Faculté de Sciences Économiques et de Gestion

Bases de données

DESS SISE
Année 2003-2004
Jérôme Darmont
http://eric.univ-lyon2.fr/~jdarmont/

Plan du cours

I. Introduction
II. Modèle conceptuel UML
III. Modèle relationnel
IV. Langage de requête SQL
V. Langage PL/SQL d'Oracle

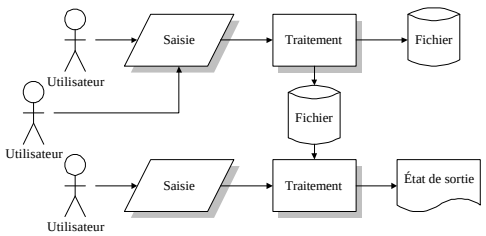
Plan du cours

☞ **I. Introduction**
II. Modèle conceptuel UML
III. Modèle relationnel
IV. Langage de requête SQL
V. Langage PL/SQL d'Oracle

Historique des bases de données

- Jusqu'aux années 60 : organisation classique en **fichiers**
- Fin des années 60 : apparition des premiers SGBD (*Systèmes de Gestion de Bases de Données*), les **systèmes réseaux** et **hiérarchiques**
- À partir de 1970 : deuxième génération de SGBD, les **systèmes relationnels**
- Début des années 80 : troisième génération de SGBD, les **systèmes orientés objet**

Limites des systèmes à fichiers

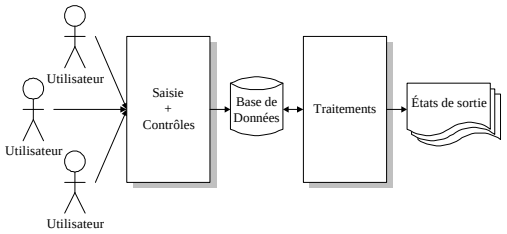


Organisation en fichiers

Limites des systèmes à fichiers

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

Organisation base de données



Organisation base de données

Organisation base de données

- Uniformisation de la saisie et standardisation des traitements (ex. tous les résultats de consultation sous forme de listes et de tableaux)
- Contrôle immédiat de la validité des données
- Partage de données entre plusieurs traitements
  $\Rightarrow$ limitation de la redondance des données

Définitions

- **Base de données (BD) :** Collection de données cohérentes et structurées
- **Système de Gestion de Bases de Données (SGBD) :** Logiciel(s) assurant structuration, stockage, maintenance, mise à jour et consultation des données d'une BD

Propriétés de l'organisation BD

- Usage multiple des données
- Accès facile, rapide, protégé, souple, puissant
- Coût réduit de stockage, de mise à jour et de saisie
- Disponibilité, exactitude, cohérence et protection des données ; non redondance
- Evolution aisée et protection de l'investissement de programmation
- Indépendance des données et des programmes
- Conception *a priori*

Objectifs des SGBD

- **Indépendance physique :** un remaniement de l'organisation physique des données n'entraîne pas de modification dans les programmes d'application (traitements)
- **Indépendance logique :** un remaniement de l'organisation logique des fichiers (ex. nouvelle rubrique) n'entraîne pas de modification dans les programmes d'application non concernés

Objectifs des SGBD

- **Manipulation facile des données :** un utilisateur non-informaticien doit pouvoir manipuler simplement les données (interrogation et mise à jour)
- **Administration facile des données :** un SGBD doit fournir des outils pour décrire les données, permettre le suivi de ces structures et autoriser leur évolution (tâche de l'*administrateur BD*)

Objectifs des SGBD

- **Efficacité des accès aux données :** garantie d'un bon *débit* (nombre de transactions exécutées par seconde) et d'un bon *temps de réponse* (temps d'attente moyen pour une transaction)
- **Redondance contrôlée des données :** diminution du volume de stockage, pas de mise à jour multiple ni d'incohérence

Objectifs des SGBD

- **Cohérence des données :** ex. L'âge d'une personne doit être un nombre entier positif. Le SGBD doit veiller 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

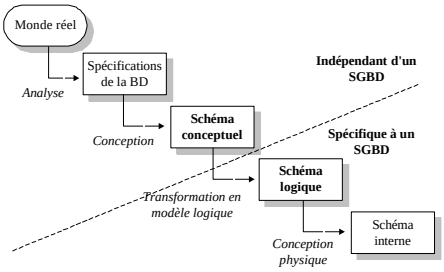
Fonctions des SGBD

- Description des données : *Langage de Définition de Données* (LDD)
- Recherche des données
- Mise à jour des données
- Transformation des données

} *Langage de Manipulation de Données* (LMD)

- Contrôle de l'intégrité des données (respect des contraintes d'intégrité)
- Gestion de transactions (*atomicité* des transactions) et sécurité (mots de passe, etc.)

Processus de conception d'une BD



Plan du cours

I. Introduction

☞ **II. Modèle conceptuel UML**

III. Modèle relationnel

IV. Langage de requête SQL

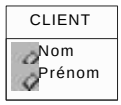
V. Langage PL/SQL d'Oracle

Généralités

- UML = *Unified Modeling Language*
- Ensemble de *formalismes graphiques* pour la modélisation *orientée objet* (analyse)
- Auteurs : J. Rumbaugh, G. Booch, I. Jacobson
- Standard de l'OMG (*Object Management Group*) depuis 1997, standard de fait soutenu par *Rational Software*, Microsoft, Hewlett-Packard, Oracle, IBM…
- Mise en œuvre d'une BD : transformation d'un *diagramme de classes* UML en schéma logique

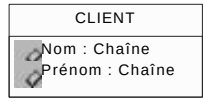
Classes et attributs

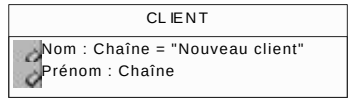
- **Classe :** Groupe d'entités du monde réel ayant les mêmes caractéristiques et le même comportement (ex. CLIENT)
- **Attribut :** Propriété de la classe (ex. Nom et Prénom du client)



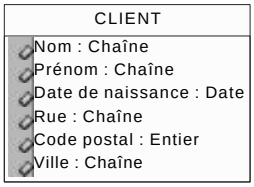
Classes et attributs

- **Type d'attribut :**
  - Entier
  - Réel
  - Chaîne de caractères
  - Date
- **Valeur par défaut** (=)





Classes et attributs



Exemple de classe avec ses attributs

Instances

- **Classe :** ex. CLIENT
- **Instances** (objets) de la classe CLIENT : les clients

Albert Dupont
James West
Marie Martin
Gaston Durand
...

Identifiants

Liste des clients

Nom	*Prénom*	*Date de Naissance*	*Etc.*
Dupont	Albert	01/06/70	...
West	James	03/09/63	...
Martin	Marie	05/06/78	...
Durand	Gaston	15/11/80	...
Titgoutte	Justine	28/02/75	...
Dupont	Noémie	18/09/57	...
Dupont	Albert	23/05/33	...

Problème : Comment distinguer les Dupont ?

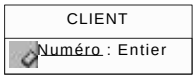
Identifiants

Solution : Ajouter un attribut *Numéro de client* !

