Chapitres du cours

- Chapitre 1 : Introduction aux BD et SGBD
- **Chapitre 2 : Le langage SQL**
- Chapitre 3 : Bases du langage PL/SQL

Chapitre 2

Le Langage SQL

Plan du chapitre 2

- Le Langage de Définition des Données (LDD)
- Le Langage de Manipulation des Données (LMD)
- Le Langage d'Interrogation des Donnés (LID)

SQL ? (2)

❑ Sous langages SQL

- **LDD** : **L**angage de **D**éfinition des **D**onnées
  - ✓ Création, modification, suppression du schéma de la BD, des composants du schéma de la BD (colonne, contrainte etc.)

    `CREATE, ALTER, RENAME, DROP`
- **LMD** : **L**angage de **M**anipulation des **D**onnées
  - ✓ Manipulation des données : ajout, suppression, modification

    `INSERT, UPDATE, DELETE`
- **LID** : **L**angage d'**I**nterrogation des **D**onnées
  - ✓ Sélection de données

    `INSERT`
- **LCD** : **L**angage de **C**ontrôle des **D**onnées
  - ✓ Contrôle de l'accès aux données : droits d'accès, vues etc.

    `GRANT, REVOKE`
- **LCT : L**angage **de C**ontrôle **des T**ransactions
  - ✓ Validation et annulation des transactions

    `COMMIT, ROLLBACK`

Objets SQL

- **Identificateurs**
  - Chaîne de 30 caractères au plus : lettres, chiffres, caractères spéciaux, commence par une lettre
  - Identificateur ≠ mot clé
- **Tables**
  - Implémentation des relations, associées à un schéma
- **Colonnes**
  - Implémentation de l'attribut d'une relation
  - Les données d'une même colonne sont de même type
  - Des colonnes de différentes tables peuvent porter le même nom

Types des colonnes

- **Numérique**
- **Chaîne de caractères**
- **Temporels**
- **Binaire**

Type numérique

- **TYPE NUMERIQUE / SQL2**
  - Nombres entiers : -32768..32767
  - TINYINT : 1octet, 0..255
  - SMALLINT : 2 octets
  - Integer : 4 octets, -2147483648..2147483647
  - Nombres décimaux
  - Nombres réels : DOUBLE PRECISION, FLOAT

- **TYPE NUMERIQUE / ORACLE**
  - 1 seul type numérique : NUMBER
  - NUMBER
  - NUMBER(Taille-Max)
  - NUMBER (Taille-Max, décimales)

Type chaîne de caractères

TYPE CHAINE DE CARACTERES / SQL2

- Constantes chaînes de caractères entourée par l'apostrophe (‘), doublée dans le cas où la chaîne contient elle-même l'apostrophe
- CHAR (*Longueur*)
  - Chaînes de caractères de longueur fixe (<4000 car)
  - *Longueur* est la longueur maximale de la chaîne
  - Toute constante de longueur inférieure à *Longueur* sera complétée par des caractères blancs
- VARCHAR2(*Longueur*)
  - Chaînes de caractères de longueur variable (<4000 car)
  - *Longueur* est la longueur maximale de la chaîne

TYPE CHAINE DE CARACTERES / ORACLE

- CHAR et VARCHAR2 (équivalent à VARCHAR, taille maximale de 4000 car)

Type temporel

- **TYPES TEMPORELS / SQL2**
  - DATE : 2 chiffres mm, 2 chiffres JJ, 4 chiffres AAAA
  - TIME : 2 chiffres hh, 2 chiffres mm, 2 chiffres ss
  - TIMESTAMP : moment précis (hh, mm, ss)

- **TYPES TEMPORELS / ORACLE**
  - DATE : seul type
  - Valeur sous forme de chaîne de caractères entre apostrophes 'JJ/MM/AA'
  - Le format de la date peut être positionné : SET DATE TO <Format>

Type BINAIRE

- **TYPES BINAIRE**
  - Type non normalisé SQL2
  - Données son, image
  - Type précis dépendant du SGBD
    - ORACLE : LONG RAW
    - SYBASE : IMAGE
    - INFORMIX : BYTE
    - …

Les types largement utilisés

VARCHAR2(size)	Chaine de caractères de longueur variable de taille comprise entre 1 et 4000 caractères (octets)
CHAR (size)	Chaine de caractères de longueur fixe de taille comprise entre 1 et 4000 caractères (octets)
NUMBER(t[,p])	Numérique de taille `t` et précision `p`
DATE	Date valide, champs : `day, month,hour, minute` et `second`
CLOB	Chaine de caractères de longueur variable
BLOB	Chaîne binaire

Définition des données

- Création de tables
- Définition de contraintes d'intégrité
- Mise à jour de la structure de tables
- Suppression de tables

Création de tables (1)

- **Création de tables sans données :** crée le schéma physique d'une **table vierge** avec une structure définie dans l'ordre CREATE TABLE

```
CREATE TABLE <nom_table>
   (<colonne1>   type1[(longueur1)] [contrainte_colonne1],
    <colonne2>   type2[(longueur2)] [contrainte_colonne2],
    ….
    [contrainte_table1], [contrainte_table2],…
);
```

On peut préciser la clause DEFAULT sur une colonne, valeur par défaut sous forme : constante, variable (sysdate, ..), fonction SQL

Création de tables (2)

- **Exemple**

Création de tables (3)

- **Création le schéma physique d'une table avec importation de données :** crée une table avec une structure définie dans l'ordre CREATE TABLE **et l'instancie avec les données** issues de l'évaluation de l'ordre SELECT

```
CREATE TABLE <nom_table>
   [(<colonne1> [,<colonne2>,…])]
    AS SELECT …
;
```

- **Exemple**

Interrogation du méta-shéma

- **Rappel**
  - Méta-schéma : informations qui décrivent les objets de la base, dont les tables

- **Affichage des tables de l'utilisateur**

```
SELECT TABLE_NAME FROM User_Tables;
```

- **Description d'une table existante**

```
DESC <nom_table>;
```

Création de tables

```
CREATE TABLE Produit(RefProd VARCHAR2(10),
 Designation VARCHAR2(20),Categorie CHAR(1),
 Prix NUMBER(5,2),Stock NUMBER(2));
```

Produit

RefProd	Designation	Categorie	Prix	Stock	
TT43	Table basse	A	20	-3	Valeur de stock négative
CH67	Chaise classique	A	-4	67	Valeur de prix négative
TR56		C	45	125	Désignation non rensignée
TT567	Table basse	B	90	100	Même nom pour 2 articles
	Tabouret	A	15	250	Clé primaire absente

Définition de contraintes d'intégrité (1)

