;

- ***Opérateurs***
  - Logiques : AND OR NOT
  - Conditionnels : < = > <= >= <>
  - Prédicat :
    - Between (ensemble fermé : bornes incluses)
    - IN ( )
    - LIKE _ : 1 et 1 seule occurrence dans le champ
    - LIKE % :

Ex : lieu LIKE ‘’_RI_’’
→PARIS

Lieu LIKE ‘’%ER’’ → fin du mot
Lieu LIKE ‘’%ER%’’

- ***Fonctions d’évaluation***
  - Comptage
    - COUNT

SELECT COUNT (DINSTINCT NATIONALITE) FROM JOUEUR

    - AVG

SELECT AVG (PRIME) FROM GAIN

    - SUM
    - MIN
    - MAX Numérique et string

- Requêtes imbriquées 1 seule colonne

SELECT _ FROM _
WHERE (SELECT _ FROM _ WHERE _)

- Fonctions d'évaluation

SELECT _ FROM _ [WHERE _]
GROUP BY liste colonne [Having < Condition >]

- Tri

SELECT _ FROM _ [WHERE _]
[GROUP BY _]
ORDER BY { liste colonne / entier } [ASC /DESC]

- Fusion

SELECT _
FROM _

UNION Même type (char, int, …) et même nombre de colonnes

SELECT _
FROM _

- Vue

CREATE VIEW < VUE >
AS < Commande SELECT >

- Modification de structure

- Ajout attribut

ALTER TABLE < TABLE >
ADD (liste < COLONNE > < TYPE >)

- Modification attribut

ALTER TABLE < TABLE >
MODIFY (Liste < COLONNE > < TYPE >)

Attention :

  - Réduction de taille : il faut que toutes les instances soient nulles
  - Modifier le type : idem
- Renommer Table

RENAME < Ancien nom > TO < Nouveau nom >

- ***Ajout n-uplet***

INSERT INTO < TABLE > [< liste colonne >] VALUES < liste valeurs >

- ***Mise à jour***

UPDATE < TABLE >
SET < liste colonne > = < expression >
[WHERE condition]

- ***Suppression de table***

DROP TABLE < TABLE >

- ***Suppression de données***

DELETE FROM < TABLE > [WHERE < condition >]

Exercice : Algèbre relationnel (Manière différente)