Numéro	*Nom*	*Prénom*	*Date de Naissance*
1	Dupont	Albert	01/06/70
2	West	James	03/09/63
3	Martin	Marie	05/06/78
4	Durand	Gaston	05/11/80
5	Titgoutte	Justine	28/02/75
6	Dupont	Noémie	18/09/57
7	Dupont	Albert	23/05/33

Identifiants

- Le numéro de client est un **identifiant**. Un identifiant caractérise *de façon unique* les instances d'une classe.
- NB : Dans le paradigme objet, un objet est *déjà identifié* par son OID (*Object IDentifier*). La finalité de notre utilisation d'UML n'étant pas OO, nous ajouterons un attribut identifiant.
- **Convention graphique :**
  NB : Ne pas confondre avec les attributs de classe UML dont c'est la notation usuelle



Associations et multiplicité

Association : liaison perçue entre des classes

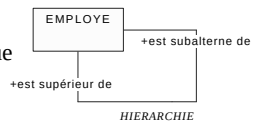
ex. Les clients commandent des produits.

- Les classes CLIENT et PRODUIT peuvent être qualifiées de *participantes* à l'association COMMANDE.
- *Degré* ou *arité* d'une association = nombre de classes participantes. En général : *associations binaires* (de degré 2).

Associations et multiplicité

Associations récursives et rôles

- *Association récursive :* une même instance de classe peut jouer plusieurs rôles dans la même association (ex. employés et supérieurs hiérarchiques).
- *Rôle :* fonction de chaque classe participante (+).



Associations et multiplicité

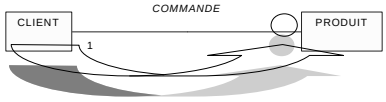
Multiplicité (ou cardinalité)

- **Définition :** Indicateur qui montre combien d'instances de la classe considérée peuvent être liées à une instance de l'autre classe participant à l'association
  - 1 Un et un seul
  - 0..1 Zéro ou un
  - 0..* ou * Zéro ou plus
  - 1..* Un ou plus
  - M..N De M à N (M, N entiers)

Associations et multiplicité

Association « 1-1 »

ex. Un client donné ne commande qu'un seul produit. Un produit donné n'est commandé que par un seul client.



Lire : Un client commande *multiplicité* (1) produit(s).

Associations et multiplicité

Association « 1-N »

ex. Un client donné commande plusieurs produits. Un produit donné n'est commandé que par un seul client.

La multiplicité « un à plusieurs » (1..*) peut aussi être « zéro à plusieurs » (0..* ou *).

Associations et multiplicité

Association « 0 ou 1-N »

ex. Un client donné commande plusieurs produits. Un produit donné est commandé au maximum par un client, mais peut ne pas être commandé.

La multiplicité « un à plusieurs » (1..*) peut aussi être « zéro à plusieurs » (0..* ou *).

Associations et multiplicité

Association « M-N »

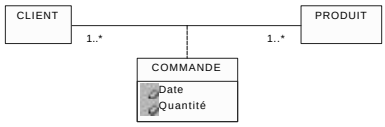
ex. Un client donné commande plusieurs produits. Un produit donné est commandé par plusieurs clients.

Les multiplicités « un à plusieurs » (1..*) peuvent aussi être « zéro à plusieurs » (0..* ou *).

Associations et multiplicité

Classe-association : Dans une *association M-N*, il est possible de caractériser l'association par des attributs (qui doivent appartenir à une classe).

ex. Une commande est passée à une Date donnée et concerne une Quantité de produit fixée.



Exemple VPC complet

Spécifications

- Les clients sont caractérisés par un numéro de client, leur nom, prénom, date de naissance, rue, code postal et ville.
- Ils commandent des produits à une date donnée et dans une quantité donnée.
- Les produits sont caractérisés par un numéro de produit, leur désignation et leur prix unitaire.

Exemple VPC complet

Spécifications (suite)

- Chaque produit est fourni par un fournisseur unique (mais un fournisseur peut fournir plusieurs produits).
- Les fournisseurs sont caractérisés par un numéro de fournisseur et leur raison sociale.

Exemple VPC complet

Marche à suivre pour produire un diagramme de classes UML :

1) Identifier les classes.
2) Identifier les associations entre les classes.
3) Identifier les attributs de chaque classe et de chaque classe d'association.
4) Evaluer la multiplicité des associations.

Exemple VPC complet

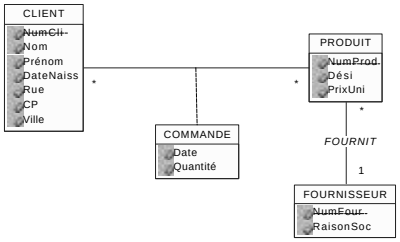


Diagramme de classes UML de l'exemple VPC

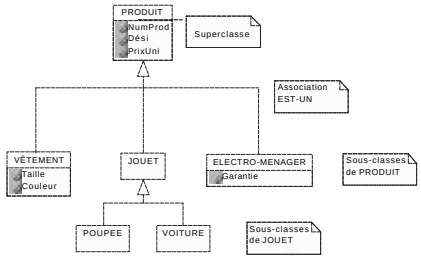
Relation de généralisation

- **Définition :** Relation de classification entre un élément général et un élément spécifique
- *Sous-classe :* classe dont les instances constituent un sous-ensemble d'une autre classe appelée *superclasse*
- *Association EST-UN (IS-A) :* association liant sous-classe et superclasse (*généralisation* ou *héritage*)

Relation de généralisation

- Une sous-classe possède tous les attributs de sa superclasse. On dit qu'elle *hérite* de ces attributs.
- Une sous-classe participe aussi à toutes les associations auxquelles participe sa superclasse.
- Une sous-classe peut posséder en plus des attributs hérités des attributs spécifiques et intervenir dans des associations spécifiques.
- Une sous-classe peut elle-même être superclasse d'une autre classe ⇒ *hiérarchie d'héritage*. Une sous-classe hérite de tous ses ascendants

Relation de généralisation



Exemple de hiérarchie d'héritage

Relation de généralisation

Spécialisation-généralisation

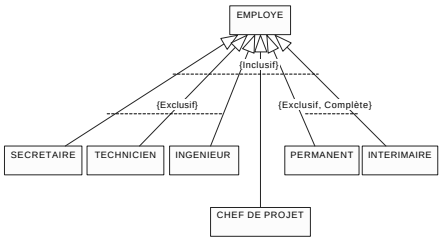
- Soit G une classe.
- Soit un ensemble Z = $\{S_1, S_2, \ldots, S_n\}$ de sous-classes de la classe G.
- G est appelée *généralisation* des sous-classes $\{S_1, S_2, \ldots, S_n\}$. Z est la *spécialisation* de G.