- **Contraintes d'intégrité structurelle**
  - Clé primaire : **PRIMARY KEY (PK)**
  - Unicité : **UNIQUE (UN)**
  - Non nullité : **NOT NULL (NN)**
  - Clé étrangère : **FOREIGN KEY (FK)**
  - Domaine : **CHECK (condition) (CK)**

- **Deux niveaux de définition**
  - Contrainte attachée à la colonne : la contrainte porte sur une (1) colonne
  - Contrainte attachée à la table : la contrainte porte sur une (1) ou plusieurs colonnes

Définition de contraintes d'intégrité (2)

Ce qui est bon à savoir

- ✓ Il est possible de créer des contraintes sans les nommer mais pas recommandé : le nommage des contraintes permet de faciliter leur gestion : activation/désactivation, suppression
- ✓ Si une contrainte n'est pas nommée, le système lui attribue automatiquement un nom : <SYS_Cnnnnnnn>, n entier
- ✓ Les contraintes sont représentées dans des méta-tables

Définition de contraintes d'intégrité (2)

Contrainte attachée à la colonne

Contrainte définie juste après la colonne

```
[CONSTRAINT <nom_contrainte>] type_contrainte;
```

Contrainte attachée à la table

Contrainte définie après toutes les colonnes de la table

```
CONSTRAINT <nom_contrainte>
            type_contrainte (<colonne1> [<colonne2>,…]);
```

Définition de contraintes d'intégrité (2)

- **Vérification des contraintes d'intégrité structurelle**
  - ***Mode par défaut*** : à chaque requête SQL.

    Peut être lourde lors de l'exécution de transaction impliquant un volume important de tuples avec de nombreuses contraintes !

  - ***Mode différé*** : indique que la vérification de la contrainte est différée à la fin de la transaction

```
CONSTRAINT nom_contrainte
[NOT] DEFERRABLE
[INITIALLY {DEFERRED|IMMEDIATE}]
```

Contrainte de clé primaire

■ Clé primaire : PRIMARY KEY (PK)

Contrainte attachée à la colonne

```
[CONSTRAINT <nom_contrainte>] PRIMARY KEY;
```

```
CREATE TABLE Produit(
RefProd VARCHAR(10) CONSTRAINT pk_Prod
PRIMARY  KEY,
Designation VARCHAR(20), ….);
```

Contrainte attachée à la table

```
[CONSTRAINT <nom_contrainte>]
            PRIMARY KEY (<colonne1> [<colonne2>,…]);
```

```
CREATE TABLE Produit (
  RefProd VARCHAR(10),
  Designation VARCHAR(20),…
  CONSTRAINT pk_Prod PRIMARY KEY (RefProd));
```

Contrainte de clé primaire (mono-attribut)

Clé primaire : PRIMARY KEY

- Toute table a une clé primaire
  - ✓ Une clé primaire porte des valeurs uniques, un index est généré automatiquement sur la ou les colonnes clé
  - ✓ Une clé primaire porte des valeurs non nulles, inutile de le spécifier; attention, par défaut tout autre colonne peut porter la valeur `null`
  - ✓ Une clé primaire peut être multi-attributs
  - ✓ Une clé primaire peut être référencée par une autre table
  - ✓ Recommandé : préfixer le nom de la contrainte par « pk_.. »

Contrainte de clé primaire (multi-attribut)

- **Clé primaire multi-attribut est définie au niveau de la table**

  Dans le cas de **clé multi-attribut**, il est recommandé de définir la **contrainte de non nullité** au niveau de chaque colonne participante à la clé.

Contrainte d'unicité

Contrainte d'unicité : UNIQUE

Deux tuples ne peuvent avoir la même valeur de l'attribut (ou groupe d'attributs) auquel est attaché la contrainte UNIQUE

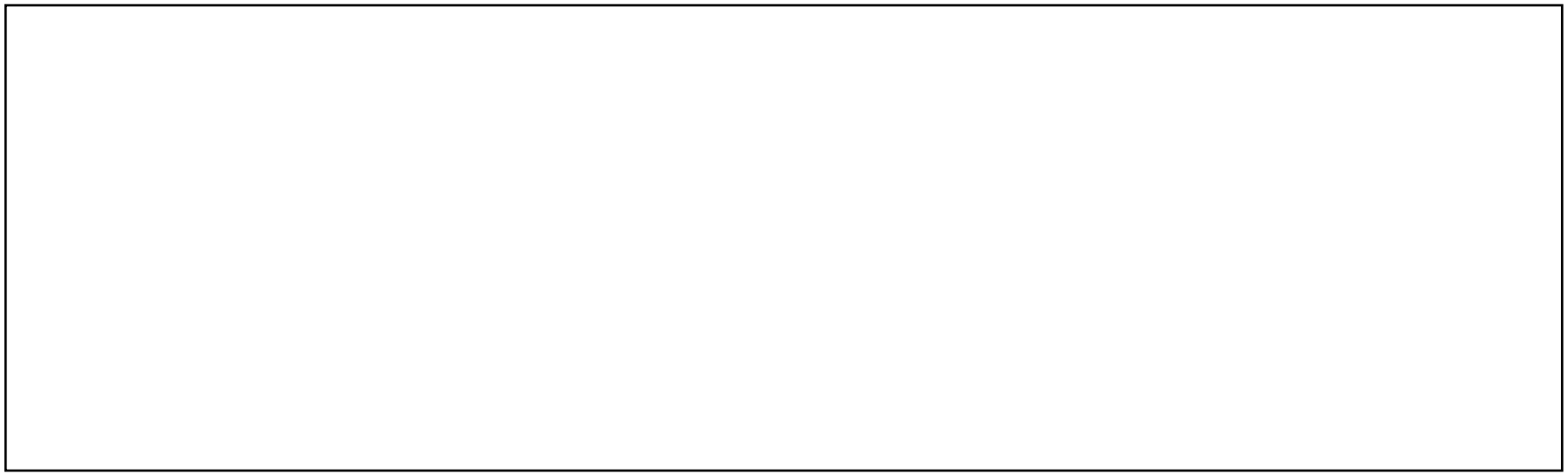
Contrainte attachée à la colonne

Contrainte attachée à la table

Contrainte de non nullité

- **Contrainte de non nullité : NOT NULL**

  L'attribut auquel est attaché la contrainte NOT NULL ne peut contenir des valeurs nulles



Contrainte de clé étrangère (1)

- **Contrainte de clé étrangère : FOREIGN KEY, REFERENCES**

Contrainte attachée à la colonne

```
[CONSTRAINT <nom_contrainte>] REFERENCES
<nom_table>(<nom_col>[,<nom_col> ]..)
[ON DELETE CASCADE];
```