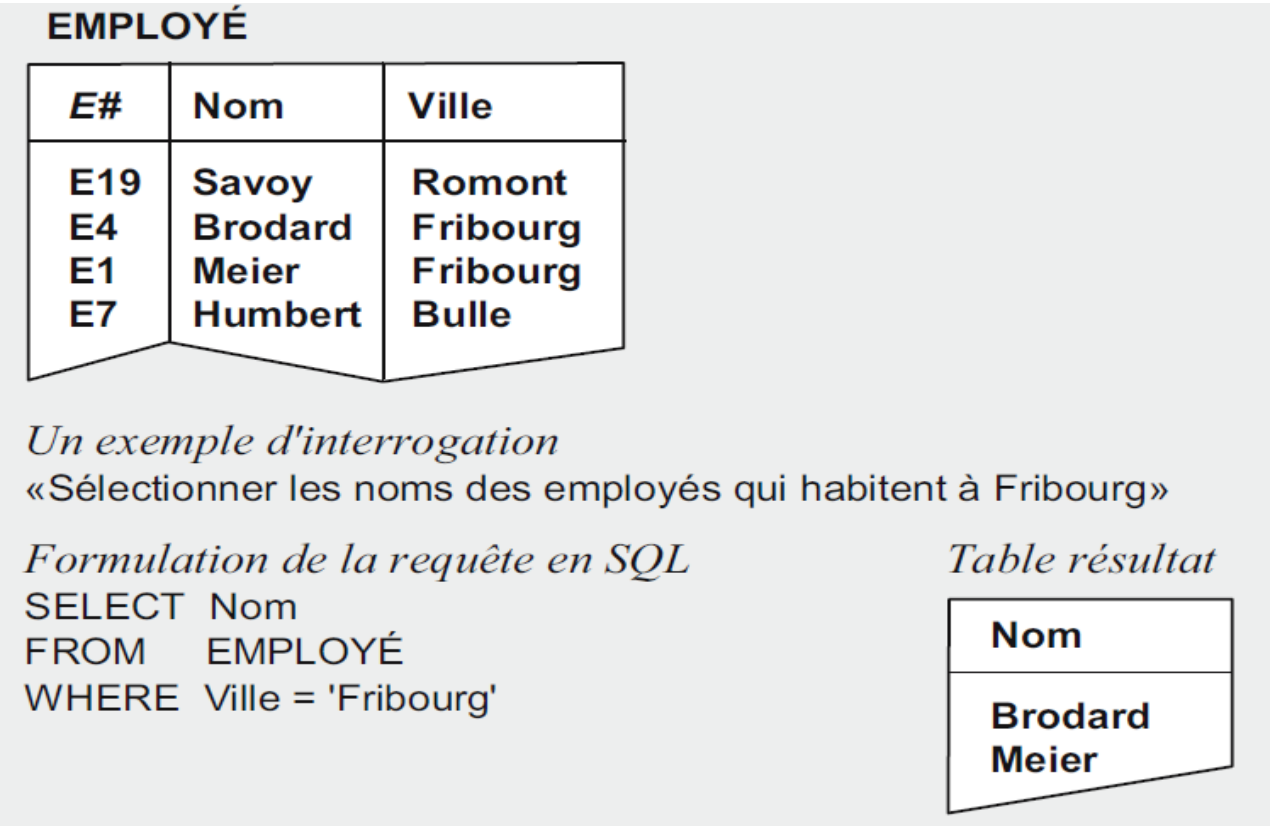
5. SELECT NOM JOUEUR FROM GAIN WHERE LIEU = ‘RG’ AND PRIME >= 1MF AND NOMJOUEUR NOT IN (SELECT NOMJOUEUR FROM GAIN WHERE LIEU = ‘RG’ AND PRIME < 1MF);

9. SELECT R1.NOMPERDANT FROM RENCONTRE R1 WHERE R1.NOMPERDANT NOT IN (SELECT R2.NOMGAGNANT FROM RENCONTRE R2);

OU

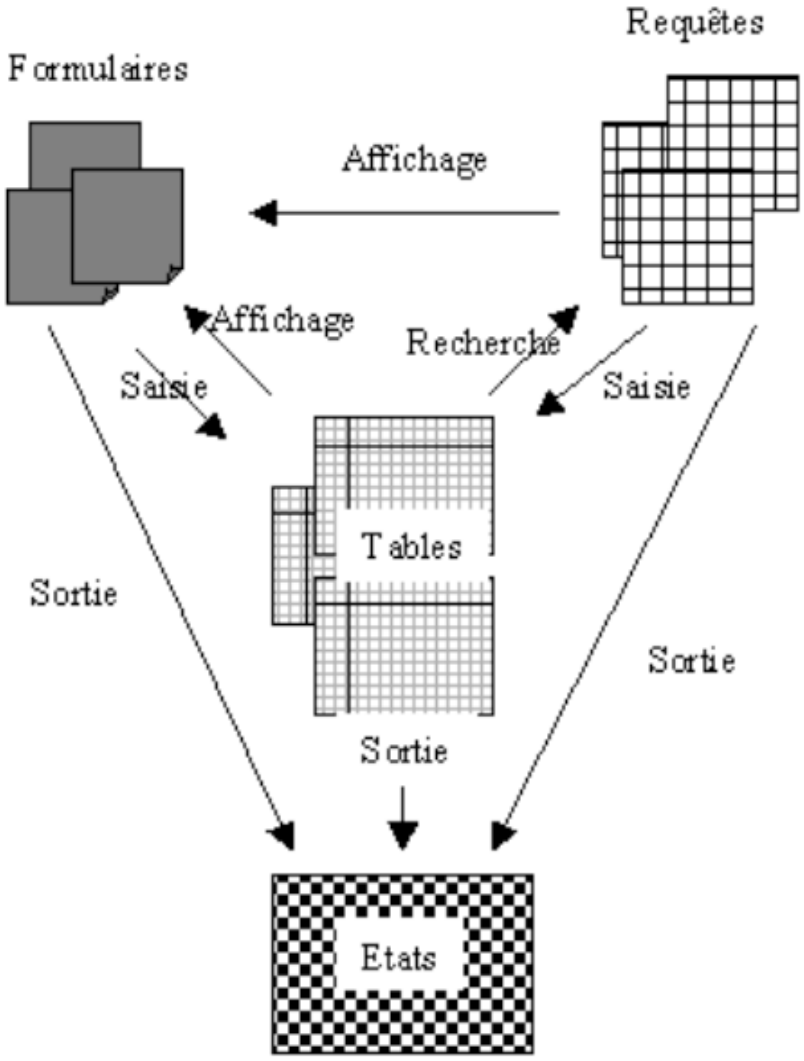
SELECT R1.NOMPERDANT FROM RENCONTRE R1 WHERE NOT EXIST (SELECT * FROM RENCONTRE R2 WHERE R1.NOMPERDANT = R2.NOMGAGNANT);