Relation de généralisation

Spécialisation-généralisation

- Deux types de contraintes peuvent être appliquées aux relations de généralisation.
  - `{Exclusif}` / `{Inclusif}` : un objet est au plus une instance d'une des sous-classes / ou non
  - `{Complète}` / `{Incomplète}` : il n'est pas possible de rajouter des sous-classes / c'est possible
- Les contraintes de couverture et de disjonction sont indépendantes ⇒ 4 types de spécialisations

Relation de généralisation



Diverses spécialisations d'une même classe

Plan du cours

I. Introduction

II. Modèle conceptuel UML

☞ **III. Modèle relationnel**

IV. Langage de requête SQL

V. Langage PL/SQL d'Oracle

Généralités

- Le modèle relationnel est un *modèle logique* associé aux SGBD relationnels (ex. Oracle, DB2, SQLServer, Access, Paradox, dBase…).
- **Objectifs du modèle relationnel :**
  - indépendance physique
  - traitement du problème de redondance des données
  - LMD non procéduraux (faciles à utiliser)
  - devenir un standard

Généralités

Caractéristiques des systèmes relationnels :

- Langages d'interrogation puissants et déclaratifs
- Accès orienté *valeur*
- Grande simplicité, absence de considérations physiques
- Description du schéma très réduite
- LDD intégré au LMD
- Grande dynamique de structure
- Optimisation de requêtes
- Utilisation interactive ou à partir d'un langage hôte

Relations, attributs et n-uplets

- Une *relation* R est un ensemble d'*attributs* $\{A_1, A_2, \ldots, A_n\}$.
  ex. La relation PRODUIT est l'ensemble des attributs {NumProd, Dési, PrixUni}
- Chaque attribut A_i prend ses valeurs dans un *domaine* $dom(A_i)$.
  ex. PrixUni ∈]0, 10.000]

Relations, attributs et n-uplets

- Un *n-uplet* t est un ensemble de valeurs $t = <V_1, V_2, \ldots, V_n>$ où $V_i \in dom(A_i)$ ou V_i est la valeur nulle (NULL).
  ex. <112, 'Raquette de tennis', 300> est un n-uplet de la relation PRODUIT.
- **Notation :** R $(A_1, A_2, \ldots, A_n)$
  ex. PRODUIT (NumProd, Dési, PrixUni)

Contraintes d'intégrité

- **Clé primaire :** Ensemble d'attributs dont les valeurs permettent de distinguer les n-uplets les uns des autres (*identifiant*).
  ex. NumProd clé primaire de la relation PRODUIT
- **Clé étrangère :** Attribut qui est clé primaire d'une autre relation.
  ex. Connaître le fournisseur de chaque produit ⇒ ajout de l'attribut NumFour à la relation PRODUIT

Contraintes d'intégrité

Notations : clés primaires soulignées, clés étrangères *en italiques*
ex. PRODUIT (NumProd, Dési, PrixUni, *NumFour*)

- **Contraintes de domaine :** les attributs doivent respecter une condition logique
  ex. PrixUni > 0 ET PrixUni ≤ 10000

Traduction UML-relationnel

1) Chaque classe devient une relation. Les attributs de la classe deviennent attributs de la relation. L'identifiant de la classe devient clé primaire de la relation.

ex. CLIENT (NumCli, Nom, Prénom, DateNaiss, Rue, CP, Ville)

Traduction UML-relationnel

2) Chaque association 1-1 est prise en compte en incluant la clé primaire d'une des relations comme clé étrangère dans l'autre relation.

ex. Si un client peut posséder un compte, on aura :
COMPTE (NumCom, Solde)
CLIENT (NumCli, Nom, Prénom, DateNaiss, Rue, CP, Ville, *NumCom*)

Traduction UML-relationnel

3) Chaque association 1-N est prise en compte en incluant la clé primaire de la relation dont la multiplicité maximale est * comme clé étrangère dans l'autre relation.

ex.
PRODUIT (NumProd, Dési, PrixUni, *NumFour*)
FOURNISSEUR (NumFour, RaisonSoc)

Traduction UML-relationnel

4) Chaque association M-N est prise en compte en créant une nouvelle relation dont la clé primaire et la concaténation des clés primaires des relations participantes. Les attributs de la classe d'association sont insérés dans cette nouvelle relation si nécessaire.

ex. COMMANDE (*NumCli, NumProd*, Date, Quantité)

Traduction UML-relationnel

Schéma relationnel complet de l'exemple VPC

CLIENT (NumCli, Nom, Prénom, DateNaiss, Rue, CP, Ville)
PRODUIT (NumProd, Dési, PrixUni, *NumFour*)
FOURNISSEUR (NumFour, RaisonSoc)

COMMANDE (*NumCli, NumProd*, Date, Quantité)

Traduction UML-relationnel

Héritage :

- Les sous-classes ont même clé primaire que la superclasse.
- Un attribut "type" est ajouté à la superclasse.

Exemple :

EMPLOYE (NumEmp, A1, …, Type)
Type ∈ {Secrétaire, Technicien, Ingénieur}
SECRETAIRE (*NumEmp*, A11, …)
TECHNICIEN (*NumEmp*, A21, …)
INGENIEUR (*NumEmp*, A31, …)

Problème de la redondance

[En dehors des clés étrangères]

ex. Soit la relation COMMANDE_PRODUIT.

NumProd	*Quantité*	*NumFour*	*Adresse*
101	300	901	Quai des brumes
104	1000	902	Quai Claude Bernard
112	78	904	Quai des Marans
103	250	901	Quai des brumes

Cette relation présente différentes anomalies.

Problème de la redondance

- **Anomalies de modification :** Si l'on souhaite mettre à jour l'adresse d'un fournisseur, il faut le faire pour tous les n-uplets concernés.
- **Anomalies d'insertion :** Pour ajouter un fournisseur nouveau, il faut obligatoirement fournir des valeurs pour *NumProd* et *Quantité*.
- **Anomalies de suppression :** ex. La suppression du produit 104 fait perdre toutes les informations concernant le fournisseur 902.

Normalisation

Objectifs de la normalisation :

- Suppression des problèmes de mise à jour
- Minimisation de l'espace de stockage (élimination des redondances)
  La taille des fichiers normalisés croît de façon arithmétique alors que la taille des fichiers non normalisés croît de façon géométrique.

Normalisation

Les dépendances fonctionnelles (DF)

Soit R (X, Y, Z) une relation où X, Y, et Z sont des ensembles d'attributs. Z peut être vide.

Définition : *Y dépend fonctionnellement de X* ($X \rightarrow Y$) si c'est toujours la même valeur de Y qui est associée à X dans la relation R.

ex. PRODUIT (NumProd, Dési, PrixUni)
NumProd $\rightarrow$ Dési, Dési $\rightarrow$ PrixUni

Normalisation

Propriétés des dépendances fonctionnelles

- *Réflexivité :* Si $Y \subseteq X$ alors $X \rightarrow Y$.
- *Augmentation :* Si $W \subseteq Z$ et $X \rightarrow Y$ alors $X, Z \rightarrow Y, W$.
- *Transitivité :* Si $X \rightarrow Y$ et $Y \rightarrow Z$ alors $X \rightarrow Z$.

(Règles d'inférence d'Armstrong)

Normalisation

Propriétés des dépendances fonctionnelles

- *Pseudo-transitivité :* Si $X \rightarrow Y$ et $Y, Z \rightarrow T$ alors $X, Z \rightarrow T$.
- *Union :* Si $X \rightarrow Y$ et $X \rightarrow Z$ alors $X \rightarrow Y, Z$.
- *Décomposition :* Si $Z \subseteq Y$ et $X \rightarrow Y$ alors $X \rightarrow Z$.

NB : La notation X, Y signifie $X \cup Y$.

Normalisation

Première forme normale (1FN)

Une relation est en 1FN si tout attribut n'est pas décomposable.

ex. Les relations PERSONNE (Nom, Prénoms, Age) et DEPARTEMENT (Nom, Adresse, Tel) ne sont pas en 1FN si les attributs *Prénoms* et *Adresse* peuvent être du type [Jean, Paul] ou [Rue de Marseille, Lyon].