Contrainte attachée à la table

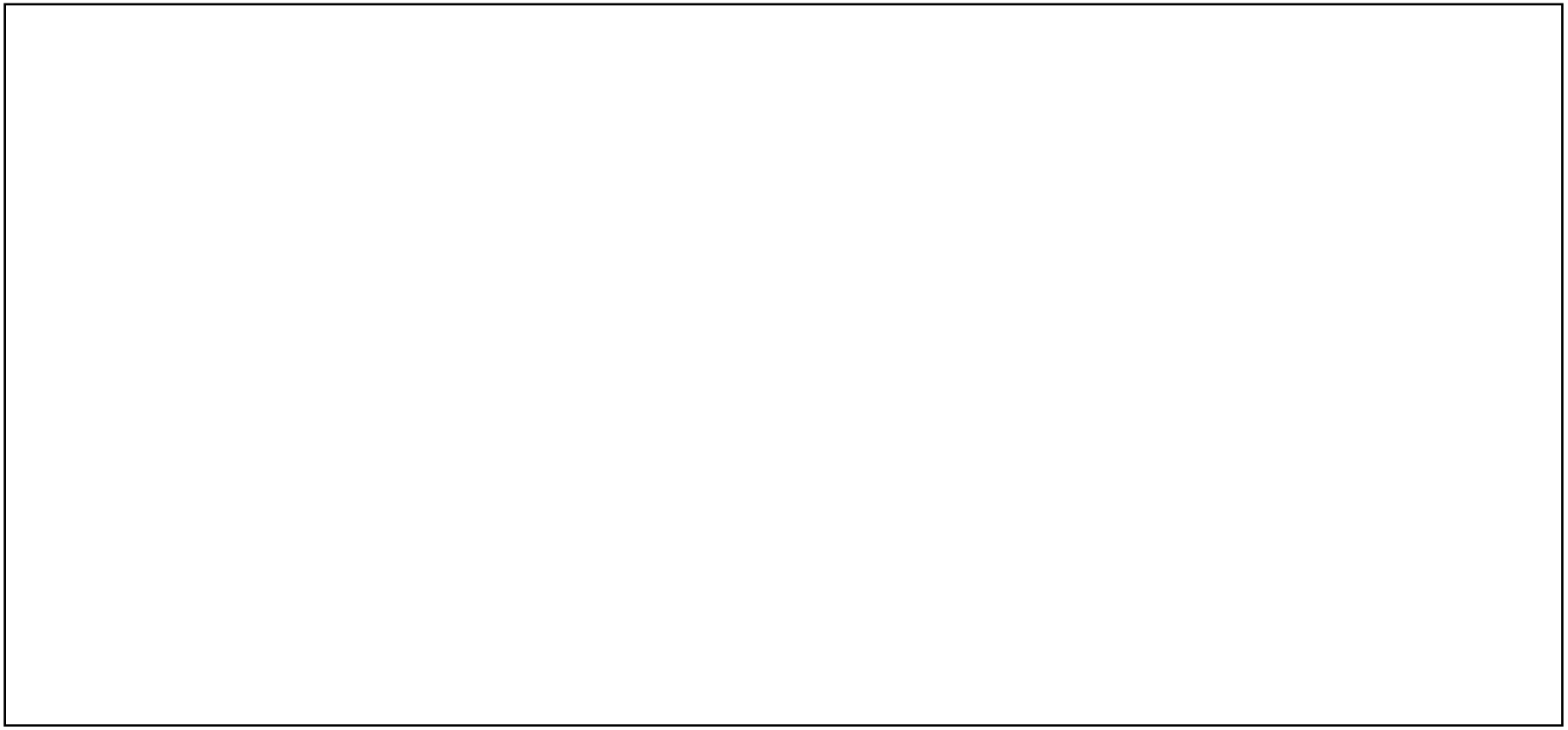
```
CONSTRAINT <nom_contrainte> FOREIGN KEY (<nom_col>)
REFERENCES <nom_table>(<nom_col>[,<nom_col>].. )
[ON DELETE CASCADE];
```

Contrainte de clé étrangère (2)

- **Contrainte de clé étrangère : FOREIGN KEY, REFERENCES**

- ✓ Permet de déclarer une clé étrangère avec une (ou plusieurs) clés primaires ou UNIQUE de la même table ou d'une autre table
- ✓ Valeurs nulles ou référençant une valeur existante dans la colonne référencée. Les clés étrangères peuvent être nulles si aucune contrainte NOT NULL n'est spécifiée
- ✓ Option ON DELETE CASCADE indique que la suppression d'une ligne de la table-parent va entraîner la suppression des lignes de la table fils qui la référencent

Contrainte de clé étrangère (3)



Contrainte de clé étrangère (4)

- **Exemple : Clause ON DELETE CASCADE**

LignComm

RefCom	RefProd#	Quantite
21A12	23	18
34B10	25	23

Commande

RefCom#	CodCli#
21A12	18
34B10	2

Client

CodCli	Nom	Prenom
18	TOTO	Martin
2	TITI	Amélie

- **Exemple :** référence dans la même table

Employé

IdEmp	Nom	Chef	Salaire
21	**TOTO**	**18**	**1500**
18	**TINTIN**	**18**	**2800**

Contrainte de DOMAINE

Contrainte de DOMAINE : CHECK (condition)

L'attribut auquel est attaché la contrainte CHECK (condition) prend une valeur qui satisfait la condition

Contrainte attachée à la colonne

```
Att ….. CHECK (condition);
Att ….. CONSTRAINT nom_contrainte CHECK (condition);
```

Contrainte attachée à la table

```
CONSTRAINT nom_contrainte CHECK (condition);
```

Contrainte de DOMAINE

Contrainte de DOMAINE : CHECK (condition)

```
CREATE TABLE LignComm (
RefComm VARCHAR(10) CONSTRAINT nn_RefComm NOT NULL,
RefProd VARCHAR(10) CONSTRAINT nn_ RefProd NOT NULL,
QtiteComm NUMBER(4) CONSTRAINT ck_Qtite CHECK(QtiteComm>0),
CONSTRAINT pk_LignComm PRIMARY KEY(RefComm,Refprod));
```

```
CREATE TABLE Employe (
NumEmp NUMBER (4) CONSTRAINT pk_NumEmp PRIMARY KEY,
Salaire NUMBER(9,2),
Commission NUMBER(5,2),
CONSTRAINT ck_salgar CHECK (salaire + NVL(commission)>1500)
CONSTRAINT ck_salsupcomm CHECK (salaire > commission));
```