Figure 3 : Formulation d’une requête en SQL



2- DELPHI

resumé



Annexe : CONNEXION AUX BASES DE DONNÉES

Source : document de la Faculté de Sciences Économiques et de Gestion. Université Lyon 2.

.1. LES COMPOSANTS SOURCES

a) Table
b) Queryc) DataSource..

2. FORMULAIRES BASÉS SUR DES TABLES

a) Composants BD visuels
b) Formulaires simples
c) Naviguer dans les données
d) Utilisation d'une grille
e) Formulaires composés
f) L'expert fiche base de données

3. REQUÊTES SQL

a) Formulaire basé sur une requête
b) Requête paramétrée

4. TRANSFERT DE DONNÉES

5. MANIPULATION DE DONNÉES PAR PROGRAMME

a) Opérations de base
b) Navigation dans la base de données
c) Requêtes SQL
d) Recherche d'enregistrements

1. Les composants sources

Les composants sources permettent d'accéder à des bases de données. Ils sont accessibles dans l'onglet *AccèsBD* de la palette des composants.

NB : Ces composants sont « invisibles », i.e., non visuels. Ils apparaissent sur une fiche en mode création mais pas à l'exécution du programme.

a) Table

Le composant *Table* permet d'accéder aux données contenues dans une table relationnelle.

Propriétés principales

Propriété	Description
`Active`	Ouvre ou ferme la table
`DataBaseName`	Nom de la base de données contenant la table
`Exclusive`	Empêche d'autres utilisateurs d'ouvrir la table si positionnée à true
`MasterFields`	Attributs de la table (détail) liés à une table maître
`MasterSource`	Source de données d'une table maître
`ReadOnly`	Autorise ou non l'utilisateur à modifier la table
`TableName`	Nom de la table

b) Query

Le composant *Query* (requête) permet d'effectuer une sélection sur une base de données. Il est identique au composant *Table*, mis à part la provenance des données.

Propriétés principales

Propriété	Description
`Active`	Exécute ou non la requête
`DataBaseName`	Nom de la base de données interrogée
`DataSource`	Origine des valeurs à fournir aux paramètres de la requête
`Params`	Paramètres de la requête
`SQL`	Libellé de la requête SQL

c) DataSource

Le composant *DataSource* sert à visualiser les enregistrements d'une table ou d'une requête dans des composants visuels de Delphi. Tous ces composants BD visuels utilisent un composant *DataSource* comme source de données.

Propriété principale

Propriété	Description
`DataSet`	Indique le composant (*Table* ou *Query*) source des données

2. Formulaires basés sur des tables

a) Composants BD visuels

Une fiche Delphi sur laquelle apparaissent des données issues d'une base de données est conçue à partir de composants similaires aux composants classiques, les composants BD visuels, réunis dans l'onglet *ContrôleBD* de la palette des composants.