Normalisation

Deuxième forme normale (2FN)

Une relation est en 2FN si :
- elle est en 1FN ;
- tout attribut non clé primaire est dépendant de la clé primaire entière.

ex. La relation CLIENT (NumCli, Nom, Prénom, DateNaiss, Rue, CP, Ville) est en 2FN.

Normalisation

Deuxième forme normale (2FN)

ex. La relation COMMANDE_PRODUIT (NumProd, Quantité, NumFour, Ville) n'est pas en 2FN car on a NumProd, NumFour → Quantité et NumFour → Ville.

La décomposition suivante donne deux relations en 2FN : COMMANDE (NumProd, NumFour, Quantité) ; FOURNISSEUR (NumFour, Ville).

Normalisation

Troisième forme normale (3FN)

Une relation est en 3FN si :
- elle est en 2FN ;
- il n'existe aucune DF entre deux attributs non clé primaire.

ex. La relation CINEMA (NoFilm, NoRéal, Nom) avec les DF NoFilm → NoRéal, NoRéal → Nom et NoFilm → Nom est en 2FN, mais pas en 3FN.

Normalisation

Troisième forme normale (3FN)

Anomalies de mise à jour sur la relation CINEMA : il n'est pas possible d'introduire un nouveau réalisateur sans préciser le film correspondant.

La décomposition suivante donne deux relations en 3FN qui permettent de retrouver (par transitivité) toutes les DF : R1 (NoFilm, NoRéal)
R2 (NoRéal, Nom).

Algèbre relationnelle

- Ensemble d'opérateurs qui s'appliquent aux relations
- Résultat : nouvelle relation qui peut à son tour être manipulée

⇒ L'algèbre relationnelle permet d'effectuer des recherches dans les relations.

Algèbre relationnelle

Opérateurs ensemblistes

- *Union :* T = R ∪ S ou T = UNION (R, S)
  R et S doivent avoir même schéma.

  ex. R et S sont les relations PRODUIT de deux sociétés qui fusionnent et veulent unifier leur catalogue.

  Notation graphique :

Algèbre relationnelle

Opérateurs ensemblistes

- *Intersection :* T = R ∩ S ou
  T = INTERSECT (R, S)
  R et S doivent avoir même schéma.

  ex. Permet de trouver les produits communs aux catalogues de deux sociétés.

  Notation graphique :

Algèbre relationnelle

Opérateurs ensemblistes

- *Différence :* T = R - S ou T = MINUS (R, S)
  R et S doivent avoir même schéma.

  ex. Permet de retirer les produits de la relation S existant dans la relation R.

  Notation graphique :

Algèbre relationnelle

Opérateurs ensemblistes

- *Produit cartésien :* T = R x S
  ou T = PRODUCT (R, S)

  Associe chaque n-uplet de R à chaque n-uplet de S.

  Notation graphique :

Algèbre relationnelle

ex.

NumProd	*Dési*
0	P1
1	P2

X

NumFour	*RaisonSoc*
10	F1
20	F2
30	F3

=

NumProd	*Dési*	*NumFour*	*RaisonSoc*
0	P1	10	F1
1	P2	10	F1
0	P1	20	F2
1	P2	20	F2
0	P1	30	F3
1	P2	30	F3

Algèbre relationnelle

Opérateurs ensemblistes

- *Division :* T = R ÷ S ou T = DIVISION (R, S)

  R (A_1, A_2, …, A_n) S (A_{p+1}, …, A_n)
  T (A_1, A_2, …, A_p) contient tous les n-uplets tels que leur concaténation à *chacun* des n-uplets de S donne toujours un n-uplet de R.

  Notation graphique :

Algèbre relationnelle

ex.

NumCli	*Date*	*Quantité*
1	22/09/99	1
1	22/09/99	5
1	10/10/99	2
2	15/10/99	9
3	22/09/99	1
3	10/10/99	2

÷

Date	*Quantité*
22/09/99	1
10/10/99	2

=

NumCli
1
3

Algèbre relationnelle

Opérateurs spécifiques

- *Projection :* T = Π <A, B, C> (R)
  ou T = PROJECT (R / A, B, C)

  T ne contient que les attributs A, B et C de R.

  ex. Noms et prénoms des clients.

  Notation graphique :

Algèbre relationnelle

Opérateurs spécifiques

- *Restriction :* T = σ <C> (R)
  ou T = RESTRICT (R / C)

  T ne contient que les attributs de R qui satisfont la condition C.

  ex. Clients qui habitent Lyon (C : Ville = 'Lyon')

  Notation graphique :

Algèbre relationnelle

Opérateurs spécifiques

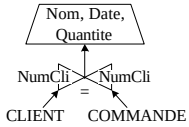
- *Jointure naturelle :* T = R >< S
  ou T = JOIN (R, S)

  Produit cartésien R x S et restriction A = B sur les attributs A ∈ R et B ∈ S.

  Notation graphique :

Algèbre relationnelle

ex. Commandes avec le nom du client et pas seulement son numéro



NB : *Requête* = enchaînement d'opérations

Algèbre relationnelle

Décomposition des opérations

CL

NumCli	*Nom*
1	C1
2	C2
3	C3

X

CO

NumCli	*Date*	*Quantité*
1	22/09/99	1
3	22/09/99	5
3	22/09/99	2

Algèbre relationnelle

=

CL.NumCli	*Nom*	*CO.NumCli*	*Date*	*Quantité*
1	C1	1	22/09/99	1
2	C2	1	22/09/99	1
3	C3	1	22/09/99	1
1	C1	3	22/09/99	5
2	C2	3	22/09/99	5
3	C3	3	22/09/99	5
1	C1	3	22/09/99	2
2	C2	3	22/09/99	2
3	C3	3	22/09/99	2

Algèbre relationnelle

CL >< CO

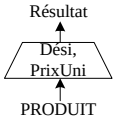
CL.NumCli	*Nom*	*CO.NumCli*	*Date*	*Quantité*
1	C1	1	22/09/99	1
3	C3	3	22/09/99	5
3	C3	3	22/09/99	2

Π <Nom, Date, Quantité> (CL >< CO)

Nom	*Date*	*Quantité*
C1	22/09/99	1
C3	22/09/99	5
C3	22/09/99	2

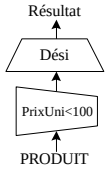
Exemples de requêtes

Ex. 1 : *Désignation et prix unitaire de tous les produits*



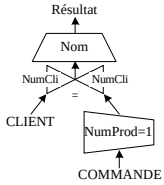
Exemples de requêtes

Ex. 2 : *Désignation des produits de prix inférieur à 100 €*



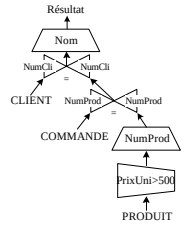
Exemples de requêtes

Ex. 3 : *Nom des clients qui ont commandé le produit n° 1*



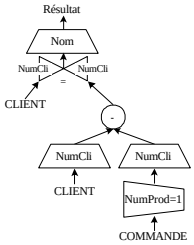
Exemples de requêtes

Ex. 4 : *Nom des clients qui ont commandé au moins un produit de prix supérieur à 500 €*



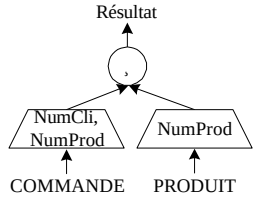
Exemples de requêtes

Ex. 5 : *Nom des clients qui n'ont pas commandé le produit n° 1*



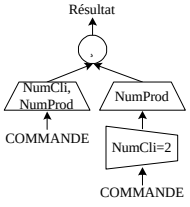
Exemples de requêtes

Ex. 6 : *Numéro des clients qui ont commandé tous les produits*



Exemples de requêtes

Ex. 7 : *Numéro des clients qui ont commandé tous les produits commandés par le client n° 2*



Classification des SGBD relationnels

- Niveau 1 : *Systèmes non relationnels*. Supportent uniquement la structure tabulaire.
- Niveau 2 : *Systèmes relationnellement minimaux*. Permettent les opérations de sélection, projection et jointure.
- Niveau 3 : *Systèmes relationnellement complets*. Toutes les opérations de l'algèbre relationnelle.
- Niveau 4 : *Systèmes relationnellement pleins*. Permettent la définition des contraintes d'intégrité.