Bilan sur les contraintes

- **PRIMARY KEY (pk)**
  - Valeurs d'une clé primaire, uniques et non nulles
- **FOREIGN KEY (fk)**
  - Valeurs d'une clé étrangère incluses dans les valeurs d'une colonne **(source)** unique
- **UNIQUE (un)**
  - Valeurs d'une colonne sans autorisation de doublons, avec valeurs `null` autorisées
- **CHECK (ck)**
  - Valeurs d'une colonne qui satisfont une condition spécifiée dans la contrainte
- **NOT NULL (nn)**
  - Toutes les valeurs de la colonne sont renseignées (valeur `null` interdite)

Méta-schéma et contraintes

Interrogation des contraintes définies dans une table

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, STATUS
FROM USER_CONSTRAINTS
WHERE TABLE_NAME= <table_name>
ORDER BY CONSTRAINT_NAME;
```

```
SELECT CONSTRAINT_NAME, CPNSTRAINT_TYPE, STATUS
FROM USER_CONSTRAINTS
WHERE TABLE_NAME= 'Produit'
ORDER BY CONSTRAINT_NAME;
```

Renommage d'une table

```
RENAME <ancienNom> TO <nouveaunom>;
```

- ✓ Toutes les contraintes de l'ancienne table sont reportées automatiquement sur la nouvelle
- ✓ Vues, synonymes et procédures doivent être recrées

Mise à jour de la structure des tables

- **Mise à jour des colonnes**
  - Ajout, suppression de colonnes
  - Modification du type des colonnes

- **Mise à jour des contraintes**
  - Ajout, suppression de contraintes
  - Renommage de contraintes
  - Activation/désactivation de contraintes

Mise à jour des colonnes

Ajout de colonnes

```
ALTER TABLE <nom_table>
ADD (col1 <type1>, [col2 <type2> …]);
```

Modification des colonnes

```
ALTER TABLE <nom_table>
MODIFY (col1 <type1>, [col2 <type2> …]);
```

Modification possible si : (1) les valeurs de la colonne à modifier sont nulles ou (2) nouveau type compatible avec les données déjà instanciées

Suppression des colonnes

```
ALTER TABLE <nom_table>
DROP COLUMN <nom_colonne>;
```

Mise à jour d'une table : exemples

Mise à jour des contraintes (1)

- **Ajout, suppression, renommage de contraintes**

```
ALTER TABLE <nom_table>

DROP CONSTRAINT <nom_contrainte>;

ADD CONSTRAINT <nom_contrainte> <type_contrainte>;

RENAME CONSTRAINT <anc_nom> TO <nouv_nom>;
```

Mise à jour des contraintes : exemples

- **Ajout, suppression, renommage de contraintes**

Mise à jour des contraintes (2)

Activation/désactivation des contraintes

Permet d'activer (ENABLE) ou d'annuler (DISABLE) momentanément la vérification de certaines contraintes, parfois pour améliorer les performances lors de l'ajout d'un grand volume de données.

```
ALTER TABLE <nom_table>
ENABLE|DISABLE CONSTRAINT <nom_contrainte>;
```

Les contraintes existent dans le dictionnaire des données mais ne sont plus actives !

La commande ENABLE met la valeur de l'attribut STATUS à ON

La commande DISABLE met la valeur de l'attribut STATUS à OFF

Mise à jour des contraintes : exemples

Activation/désactivation des contraintes

```
ALTER TABLE Lign_Comm
DISABLE CONSTRAINT ck_Qtite;

ALTER TABLE Employe
DISABLE PRIMARY KEY;

ALTER TABLE Lign_Comm
ENABLE CONSTRAINT ck_Qtite;
```

Mise à jour des contraintes (3)

Récupération des données non valides

Permet de récupérer dans une table « rejet » des données ne respectant pas une contrainte d'intégrité (généralement suite à son activation)

```
ALTER TABLE <nom_table>
ENABLE <nom_contrainte>
EXCEPTION INTO <nom-table-rejet>;
```

La table rejet a le schéma :

```
CREATE TABLE <nom_table_rejet> (
row ROWID,
owner VARCHAR2(30),
Table_name VARCHAR2(30),
Constraint VARCHAR2(30));
```

Pour récupérer les données ne satisfaisant pas les contraintes

```
SELECT <nom_table>.*
FROM <nom_table>, <nom-table-rejet>
WHERE <nom-table>.ROWID=<nom-table-rejet>.row;
```

Suppression d'une table

Suppression de table

Suppression du schéma physique de la table et des données

```
DROP TABLE <nom_table>;
```

```
DROP TABLE EMPLOYE;
```

-- Attention à la suppression d'une table avec clé référencée par un FOREIGN KEY dans une table fille ('*enfant orphelin*') → ERROR ORA-02449

-- Suppression de table et des contraintes de référence filles

```
DROP TABLE <nom_table> CASCADE CONSTRAINTS;
```

```
DROP TABLE LignComm CASCADE CONSTRAINTS;
```

Manipulation des données

- **Insertion de données**
  - ✓ Insertion de tuples dont les valeurs sont spécifiées directement dans la commande ou issues de l'évaluation d'un ordre SELECT
  - ✓ Commande : `INSERT`
- **Modification des données**
  - ✓ Modification des valeurs de colonnes des tuples d'une table
  - ✓ Commande : `UPDATE`
- **Suppression des données**
  - ✓ Suppression de tous les tuples d'une table ou de ceux qui répondent à une condition
  - ✓ Commande : `DELETE`

Insertion des données

Insertion de valeurs directes

```
INSERT INTO <nom_table> [(liste de colonnes)]
VALUES (liste des valeurs);
```

Insertion de valeurs issues d'autres tables

```
INSERT INTO <nom_table> [(liste de colonnes)]
   SELECT …;
```

Par défaut, toutes les colonnes sont dans l'ordre donné lors de la création de la table

Les colonnes qui ne sont pas dans la liste auront la valeur par défaut NULL

Les valeurs doivent respecter les contraintes des colonnes

Les chaînes de caractères sont entre ' '

Si la clé étrangère est déclarée NOT NULL, l'insertion d'une ligne « fils » n'est possible que si elle est rattachée à une ligne « père » existant. Dans le cas contraire, insertion possible (père inconnu)

Insertion des données : exemples

Insertion des données : cas des valeurs auto-incrémentées

```
-- création de la séquence
CREATE SEQUENCE <nomsequence>
[START WITH N]    -- valeur par défaut de N est 1
[INCREMENT BY M]; -- valeur par défaut de M est 1

-- utilisation de la séquence
INSERT INTO <nomTable> VALUES
(<nomsequence>.nextval, …..);
```

```
-- création de la séquence
CREATE SEQUENCE codeemp
START WITH 1   INCREMENT BY 1;
-- utilisation de la séquence
INSERT INTO Employe VALUES(codeemp.nextval,2500, 200);
```

Modification des données (1)

- **Mise à jour d'une ou plusieurs colonnes associée (s) à une ou plusieurs lignes**