Composants BD principaux

Contrôle	Nom Pascal	Description
Texte BD	`DBText`	Texte non modifiable
Édition BD	`DBEdit`	Boîte d'édition d'une ligne
Mémo BD	`DBMemo`	Boîte d'édition multi-lignes
Image BD	`DBImage`	Image
Navigateur BD	`DBNavigateur`	Barre d'icônes pour la navigation dans la BD

b) Formulaires simples

Pour construire un formulaire simple, il suffit de suivre les étapes suivantes.

1. Placer un composant *Table* sur la fiche.
   Donner une valeur à la propriété *DatabaseName*.
   Donner une valeur à la propriété *TableName*.
   Ouvrir la table en positionnant la propriété *Active* à true.
2. Placer un composant *DataSource* sur la fiche.
   Sélectionner la table créée à l'étape 1 dans la propriété *DataSet*.
3. Ajouter sur la fiche autant de composants BD visuels que nécessaire pour afficher les données.
   Sélectionner le composant *DataSource* créé à l'étape 2 dans la propriété *DataSource* de chaque composant.
   Sélectionner un champ de la table dans la propriété *DataField* de chaque composant.

c) *Naviguer dans les données*

La façon la plus simple de naviguer dans les données est d'utiliser le composant navigateur BD (*DBNavigator*). C'est un composant graphique représentant des boutons type magnéto-cassette qui permettent de passer d'un enregistrement à l'autre, de sauter en fin de table, etc.

Pour utiliser un navigateur BD, il suffit de l'ajouter à la fiche contenant les données et de donner une valeur à sa propriété *DataSource*.

d) *Utilisation d'une grille*

Il est possible de visualiser plus d'un enregistrement à la fois à l'aide du composant universel grille BD (*DBGrid*), qui permet d'obtenir une vue des données sous forme tabulaire. Ce composant s'adapte à la structure de la table référencée afin d'en montrer tous les champs.

Pour construire un formulaire basé sur une grille, il suffit de reprendre les étapes 1 et 2 du b), puis d'ajouter un composant grille et de donner une valeur à sa propriété *DataSource*.

e) *Formulaires composés*

Les formulaires composés permettent de visualiser des associations 1, N entre deux tables. La démarche de construction d'un tel formulaire est la suivante.

1. Placer deux composants *Table* et deux composants *DataSource* sur la fiche (cf. étapes 1 et 2 du b)). L'une des tables sera la *table maître* (côté « 1 » de la relation 1, N) et l'autre la *table détail* (côté « N » de la relation 1, N).
2. Sélectionner la table détail.
   Indiquer dans sa propriété *MasterSource* le composant *DataSource* associé à la table maître.
   Éditer la propriété *MasterFields* pour effectuer le lien entre les tables (apparition d'une boîte de dialogue). Cliquer dans les parties *champ détail* et *champ maître* sur-le-

champ permettant de faire la jointure entre les deux tables, ajouter les champs joints (bouton ajouter) et valider.

3. Afficher les informations des deux tables à l'aide des composants BD visuels.

f) L'expert fiche base de données

L'expert fiche BD est accessible par le menu Base de données/Expert fiche… Il permet de dessiner facilement, grâce à une série de boîtes de dialogues, des formulaires simples ou composés.

3. Requêtes SQL

Le propos de cette section n'est pas de présenter le langage SQL, mais d'indiquer comment formuler et exécuter une requête SQL avec Delphi.

a) Formulaire basé sur une requête

La démarche est la même que pour créer un formulaire simple basé sur une table. Il suffit de remplacer le composant *Table* par un composant *Query* et de renseigner la propriété *SQL*.

b) Requête paramétrée

Soit la requête SQL suivante : `SELECT * FROM CLIENT WHERE VILLE = 'Paris'.`

Pour accéder aux habitants de Lyon, une autre requête similaire est nécessaire. Et ainsi de suite pour toutes les autres villes. Pour remédier à ce problème, Delphi permet l'insertion de paramètres dans le texte de la requête.

Version paramétrée de la requête initiale : `SELECT * FROM CLIENT WHERE VILLE = :ville.`

Il suffit d'instancier le paramètre « ville » avant l'activation de la requête (propriété *Params*).