Plan du cours

I. Introduction

II. Modèle conceptuel UML

III. Modèle relationnel

☞ **IV. Langage de requête SQL**

V. Langage PL/SQL d'Oracle

Généralités

- SQL = *Structured Query Language*, issu de SEQUEL (*Structured English as a QUery Language*)
- SQL permet la définition, la manipulation et le contrôle d'une base de données relationnelle. Il se base sur l'algèbre relationnelle.
- SQL est un standard ANSI depuis 1986.
- Nous adoptons dans ce chapitre la syntaxe du SQL d'Oracle (très proche de la norme).

Définition des données

Définitions des tables (relations)

```
CREATE TABLE NomTable (   Attribut1 TYPE,
                          Attribut2 TYPE, …,
                          contrainte_integrité1,
                          contrainte_integrité2, …);
```

Type des données principaux :

- NUMBER(n) : Entier à n chiffres
- NUMBER(n, m) : Réel à n chiffres au total (virgule comprise), m après la virgule
- VARCHAR(n) : Chaîne de n caractères
- DATE : Date au format 'JJ-MM-AAAA'

Définition des données

Définitions des contraintes d'intégrité

- *Clé primaire :*

CONSTRAINT nom_contrainte PRIMARY KEY (attribut_clé [, attribut_clé2, …])

- *Clé étrangère :*

CONSTRAINT nom_contrainte FOREIGN KEY (attribut_clé_ét) REFERENCES table(attribut)

- *Contrainte de domaine :*

CONSTRAINT nom_contrainte CHECK (condition)

Définition des données

ex. CREATE TABLE Client (NumCli NUMBER(3),
Nom VARCHAR(30),
DateNaiss DATE,
Salaire NUMBER(8,2),
NumEmp NUMBER(3),

CONSTRAINT cle_pri PRIMARY KEY (NumCli),
CONSTRAINT cle_etr FOREIGN KEY (NumEmp)
REFERENCES EMPLOYEUR(NumEmp),
CONSTRAINT date_ok CHECK (DateNaiss < SYSDATE));

Définition des données

- **Création d'index** (accélération des accès)

CREATE [UNIQUE] INDEX nom_index ON nom_table (attribut [ASC|DESC], …);

UNIQUE ⇒ pas de double

ASC/DESC ⇒ ordre croissant ou décroissant

ex. CREATE UNIQUE INDEX Icli ON Client (Nom);

- **Destructions :** DROP TABLE nom_table;
DROP INDEX nom_index;

Définition des données

- **Ajout d'attributs**
  ALTER TABLE nom_table ADD (attribut TYPE, …);
  ex. ALTER TABLE Client ADD (tel NUMBER(8));
- **Modifications d'attributs**
  ALTER TABLE nom_table MODIFY (attribut TYPE, …);
  ex. ALTER TABLE Client MODIFY (tel NUMBER(10));
- **Suppression d'attributs**
  ALTER TABLE nom_table DROP COLUMN attribut, ...;
  ex. ALTER TABLE Client DROP COLUMN tel;

Définition des données

- **Ajout de contrainte**
  ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte définition_contrainte;
  ex. ALTER TABLE Client
  ADD CONSTRAINT sal_ok CHECK (salaire>0);
- **Suppression de contrainte**
  ALTER TABLE nom_table DROP CONSTRAINT nom_contrainte;
  ex. ALTER TABLE Client
  DROP CONSTRAINT sal_ok;

Mise à jour des données

- **Ajout d'un n-uplet**
  INSERT INTO nom_table
  VALUES (val_att1, val_att2, …);
  ex. INSERT INTO Produit
  VALUES (400, 'Nouveau produit', 78.90);
- **Mise à jour d'un attribut**
  UPDATE nom_table SET attribut=valeur
  [WHERE condition];
  ex. UPDATE Client SET Nom='Dudule'
  WHERE NumCli = 3;

Mise à jour des données

- **Suppression de n-uplets**

  DELETE FROM nom_table [WHERE condition];

  ex. DELETE FROM Produit;

  DELETE FROM Client
  WHERE Ville = 'Lyon';

Interrogation des données

Forme générale d'une requête :

```
SELECT [ALL|DISTINCT] attribut(s) FROM table(s)
[WHERE condition]
[GROUP BY attribut(s) [HAVING condition]]
[ORDER BY attribut(s) [ASC|DESC]];
```

- **Tous les n-uplets d'une table**
  ex. SELECT * FROM Client;
- **Tri du résultat**
  ex. *Par ordre alphabétique inverse de nom*
  SELECT * FROM Client
  ORDER BY Nom DESC;

Ou… par l'exemple

Interrogation des données

- **Calculs** ex. *Calcul de prix TTC*
  SELECT PrixUni+PrixUni*0.206 FROM Produit;
- **Projection**
  ex. *Noms et Prénoms des clients, uniquement*
  SELECT Nom, Prenom FROM Client;
- **Restriction**
  ex. *Clients qui habitent à Lyon*
  SELECT * FROM Client
  WHERE Ville = 'Lyon';

Interrogation des données

ex. *Commandes en quantité au moins égale à 3*

```
SELECT * FROM Commande
WHERE Quantite >= 3;
```

ex. *Produits dont le prix est compris entre 50 et 100 €*

```
SELECT * FROM Produit
WHERE PrixUni BETWEEN 50 AND 100;
```

ex. *Commandes en quantité indéterminée*

```
SELECT * FROM Commande
WHERE Quantite IS NULL;
```


Interrogation des données

ex. *Clients habitant une ville dont le nom se termine par* sur-Saône

```
SELECT * FROM Client
WHERE Ville LIKE '%sur-Saône';
```

'sur-Saône%' ⇒ commence par *sur-Saône*
'%sur%' ⇒ contient le mot *sur*

Interrogation des données

ex. *Prénoms des clients dont le nom est Dupont, Durand ou Martin*

```
SELECT Prenom FROM Client
WHERE Nom IN ('Dupont', 'Durand', 'Martin');
```

NB : Possibilité d'utiliser la négation pour tous ces prédicats : NOT BETWEEN, NOT NULL, NOT LIKE, NOT IN.