```
UPDATE <nom_table>
SET colonne1 = valeur1
   [, colonne2 = valeur2…]
[WHERE predicat];
```

- Pour chaque tuple qui satisfait le prédicat spécifié dans la clause WHERE, on affecte le résultat de chaque expression à chaque colonne col¡ spécifiée
- La valeur affectée doit respecter le type et les contraintes de la colonne

Modification des données (2)

- **Mise à jour d'un ensemble de colonnes à partir d'une requête SELECT**

```
UPDATE <nom_table>
SET (colonne1, colonne2, ..) = (SELECT ….)
[WHERE predicat];
```

Pour chaque tuple qui satisfait le prédicat spécifié dans la clause WHERE, le résultat de la clause SELECT est affecté à chaque colonne tuple (col1, col2, ..) de la table.

L'ordre SELECT ne doit retourner qu'une seule ligne au plus !

Modification des données (2)

- **Mise à jour d'un ensemble de colonnes à partir d'une requête SELECT**

Employé

IdEmp	Nom	Chef	Salaire
21	**TOTO**	**18**	**1500**
18	**TINTIN**	**18**	**2800**

FichEMP

CodEmp	NbH	QuotH
21	**20**	**80**
18	**10**	**70**

Suppression des données

- **Suppression d'une ou plusieurs lignes**

```
DELETE FROM <nom_table> [WHERE predicat];
```

- **Exemples**

```
DELETE FROM Produit
WHERE Categorie='A';
```

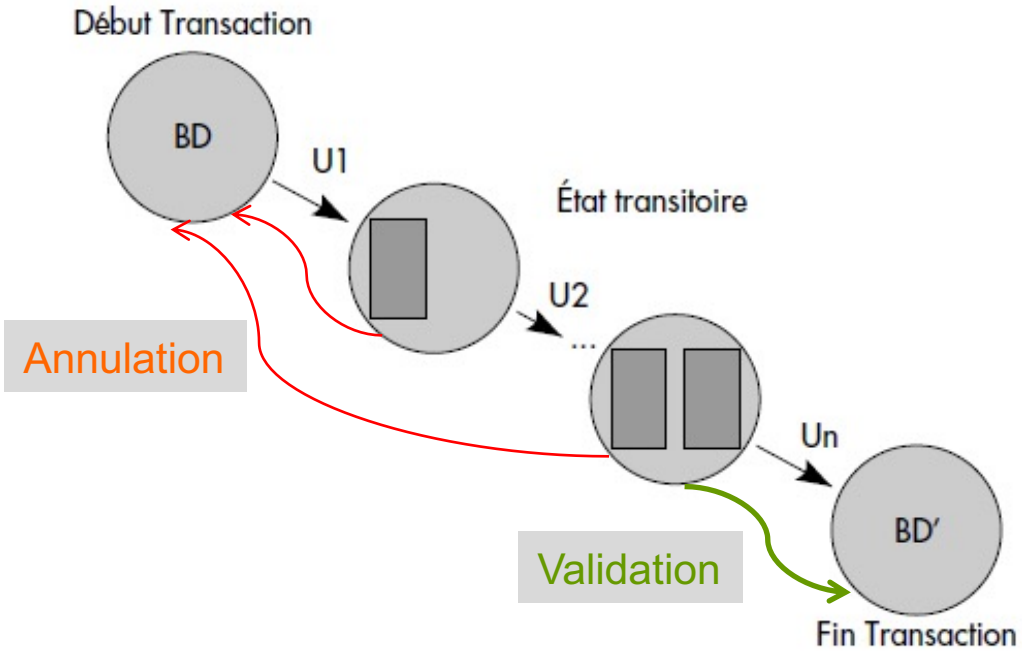
```
DELETE FROM Commande
WHERE CodCli IN (SELECT CodCli
                 FROM MauvaisPayeurs);
```

Cohérence de l'intégrité référentielle lors de la manipulation de données

Commande	**Table « parent »**	**Table « fils »**
INSERT	Correcte si la clé primaire est unique	Correcte si la clé étrangère est référencée dans la table « père » ou est nulle
UPDATE	Correcte si la commande ne laisse pas de lignes dans la table « fils » ayant une clé étrangère non référencée	Correcte si la nouvelle clé étrangère référence une ligne « père » existant
DELETE	Correcte si aucune ligne de la table « fils » ne référence le ou les lignes supprimées	Correcte sans condition
DELETE CASCADE	Correcte sans condition	Correcte sans condition

Principe des transactions

Transaction



Source : https://sgbd.developpez.com/

Principes des transactions

Transaction

Ensemble d'opérations (INSERT, UPDATE, DELETE) exécutés comme un bloc (TOUT ou RIEN)

COMMIT (Validation, confirmation)

- ✓ Commande permettant le transfert physique des données résultant de la mise à jour vers la BD : les transactions de mise à jour reportées dans la BD
- ✓ Le SGBD enregistre l'instant de validation (SAVEPOINT) qui correspond à un état cohérent de la base

ROLLBACK (Annulation, retour arrière)

- ✓ Les mises à jour sont annulées jusqu'au point de validation

Interrogation des données (1)

- Objectif : sélectionner des données à partir de la BD qui satisfont des conditions
  - ✓ *Sélectionner* : `SELECT col1, col2, …`
  - ✓ *À partir de la BD* : `FROM table1, table2, …`
  - ✓ *Conditions à satisfaire* : `WHERE <condition>`

- Requête `SELECT` sur tables et opérations algébriques sur relations
  - ✓ *Projection* $\pi_{a1\ldots an}R$: sur les colonnes précisées après le SELECT
  - ✓ *Sélection* $\sigma_{predicat}\ R$: sur les lignes qui répondent à la `<condition>` précisée après `WHERE`
  - ✓ *Produit cartésien* RXS: des tables avec évaluation de la `<condition>`
  - ✓ Le résultat de toute opération est une relation : propriété de fermeture

Interrogation des données (2)

Commande

RefCom	CodCli
21A12	18
34B10	2

Projection

Selection

LignComm

RefCom	RefProd	Quantite
21A12	**23**	18
21A12	**53**	14
34B10	**25**	23

Client

CodCli	Nom	Prenom
18	**TOTO**	**Martin**
2	**TITI**	**Amelie**

Interrogation des données (3)

Syntaxe générale