4. Transfert de données

Lorsque Delphi effectue une requête, il utilise une table virtuelle (en mémoire) pour stocker le résultat. Pour conserver ce résultat sur disque, il faut créer une table réponse et y recopier les données de la table virtuelle. Pour ce faire, il faut disposer sur une fiche les composants suivants :

- un composant *Query* pour effectuer une requête ;

- un composant *Table* pour désigner la table physique (sur disque) à créer ;
- un composant *BatchMove* (onglet *AccèsBD* de la palette des composants) pour effectuer le transfert.

Attribuer à la propriété *Source* du composant *BatchMove* le composant requête, à la propriété *Destination* le composant table et à la propriété *Mode* la valeur `batCopy` (ce qui permet de créer la table si elle n'existe pas).

Il suffit ensuite de programmer un bouton de commande pour appeler la méthode *Execute* du composant *BatchMove*.

Pour transférer des enregistrements d'une table à une autre, il suffit de calquer l'opération précédente en remplaçant le composant requête par un composant table.

5. Manipulation de données par programme

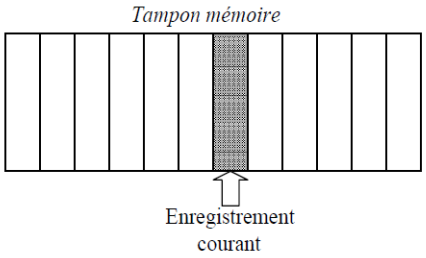
a) Opérations de base

Classe *DataSet*

Les composants *Table* et *Query* héritent indirectement de la classe *DataSet*, qui permet de représenter un ensemble de données.

Enregistrement courant

Lorsqu'un *DataSet* est ouvert, un certain nombre d'enregistrements sont lus sur disque et placés dans un tampon mémoire. Cependant, un seul enregistrement est considéré comme l'enregistrement courant. La lecture d'un champ ou la suppression d'un enregistrement se font toujours par rapport à l'enregistrement courant



Lecture d'un champ

L'accès au contenu d'un champ se fait par la propriété *Fields* du *DataSet*, qui est un tableau de champs. L'accès à la valeur d'un champ peut se faire par sa position dans le tableau ou par le nom de ce champ, grâce à la méthode *FieldByName*. Puisqu'un champ peut être de nature très variée, son contenu est stocké en mémoire sous forme *physique*. Lorsque cette donnée est manipulée, elle doit l'être sous forme *logique*. Par conséquent, il faut indiquer dans quel format logique on désire accéder à la donnée.

Ex.
```
var a: integer;
Begin
    a:=Table1.Fields[0].asInteger;
    a:=Table1.FieldByName('Numero').asInteger;
    { Conversion physique-logique :
        asBoolean
        asFloat
        asInteger
        asString                    }
```

Édition d'une table

Par défaut, le *DataSet* est en mode édition, c'est-à-dire que ses données peuvent être modifiées via des contrôles. Pour empêcher cela, il faut passer la propriété *AutoEdit* du composant *DataSource* à `false`. Pour passer le *DataSet* en mode édition, il faut appeler sa méthode *Edit*.

Ex.
```
Table1.Edit;
```

Ajout/insertion d'enregistrements

- Ajout : méthode *Append* ; un enregistrement vierge est ajouté à la fin de la table.
- Insertion : méthode *Insert* ; l'enregistrement inséré se place avant l'enregistrement courant

Il faut ensuite remplir chaque champ.

Ex.
```
Table1.Insert;
Table1.FieldByName('Numero').asInteger:=100;
Table1.Append;
Table1.FieldByName('Numero').asInteger:=500;
```

Il est également possible d'ajouter ou d'insérer un enregistrement tout en remplissant mes champs grâce aux méthodes *AppendRecord* et *InsertRecord*.