Interrogation des données

- **Fonctions d'agrégat**
  Elles opèrent sur un ensemble de valeurs.
  - AVG(), VARIANCE(), STDDEV() : moyenne, variance, écart-type des valeurs
  - SUM() : somme des valeurs
  - MIN(), MAX() : valeur minimum, valeur maximum
  - COUNT() : nombre de valeurs

ex. *Moyenne des prix des produits*

```
SELECT AVG(PrixUni) FROM Produit;
```


Interrogation des données

Opérateur DISTINCT

ex. *Nombre total de commandes*

```
SELECT COUNT(*) FROM Commande;
SELECT COUNT(NumCli) FROM Commande;
```

ex. *Nombre de clients ayant passé commande*

```
SELECT COUNT( DISTINCT NumCli)
FROM Commande;
```


Interrogation des données

Table COMMANDE (simplifiée)

NumCli	*Date*	*Quantite*
1	22/09/99	1
3	22/09/99	5
3	22/09/99	2

COUNT(NumCli) ⇒ Résultat = 3

COUNT(DISTINCT NumCli) ⇒ Résultat = 2

Interrogation des données

- **Jointure**

ex. *Liste des commandes avec le nom des clients*

```
SELECT Nom, Date, Quantite
FROM Client, Commande
WHERE   Client.NumCli =
        Commande.NumCli;
```


Interrogation des données

ex. *Idem avec le numéro de client en plus*

```
SELECT C1.NumCli, Nom, Date, Quantite
FROM Client C1, Commande C2
WHERE C1.NumCli = C2.NumCli
ORDER BY Nom;
```

NB : Utilisation d'*alias* (C1 et C2) pour alléger l'écriture + tri par nom.

Interrogation des données

Jointure exprimée avec le prédicat IN

ex. *Nom des clients qui ont commandé le 23/09*

```
SELECT Nom FROM Client
WHERE NumCli IN (
    SELECT NumCli FROM Commande
    WHERE Date = '23-09-1999'     );
```

NB : Il est possible d'imbriquer des requêtes.

Interrogation des données

- **Prédicats EXISTS / NOT EXISTS**

ex. *Clients qui ont passé au moins une commande [n 'ont passé aucune commande]*

```
SELECT * FROM Client C1
WHERE [NOT] EXISTS (
    SELECT * FROM Commande C2
    WHERE C1.NumCli = C2.NumCli     );
```


Interrogation des données

- **Prédicats ALL / ANY**

ex. *Numéros des clients qui ont commandé au moins un produit en quantité supérieure à chacune [à au moins une] des quantités commandées par le client n° 1.*

```
SELECT DISTINCT NumCli FROM Commande
WHERE Quantite > ALL [ANY] (
    SELECT Quantite FROM Commande
    WHERE NumCli = 1     );
```


Interrogation des données

- **Groupement**

ex. *Quantité totale commandée par chaque client*

```
SELECT NumCli, SUM(Quantite)
FROM Commande
GROUP BY NumCli;
```

ex. *Nombre de produits différents commandés...*

```
SELECT NumCli, COUNT(DISTINCT NumProd)
FROM Commande
GROUP BY NumCli;
```

Interrogation des données

ex. *Quantité moyenne commandée pour les produits faisant l'objet de plus de 3 commandes*

```
SELECT NumProd, AVG(Quantite)
FROM Commande
GROUP BY NumProd
HAVING COUNT(*)>3;
```

Attention : La clause HAVING ne s'utilise qu'avec GROUP BY.

Interrogation des données

- **Opérations ensemblistes**

  INTERSECT, MINUS, UNION

  ex. *Numéro des produits qui soit ont un prix inférieur à 100 €, soit ont été commandés par le client n° 2*

```
SELECT NumProd FROM Produit WHERE PrixUni<100
UNION
SELECT NumProd FROM Commande WHERE NumCLi=2;
```


Les vues

- **Vue** : table *virtuelle* calculée à partir d'autres tables grâce à une requête
- **Définition d'une vue**
  CREATE VIEW nom_vue AS requête;

  ex. CREATE VIEW Noms AS
  SELECT Nom, Prenom FROM Client;

Les vues

Intérêt des vues

- Simplification de l'accès aux données en masquant les opérations de jointure

  ex. CREATE VIEW Prod_com AS
  SELECT P.NumProd, Dési, PrixUni, Date, Quantite
  FROM Produit P, Commande C
  WHERE P.NumProd=C.NumProd;

  SELECT NumProd, Dési FROM Prod_com
  WHERE Quantite>10;

Les vues

Intérêt des vues

- Sauvegarde indirecte de requêtes complexes
- Présentation de mêmes données sous différentes formes adaptées aux différents usagers particuliers
- Support de l'indépendance logique
  ex. Si la table Produit est remaniée, la vue Prod_com doit être refaite, mais les requêtes qui utilisent cette vue n'ont pas à être remaniées.

Les vues

Intérêt des vues

- Renforcement de la sécurité des données par masquage des lignes et des colonnes sensibles aux usagers non habilités

Problèmes de mise à jour, restrictions

La mise à jour de données via une vue pose des problèmes et la plupart des systèmes impose d'importantes restrictions.

Les vues

Problèmes de mise à jour, restrictions de màj

- Le mot clé DISTINCT doit être absent.
- La clause FROM doit faire référence à une seule table.
- La clause SELECT doit faire référence directement aux attributs de la table concernée (pas d'attribut dérivé).
- Les clauses GROUP BY et HAVING sont interdites.

Catalogue du système

- **Catalogue du système :** Ensemble de vues maintenues automatiquement par le système et contenant sous forme relationnelle la définition de tous les objets créés par le système et les usagers.

- Ces vues sont accessibles avec SQL (en mode consultation uniquement).

Catalogue du système

- ALL_TABLES (OWNER, TABLE_NAME, …)
- ALL_VIEWS (OWNER, VIEW_NAME, …)
- ALL_CONSTRAINTS (OWNER, TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION, ...)
- ALL_CONS_COLUMNS (OWNER, TABLE_NAME, CONSTRAINT_NAME, COLUMN_NAME, ...)
- USER_CATALOG (TABLE_NAME, TABLE_TYPE)
- USER_TAB_COLUMNS (TABLE_NAME, COLUMN_NAME, …)
- USER_IND_COLUMNS (INDEX_NAME, TABLE_NAME, COLUMN_NAME, …)
- USER_CONSTRAINTS (TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION, ...)