```
SELECT [DISTINCT|UNIQUE| ALL] {listeColonnes/expression}
FROM nomTable1, [, nomTable2]
[WHERE condition]
[GROUP BY <colonne>|<expression>]
[HAVING <condition>]
[{UNION|UNION ALL|INTERSECT|MINUS} (sousRequête)]
[ORDER BY <colonne>|<expression> [asc|desc]];
```

Clause SELECT

- ***SELECT [DISTINCT|UNIQUE|ALL] ListeColonnes***
  - `DISTINCT/UNIQUE` *:* permet de donner les valeurs uniques (sans doublons)
  - `ALL` *:* prend en compte les doublons
  - `ListeColonnes`

    ✓ *Forme*

```
{*|expression1 [[AS] alias1 ] [,expression2 [AS] alias2 ]…}
```

    * : pour désigner toutes les colonnes

    `expression` : nom de la colonne, fonction, constante ou calcul

    `alias` : renomme l'expression

    nom1, nom2, … : alias pour désigner la colonne

Clause FROM

- ***SELECT Monotable***

```
SELECT FROM <Table> [Alias]
```

  - Une table après la clause SELECT
  - La table peut être nommée avec un synonyme (alias) : l'intérêt étant d'utiliser une abréviation du nom de la table

- ***SELECT Multi-tables***

```
SELECT FROM <Table1> [Alias], <Table2> [Alias], …
```

  - Plusieurs tables après la clause SELECT
  - Les tables peuvent être nommées à l'aide d'un alias

Clauses SELECT…FROM : exemples SELECT (1)

```
SELECT * FROM PRODUIT;
```

Toutes les colonnes

RefProd	Designation	Categorie	Prix	Stock
18	Tabouret	A	16,78	200
23	Chaise	B	23,00	350
25	Table	B	123,00	36

```
SELECT RefProd, Stock FROM Produit;
```

RefProd	Designation	Categorie	Prix	Stock
18	Tabouret	A	16,78	200
23	Chaise	B	23,00	350
25	Table	B	123,00	36

Clauses SELECT…FROM : exemples (2)

```
SELECT RefComm AS "Reference" FROM LignComm;
```

Reference
21A12
21A12
34B10

```
SELECT DISTINCT RefComm FROM LignComm;
```

Valeurs uniques

RefCom
21A12
34B10

Clause WHERE : forme générale (1)

- ***[WHERE CONDITION]***

  La clause WHERE permet de préciser un prédicat sur les lignes

  Condition : expression logique (retournant VRAI ou FAUX) composée de colonnes, expressions, constantes qui sont des opérandes liés par des opérateurs

  ```
  <opérande> <opérateur> <opérande>
  ```

  `<opérateur>` de base

  - *De comparaison (>, =, <, >=, <=, <>, !=)*
  - *Logiques (NOT, AND, OR)*

Clause WHERE : opérateurs ensemblistes (1)

X : (x_1,x_2,...,x_N); a : colonne, valeur, expression

a IN (X)	`Couleur IN ('bleu', 'blanc', 'rouge')`
a NOT IN (X)	`Diplôme NOT IN (1,2,3)`
a= ANY(X) ***a= SOME(X)***	Condition vraie si satisfaite pour une ou plusieurs valeurs de la liste `PRIX < ANY (12, 24, 45)`
a<> ALL(X) ***a> ALL(X)*** ***a>= ALL(X)***	Condition vraie si satisfaite pour toutes les valeurs de la liste `PRIX < ALL (12, 24, 45)`
IS [NOT] NULL	Condition vraie si la valeur est [n'est pas] `NULL`

Attention, ne jamais utiliser les opérateurs '=' ou '<>' devant la valeur 'NULL'.
Utiliser strictement IS NULL, IS NOT NULL
X=NULL est toujours évaluée à faux

Clause WHERE : opérateurs (2)

a[NOT] BETWEEN v1 AND v2	Condition vraie si *a* est (ou n'est pas) compris entre les deux valeurs données *v1 et v2* (bornes comprises) `Datefact Between '01-JAN-95' AND '31-DEC-95'`
[NOT] EXISTS <sous-requête>	Condition vraie si la requête retourne (ou ne retourne pas) au moins un tuple (voir plus tard)
a[NOT] LIKE <exp>	Condition vraie si *a* contient (ou ne contient pas) un ensemble de caractères définis avec les jokers « - » et « % » _: caractère quelconque %: suite de caractères quelconques `NOM like '_ _ _O%'`

Clause WHERE : opérateurs (3)

- Opérande
  - ✓ *Nom de colonne*
  - ✓ *Valeur*
    - o Numérique (entière ou décimale)
    - o Alphanumérique (entre quotes '')
    - o Date
  - ✓ *Expression*
    - o Arithmétique (utilisant les + ,- ,*, /)
    - o Fonctions
  - ✓ *Sous-requête*
    - o Requête fournissant une valeur ou une liste de valeurs compatibles

Clause WHERE : exemples

Produits de la catégorie 'A' ou 'C'

RefProd	Designation	Categorie	Prix	Stock
18	Tabouret	A	16,78	200

Références des produits dont le Prix est compris entre 15 et 25 euros et dont la désignation commence par la caractère 't', casse non respectée

RefProd
18

Clause WHERE : exemples

Références des produits commandés avec une quantité supérieure strictement à 15, et quantités commandées associées

RefProd	QuantiteCommandee
23	18
25	23

Clause WHERE : fonctions (1)

- Utilisées dans les expressions, opérandes des conditions
- Fonctions catégorisées selon le type des arguments associés
  - Fonctions numériques
  - Fonctions chaînes de caractères
  - Fonctions date
  - Fonctions de conversion
  - Autres fonctions

Fonctions arithmétiques

ABS(<n>)	Valeur absolue de *n*
CEIL(<n>)	Entier supérieur ou égal à *n*
FLOOR(<n>)	Troncature à valeur entière
MOD(<m>,<n>)	Reste de la division de *m* par *n*
POWER(<m>,<n>)	*m* élevé à la puissance *n*
ROUND(<m>,<n>)	*m* arrondi à *n* décimales
SIGN(<n>)	Signe (-1 si <0, 0 si =0, 1 si >0)
SQRT(<n>)	Racine carrée de *n* (0 si *n*<0)
TRUNC(<m>,<n>)	*m* tronqué à *n* décimales
LN (<n>)	Logarithme népérien
EXP(n)	*e* (2.7182) à la puissance *n*

Fonctions sur les chaînes de caractères

INITCAP(<c>)	La première lettre de chaque mot est mise en majuscules
LOWER(<c>)	Conversion en minuscules
UPPER(<c>)	Conversion en majuscules
LTRIM(<c>)	Suppression des espaces à gauche
RTRIM(<c>)	Suppression des espaces à droite
REPLACE(<c>,<c1>,<c2>)	Remplacement dans *c* de *c1* par *c2*
SOUNDEX(<c>)	Donne le son des mots de la chaîne (permet de faire des recherches en phonétique)
SUBSTR(<c>,<c1>,<c2>)	Sous-chaîne extraite commençant au caractère de rang *d* et de longueur *l*
LENGTH(<c>)	Longueur de la

Fonctions sur les dates

ADD_MONTH(<d>,<n>)	Ajoute *n* mois à la date *d*
LAST_DAY(<d>)	Dernier jour du mois *d*
MONTH_BETWEEN(<d1>,<d2>)	Nombre de mois entre les dates *d1* et *d2*
NEW_TIME(…)	Date et heure dans un autre méridien
SYSDATE	Date et heure système
CURRENT_DATE	Retourne la date courante
LAST_DAY(d)	Retourne le dernier jour

Autres fonctions

TO_CHAR(c)	Retourne a chaîne *c* en VARCHAR2
GREATEST(expression[,expression]…)	Retourne la plus grande des expressions
LEAST(expression[,expression]…)	Retourne la plus petite des expressions
NULLIF(expr1, expr2)	Retourne NULL si *expr1=expr2*, *expr1* sinon
NVL(expr1, expr2)	Retourne *expr2* si *expr1=NULL*

CLAUSE ORDER BY

- Les résultats d'un `SELECT` peuvent être triés (ordonnés) selon une ou plusieurs clés : nom ou position de colonne