Ex.
```
Table1.InsertRecord(900,'Champ 2','Champ3',3.1416);
```

Validation/annulation des mises à jour

Méthode du *DataSet*	Description
`Post`	Valide la saisie et quitte le mode édition
`Refresh`	Valide la saisie, quitte le mode édition et rafraîchit les données visibles à l'écran
`Cancel`	Annule les modifications apportées à un enregistrement tant qu'elles n'ont pas été validées

Ex.
```
Table1.Refresh;
```

b) Navigation dans la base de données

Enregistrement précédent, suivant, premier et dernier

Méthode du *DataSet*	Description
`Prior`	L'enregistrement courant devient l'enregistrement précédent
`Next`	L'enregistrement courant devient l'enregistrement suivant
`First`	L'enregistrement courant devient le premier enregistrement
`Last`	L'enregistrement courant devient le dernier enregistrement

Ex.
```
Table1.First;
Table1.Next;
Table1.Next;
```

Début et fin de table

Méthode du *DataSet*	Description
`BOF`	Début de table (*avant* le premier enregistrement)
`EOF`	Fin de table (*après* le dernier enregistrement)

Ex.
```
Table1.First;
While not Table1.EOF do
    Begin
        Writeln(Table1.FieldByName('Numero').asInteger);
        Table1.Next;
    End;
```

c) Requêtes SQL

Requête simple

La propriété SQL d'un composants *Query* est de type *Strings*, c'est-à-dire un tableau de chaînes de caractères. Il est donc possible de modifier cette propriété en utilisant les méthodes du type *Strings* et ainsi de créer dynamiquement une requête.

Ex.
```
Query1.Database:='DBDEMOS';
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('select * from clients');
Query1.SQL.Open;
```

Requête paramétrée

Il est également possible d'accéder à la propriété *Params* de façon dynamique.

Ex.
```
Query1.Close;
Query1.Params[0].asString:='Lyon';
Query1.Open;
```

d) Recherche d'enregistrements

Recherche exacte : *SetKey/GotoKey, FindKey*

SetKey positionne le *DataSet* en mode recherche. La propriété *Fields* permet alors de fournir les critères de recherche pour chaque champ indexé. Puis, il suffit d'appeler la méthode *GotoKey* pour activer la recherche. Si aucune correspondance n'est trouvée, *GotoKey* renvoie `false`.

Ex.
```
Procedure TForm1:RchNomPrenom(n, p: string);
Begin
    Table1.SetKey;
    Table1.FieldByName('Nom').asString:=n;
    Table1.FieldByName('Prenom').asString:=p;
    Table1.GotoKey;
End;
```

FindKey permet la même opération en une instruction, en précisant le ou les critères de sélection dans un tableau ouvert.

Ex.
```
Procedure TForm1:RchNomPrenom2(n, p: string);
Begin
    If Table1.FindKey([n, p]) then ShowMessage('Trouvé !');
End;
```

Recherche approchante : *FindNearest*

Ex.
```
Procedure TForm1:RchNom(n: string);
Begin
    If Table1.FindNearest([n]) then ShowMessage('Trouvé !');
End;
```

Filtres : *SetRangeStart/SetRangeEnd/ApplyRange, SetRange*

Les filtres permettent de limiter la vue de la table à certains enregistrements seulement. Ils sont plus particulièrement adaptés aux champs numériques, mais peuvent également être employés sur des champs alphanumériques.

Ex.
```
Table1.SetRangeStart;
Table1.FieldByName('Numero').asInteger:=50;
Table1.SetRangeEnd;
Table1.FieldByName('Numero').asInteger:=100;
Table1.ApplyRange;
```

ou `Table1.SetRange([100],[300]);`

Marqueurs

Les marqueurs sont des objets de type *TBookMark*. Ils permettent de conserver un pointeur sur un enregistrement pour pouvoir y revenir rapidement par la suite.

Méthode du *DataSet*	Description
FreeBookMark	Détruit un marqueur
GetBookmark	Pose un marqueur
GotoBookMark	Va au marqueur

Ex.
```
Var marqueur: TbookMark;
Begin
    { … }
    Marqueur:=Table1.GetBookMark;
    { … }
    Table1.GotoBookMark(marqueur);
    { … }
    Table1.FreeBookMark(marqueur);
End;
```