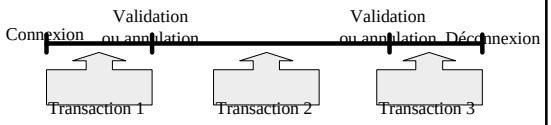
• • •

Catalogue du système

- Tables qui contiennent un attribut *Intitule*
  SELECT TABLE_NAME FROM USER_TAB_COLUMNS
  WHERE COLUMN_NAME='INTITULE';
- Attributs de la table *Client*
  SELECT COLUMN_NAME FROM USER_TAB_COLUMNS
  WHERE TABLE_NAME='CLIENT';
- Contraintes des tables de l'utilisateur *darmont*
  SELECT TABLE_NAME, CONSTRAINT_NAME
  FROM ALL_CONSTRAINTS
  WHERE OWNER='DARMONT';

Gestion des transactions

- **Transaction :** ensemble de mises à jour des données (≠ modifications structurelles)
- **Début de transaction :** début de la session de travail ou fin de la transaction précédente



Gestion des transactions

- **Validation** (et fin) d'une transaction :
  COMMIT;
- **Annulation** (et fin) d'une transaction :
  ROLLBACK;
- Fin de session de travail (avec la commande EXIT ou QUIT) ⇒ validation automatique

Sécurité et autorisation

- **Transmission de privilèges**
  GRANT privilège ON table|vue
  TO user|PUBLIC [WITH GRANT OPTION];
  Privilèges :
  SELECT : lecture INSERT : insertion
  UPDATE : mise à jour DELETE : suppression
  ALL : tous les privilèges ALTER : destruction
  INDEX : construction d'index
- **Suppression de privilèges**
  REVOKE privilège ON table|vue FROM user|PUBLIC;

Plan du cours

I. Introduction
II. Modèle conceptuel UML
III. Modèle relationnel
IV. Langage de requête SQL
☞ **V. Langage PL/SQL d'Oracle**

Généralités

- **PL/SQL :** Langage procédural de 4ème génération (L4G)
- Extension de SQL
- Déclaration de variables et de constantes
- Types de données abstraits (collections, enregistrements, objets)
- Modularité (sous-programmes, packages)
- Gestion des erreurs à l'exécution (exceptions)
- Intéraction étroite avec Oracle (avec SQL)

Généralités

Avantages :

- Support de SQL, y compris en dynamique
- Support de programmation orientée-objet
- Performance (traitements par lot)
- Productivité (uniformité dans tous les outils)
- Portabilité (sur tous systèmes Oracle)
- Intégration étroite avec Oracle (mêmes types de données que SQL, par exemple)
- Sécurité (procédures stockées, déclencheurs)

Structure d'un bloc PL/SQL

```
[DECLARE
     -- Déclaration types,
        constantes et variables]
BEGIN
     -- Instructions PL/SQL
[EXCEPTION
     -- Traitement des erreurs]
END;
```


Déclarations

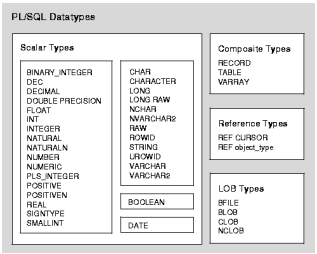


Schéma tiré de la Documentation Oracle 8 (Fig. 2-1)

Déclarations

- Partie déclarative d'un bloc PL/SQL
- **Variables**

ex.
```
date_naissance DATE;
compteur INTEGER := 0; -- Initialisation
compteur INTEGER DEFAULT 0; -- Idem
id CHAR(5) NOT NULL := 'AP001';
```

- **Constantes**

ex.
```
euro CONSTANT REAL:=6.55957;
```


Déclarations

- **Type d'une autre variable**

ex.
```
credit REAL;
debit credit%TYPE;
```

- **Type d'un attribut d'une table**

ex.
```
num_emp EMP.EMPNO%TYPE;
```

- **Type d'un n-uplet d'une table**

ex.
```
un_client client%ROWTYPE;
```


Affectation

- **Affectation simple**

ex.
```
numero := 0;
numero := numero + 1;
```

- **Valeurs issues d'une base de données**

ex.
```
SELECT numcli INTO numero
FROM client WHERE numcli=10;

SELECT empno, ename INTO num, nom
FROM emp WHERE ename='KING';
```


Expressions et comparaisons

- **Opérateurs arithmétiques :** `+ - / * **`
- **Opérateur de concaténation :** `||`
- **Opérateurs de comparaison :**
  - `= < > <= >= <>`
  - `IS NULL, LIKE, BETWEEN, IN`
- **Opérateurs logiques :** `AND, OR, NOT`

Structures de contrôle

Sélection

IF-THEN, IF-THEN-ELSE ou IF-THEN-ELSIF

```
IF condition1 THEN
   -- Instructions
[ELSEIF condition2 THEN
   -- Instructions]
[ELSE
   -- Instructions]
END IF;
```


Structures de contrôle

- **Boucle *pour***

```
FOR compteur IN [REVERSE] min..max LOOP
   -- Instructions
END LOOP;
```

- **Boucle *tant que***

```
WHILE condition LOOP
   -- Instructions
END LOOP;
```

- **Boucle "infinie"**

```
LOOP
   -- Instructions
END LOOP;
```

Structures de contrôle

- **Branchement inconditionnel**

  `GOTO étiquette;`

  ☠ **Formellement interdit** ☠

- **Branchement de sortie de boucle**

  `EXIT WHEN condition;`

  💣 **Uniquement autorisé pour sortir d'une boucle infinie** 💣

Affichage

DBMS_OUTPUT.PUT_LINE('chaîne');

```
DBMS_OUTPUT.PUT_LINE('Bonjour');
DBMS_OUTPUT.PUT_LINE('nom='||nom);
DBMS_OUTPUT.PUT_LINE('num='||TO_CHAR(num));
```

NB : Pour que l'affichage fonctionne, la variable d'environnement SERVEROUTPUT de SQL*Plus doit être à ON.

```
SET SERVEROUTPUT ON
```


Exemple 1

```
-- Affichage en francs d'un salaire stocké en euros
DECLARE
      euro CONSTANT REAL := 6.55957;
      salaire emp.sal%TYPE;
BEGIN
      -- Affectation
      SELECT sal INTO salaire FROM emp
      WHERE ename='DARMONT';
      -- Conversion
      salaire := salaire * euro;
      -- Affichage
      DBMS_OUTPUT.PUT_LINE(TO_CHAR(salaire)||' FF');
END;
```


Collections

- **Collection :** Ensemble ordonné d'éléments du même type. Chaque élément possède un indice qui détermine sa position dans la collection.
- Deux types de collections :
  - Tableau (VARRAY) : taille maximum, dense
  - Table (TABLE) : extensible, non-dense

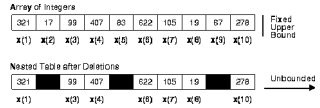


Schéma tiré de la Documentation Oracle 8 (Fig. 4-1)

Collections

1. Déclaration d'un type collection

ex. `TYPE TableChar IS TABLE OF VARCHAR(20);`
`TYPE TableauInt IS VARRAY(10) OF INTEGER;`

2. Déclaration d'une variable collection

ex. `ma_table TableChar := TableChar('Aa', 'Bb', 'Cc');`
`mon_tableau TableauInt := TableauInt();`

NB : Une collection peut être initialisée à vide.
Il n'est pas nécessaire d'initialiser tous les éléments d'un tableau.

Collections

- **Affectation de collections entières**

```
ex. DECLARE TYPE T1 IS TABLE OF INTEGER;
          TYPE T2 IS TABLE OF INTEGER;
          et11 T1 := T1(1, 2, 3, 4);
          et12 T1 := T1(5, 6);
          et2 T2 := T2();
    BEGIN et12 := et11; -- Légal
          et2 := et11; -- Illégal
```

- **Affectation d'éléments de collection**

ex. `et11(1) := 10;`

Collections

- **Méthodes** (≈ procédures) **prédéfinies**
  Utilisation : `nom_col.nom_méth[(param)]`
- `EXISTS(i)` retourne TRUE si le i^{ème} élement de la collection existe.
- `COUNT` retourne le nombre d'éléments dans la collection.
- `LIMIT` retourne la taille maximum de la collection (ou NULL pour les tables).
- `EXTEND(n)` augmente la taille de la collection de n éléments. `EXTEND` est équivalent à `EXTEND(1)`.