```
ORDER BY <expression1 >[ASC/DESC], expression2
    [ASC/DESC], …
```

Exemples

```
SELECT RefProd, Prix FROM Produit
ORDER BY Prix DESC;
```

Affichage du produit le plus cher au produit le moins cher

Projection avec SELECT sans ORDER BY

RefProd	Prix
18	16,78
23	23,00
25	123,00

RefProd	Prix
25	123,00
23	23,00
18	16,78

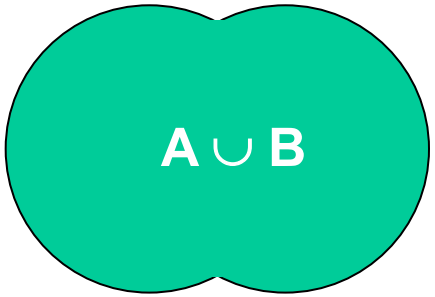
Projection avec SELECT ... ORDER BY

Opérateurs ensemblistes

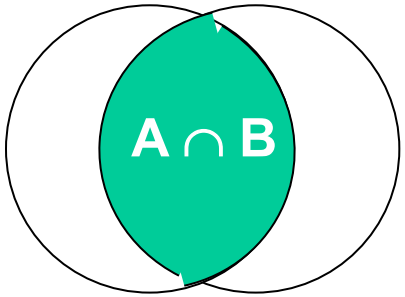
- Permettent de construire des ensembles à partir des résultats de SELECT considérés comme des ensembles de lignes

```
SELECT ….FROM nomTable1 [WHERE…]
Opérateur
SELECT ….FROM nomTable2 [WHERE…]
```

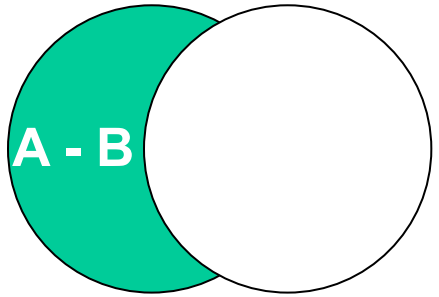
- ✓ Pas d'opérateur de division (DIVIDE) !
- ✓ Résultats du select compatibles (nombre de colonnes, types de colonnes)
- ✓ Attention à l'ordre des SELECT



UNION|UNION ALL



INTERSECT



MINUS

Opérateurs ensemblistes

- Les tables doivent être compatibles : même nombre d'attributs de mêmes types

- **Union** : les lignes des deux tables sont concaténées verticalement sans doublons dans le résultat. L'opération est commutative

- **Intersection** : les lignes communes aux deux tables sont ajoutées au résultat. L'opération est commutative

- **Différence** : les lignes présents dans la première table et absents dans la seconde table sont ajoutées au résultat. L'opération n'est pas commutative

Opérateurs ensemblistes - Union

Produits de la catégorie A et B

Requête équivalente avec l'opérateur `OR`

```
SELECT RefProd, Prix FROM PRODUIT
WHERE Upper(Categorie) ='A' OR
Upper(Categorie) ='B';
```

Requête équivalente avec l'opérateur `IN`

```
SELECT RefProd, Prix FROM PRODUIT
WHERE Upper(Categorie) IN ('A', 'B');
```

Opérateurs ensemblistes - Intersection

Produits qui sont des produits de luxe

```
SELECT Nom, Prenom FROM Client
      WHERE UPPER(Categorie)='A'
INTERSECT
SELECT Nom, Prenom FROM MauvaisPayeurs;
```

Clients de la catégorie 'A' qui sont archivés dans la table MauvaisPayeurs

Composition d'opérateurs ensemblistes - Différence

⇨

Produits qui ne sont pas des produits de luxe et dont le stock est supérieur à 50

⇨ Le parenthésage définit l'ordre d'exécution

CLAUSE GROUP BY

- Regroupe des tuples (crée des groupes ou des partitions) ayant les mêmes valeurs pour des expressions sur les colonnes
- Une fonction de groupe (d'agrégation) agrège les valeurs d'une colonne en une seule valeur de sortie

```
GROUP BY <Exp> [HAVING <Prédicat>]
```

Indique comment créer les groupes de tuples

Indique quels groupes sélectionner/filtrer

`<Exp>` : colonne ou groupe de colonnes de regroupement

`<Prédicat>` : condition sur les groupes (et non sur les tuples !!)

Les fonctions figurant dans le SELECT suivi du GROUP BY sont des fonctions de groupe

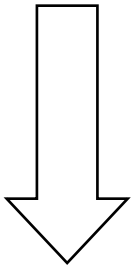
Les colonnes figurant dans le SELECT, doivent figurer dans le GROUP BY

Fonctions arithmétiques de groupe

AVG(<n>)	Moyenne des valeurs de *n* (valeurs nulles non comptées)
COUNT(*)	Nombre de tuples renvoyés par la requête
COUNT(<n>)	Nombre de valeurs non nulles
SUM(<n>)	Somme des valeurs de *n*
MAX(<n>)	Valeur maximum de *n*
MIN(<n>)	Valeur minimum de *n*
STDDEV(<n>)	Ecart type de *n* (valeurs nulles non comptées)
VARIANCE(<n>)	Variance de *n* (valeurs nulles nom comptées)
EVERY	Renvoie un booléen SQL qui vaut vrai si les occurrences sont toutes vraies
ANY|SOME	Renvoie un booléen SQL si au moins une occurrence est vraie

CLAUSE GROUP BY- Exemples (1)

RefProd	Catégorie	Prix
18	**A**	**34**
42	**B**	**20**
45	**A**	**60**
12	**C**	**50**
23	**B**	**10**

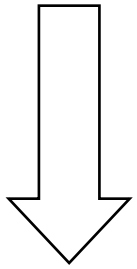