Collections

- `TRIM(n)` supprime n éléments en fin de collection (la taille de la collection est diminuée). `TRIM` est équivalent à `TRIM(1)`.
- `DELETE(i), DELETE(i, j), DELETE` effacent le i^{ème} élement, les éléments d'indice i à j et tous les éléments de la collection, respectivement (tables uniquement).
- `FIRST` et `LAST` retournent l'indice du premier et du dernier élément de la collection, resp.
  NB : FIRST=1 et LAST=COUNT pour un tableau.
- `PRIOR(n)` et `NEXT(n)` retournent l'indice de l'élément précédent et suivant de l'élément d'indice n, resp.

Collections

```
DECLARE                                  -- Exemple
     TYPE Liste_Entiers IS TABLE OF INTEGER;
     pile Liste_Entiers := Liste_Entiers();
     element INTEGER;
BEGIN
     -- Ajouts au sommet de la pile (empiler)
     pile.EXTEND;
     pile(pile.COUNT+1) := 1;
     pile.EXTEND;
     pile(pile.COUNT+1) := 11;
     -- Suppression du sommet (dépiler)
     element := pile(pile.COUNT); -- element=11
     pile.TRIM;
```


Enregistrements

- **Enregistrement :** Ensemble de données logiquement liées stockées dans des *champs.*

1. Déclaration d'un type enregistrement

ex.
```
TYPE Etudiant IS RECORD(
     num_etu INTEGER,
     nom VARCHAR(50),
     age INTEGER);
```

2. Déclaration d'une variable enregistrement

ex. `un_etudiant Etudiant;`

Enregistrements

- **Référencement direct**

ex.
```
un_etudiant.num_etu := 12212478;
un_etudiant.nom := 'Toto';
un_etudiant.age := 6;
```

- **Résultat de requête**

ex.
```
SELECT num_etu, nom, age
  INTO un_etudiant
  FROM ETUDIANT
  WHERE num_etu = 12212478;
```


Curseurs

- **Curseur :** Structure de données permettant de stocker le résultat d'une requête qui retourne plusieurs n-uplets (lignes).
- **Déclaration :** `CURSOR nom_curs IS requêteSQL;`

ex.
```
CURSOR calcul IS
  SELECT numprod, prixuni*1.206 prixttc
  FROM produit;
```

NB : Un n-uplet du curseur sera de type calcul%ROWTYPE.

Curseurs

```
-- Exemple de parcours complet
DECLARE
     CURSOR calcul IS
     SELECT numprod, prixuni*1.206 prixttc
     FROM produit;
     nuplet calcul%ROWTYPE;
BEGIN
     FOR n-uplet IN calcul LOOP
          DBMS_OUTPUT.PUT_LINE(
               TO_CHAR(nuplet.numprod)
               ||' : '||
               TO_CHAR(nuplet.prixttc));
     END LOOP;
```


Curseurs

```
-- Exemple de parcours personnalisé
DECLARE
     ...
BEGIN
     OPEN calcul;
     LOOP
          FETCH calcul INTO nuplet;
          EXIT WHEN calcul%NOTFOUND;
          ...
     END LOOP;
     CLOSE calcul;
END;
```


Curseurs

Attributs des curseurs :

- `%NOTFOUND` est égal à FALSE si FETCH retourne un résultat
- `%FOUND` est l'opposé logique de `%NOTFOUND`
- `%ROWCOUNT` retourne le nombre de lignes lues
- `%ISOPEN` est égal à TRUE si le curseur est ouvert

Exceptions

- À chaque erreur à l'exécution, une **exception** est **levée**. Ces exceptions sont gérées par des routines séparées.
- **Avantages**
  - Traitement systématique des erreurs
  - Traitement groupé d'erreurs similaires
  - Lisibilité du code (traitement des erreurs séparé)
- **Fonctions PL/SQL pour la gestion d'erreurs**
  - `SQLCODE` : Code de la dernière exception levée
  - `SQLERRM` : Message d'erreur associé

Exceptions

- **Déclaration** (section `DECLARE`)
  `nom_exception EXCEPTION;`
- **Lever l'exception** (section `BEGIN`)
  ```
  IF condition THEN
     RAISE nom_exception;
  END IF;
  ```
- **Traitement de l'exception** (section `EXCEPTION`)
  `WHEN nom_exception THEN ...;`

Exemple 2

```
-- Calcul du prix TTC des produits
-- et recopie dans la table PRODTTC


DECLARE

      nbp NUMBER(3);
      aucun_produit EXCEPTION;

      CURSOR calcul IS
            SELECT numprod, prixuni*1.206 prixttc
            FROM produit;

      nuplet calcul%ROWTYPE;
```

Exemple 2

```
BEGIN

      -- Comptage des produits
      SELECT COUNT(*) INTO nbp FROM produit;

      -- Test « il existe des produits » ou pas ?
      IF nbp = 0 THEN
            RAISE aucun_produit;
      END IF;
```


Exemple 2

```
      -- Recopie des valeurs dans la table prodttc
      FOR nuplet IN calcul LOOP
            INSERT INTO prodttc VALUES
                  (nuplet.numprod, nuplet.prixttc);
      END LOOP;
      -- Validation de la transaction
      COMMIT;

EXCEPTION
      WHEN aucun_produit THEN
            RAISE_APPLICATION_ERROR(-20501,
            'Erreur : table client vide');

END;
```


Sous-programmes

```
-- Définition de procédure

PROCEDURE nomp (param1, param2…) IS
   -- Déclarations locales
BEGIN
   -- Instructions
[EXCEPTION
   -- Traitement des exceptions]
END;
```


Sous-programmes

```
-- Définition de fonction

FUNCTION nomf (param1, param2…)
RETURN type_valeur_de_retour IS
   -- Déclarations locales
BEGIN
   -- Instructions
   RETURN valeur_de_retour;
[EXCEPTION
   -- Traitement des exceptions]
END;
```


Sous-programmes

- **Déclaration :** Tout sous-programme doit être défini avant d'être appelé ⇒ définition dans la section déclaration d'un bloc PL/SQL
- **Définition des paramètres, modes de passage**
  `nom_param [IN|OUT|IN OUT] TYPE`
  ex. `resultat OUT REAL`
  - IN : Paramètre en entrée
  - OUT : Paramètre en sortie
  - IN OUT : Paramètre en entrée-sortie

Sous-programmes

```
-- Exemple de procédure

PROCEDURE Conversion_FF_euro (prixF IN REAL,
                              prixE OUT REAL) IS
      euro CONSTANT REAL := 6.55957;
      prixnull EXCEPTION;
BEGIN
      IF prixF IS NOT NULL THEN
            prixE := prixF/euro;
      ELSE
            RAISE prixnull;
      END IF;
EXCEPTION
      WHEN prixnull THEN
      DBMS_OUTPUT.PUT_LINE('Calcul impossible');
END;
```

Sous-programmes

```
-- Exemple de fonction (récursive, calcul de n!)

FUNCTION facto (n INTEGER) RETURN INTEGER IS

BEGIN
     IF n = 1 THEN -- Condition de terminaison
          RETURN 1;
     ELSE -- Appel récursif
          RETURN n * facto(n - 1);
     END IF;
END;
```