```
Cate-
gorie    AVG(Prix)          COUNT(*)
-----    ------------       ---------
A          47                 2
B          15                 2
C          50                 1
```

CLAUSE GROUP BY- Exemples (2)

```
SELECT Categorie, AVG(prix), COUNT(*)
FROM Produit
GROUP BY Categorie
HAVING COUNT(*)>=2;
```

RefProd	Catégorie	Prix
18	**A**	**34**
42	**B**	**20**
45	**A**	**60**
12	**C**	**50**
23	**B**	**10**



```
Cate-
gorie    AVG(Prix)          COUNT(*)
-----    ------------       ---------
A          47                 2
B          15                 2
```

Jointures

- Rappel du produit cartésien entre deux tables : RXS
  - Le résultat d'un produit cartésien *RXS* comprend $|R|X|S|$ tuples : combinaison des tuples des tables R et S par juxtaposition

Commande

RefCom	CodCli
21A12	18
34B10	2

Client

CodCli	Nom	Prenom
18	TOTO	Martin
2	TITI	Amelie

Commande X Client

RefCom	CodCliCo	CodCliCl	Nom	Prénom
21A12	18	18	TOTO	Martin
21A12	18	2	TITI	Amelie
34B10	2	18	TOTO	Martin
34B10	2	2	TITI	Amelie

Jointures

- Jointure naturelle : $R \bowtie S$
  - Une jointure naturelle est un produit cartésien sur lequel est appliqué une sélection sur les attributs communs

Commande

RefCom	CodCli
21A12	18
34B10	2

Client

CodCli	Nom	Prenom
18	TOTO	Martin
2	TITI	Amelie

(Commande X Client)$_{\text{Commande.CodCli=Client.CodCli}}$

RefCom	CodCliCo	CodCliCl	Nom	Prénom
21A12	18	18	TOTO	Martin
34B10	2	2	TITI	Amelie

Jointures

- Types
  - ***Jointure relationnelles*** *:* une seule clause `FROM` qui relie des tables deux à deux
    - ✓ Equi-jointure : opérateur d'égalité dans la clause de jointure
    - ✓ Inéqui-jointure : opérateur (<>, <, >, >=, <=, BETWEEN, LIKE, IN) dans la clause de jointure
    - ✓ Auto-jointure : cas particulier de l'équijointure qui met en œuvre deux fois la même table. L'utilisation d'un alias devient nécessaire pour distinguer les tables sources des lignes jointes
    - ✓ Jointure externe : favorise une table dite *dominante* par rapport à l'autre dite *subordonnée*. Les lignes de la table dominante sont retournées même si elles ne répondent pas à la condition de jointure
  - *Jointure procédurale* : requêtes qui contiennent des sous-interrogations (`SELECT` comme sous-interrogation).

Jointure relationnelle (1)

Equi-jointure

Commande

RefCom#	CodCli#
21A12	**18**
34B10	**2**

Client

CodCli	Nom	Prenom
18	**TOTO**	**Martin**
2	**TITI**	**Amélie**

LignComm

Commande.Cocli = Client.Cocli

RefCom	CodCli	Nom	Prenom
21A12	**18**	**TOTO**	**Martin**
34B10	**2**	**TITI**	**Amélie**

```
SELECT Nom
FROM Client cl, Commande cm
WHERE cl.CodCli=cm.CodCli;
```

Utiliser l'alias pour désambiguïser

Noms des clients ayant passé des commandes

Jointure relationnelle (2)

Iné-quijointure

Les produits dont les nievaux de stock sont inférieurs à la réserve

RefProd	Catégorie	Stock
18	A	40
42	B	60
45	A	10
12	C	50
23	B	30

RefProd	Reserve	Dépôt
18	10	C
42	50	C
45	20	C
12	20	C
23	50	C

Jointure relationnelle (2)

Auto-jointure

Les produits plus chers que le produit 23

```
SELECT p1. Refprod
FROM Produit p1, Produit p2
WHERE p2.Refprod=23 AND p1.prix>p2.prix
```

Pour chaque produit, les produits plus chers ou de même prix

```
SELECT p1. RefProd, p1. Designation, p2. RefProd, p2.Designation
FROM Produit p1, Produit p2
WHERE (p1. RefProd!=p2.RefProd)
 AND (p1.Prix<=p2.prix);
```

Jointure relationnelle (2)

Auto-jointure

Les produits plus chers que le produit 23

```
SELECT p1. Refprod
FROM Produit p1, Produit p2
WHERE p2.Refprod=23 AND p1.prix>p2.prix
```

Produit p1 Produit p2

p1.Prix > p2 Prix ?

RefProd	Catégorie	Prix	RefProd	Catégorie	Prix	
18	A	10	18	A	10	30> 10 ?
42	B	40	42	B	40	30> 40 ?
45	A	20	45	A	20	30> 20 ?
12	C	50	12	C	50	30> 50 ?
23	B	30	23	B	30	30< 30 ?

Jointure naturelle

- Jointure externe (Semi-jointure)
  - Jointure dont on ne projette que les colonnes provenant d'une tabe (gauche ou droite)
  - En SQL, consiste à ajouter le signe **+** au nom de la colonne de la table dont on suppose les éléments manquants.
  - Les colonnes issues de cette table auront la valeur `NULL`
  - Sous Oracle : `OUTER JOIN`

```
SELECT … FROM <table1>, <table2>
WHERE <table1>.<col> [(+)]=<table2>.<col>[(+)];
```

Jointure naturelle

Jointure externe

Client : table dominante

CodCli	Nom	Prenom
18	TOTO	Martin
15	MOMO	Laure
2	TITI	Amélie

Commande

RefCom#	**CodCli#**
21A12	18
34B10	2

Client X Commande (jointure externe)

RefCom#	CodCli#	Nom	Prenom
21A12	18	TOTO	Martin
	15	MOMO	Laure
34B10	2	TITI	Amélie

```
SELECT c.Codcli, cm.RefComm
FROM Client c, Commande cm
WHERE c.codcli(+) = cm.codcli;
```

Noms des clients et références des commandes associées

Jointures

- Types
  - ***Jointure relationnelles*** *:* une seule clause `FROM` qui relie des tables deux à deux
    - ✓ Equi-jointure : opérateur d'égalité dans la clause de jointure
    - ✓ Inéquijointure : opérateur (<>, <, >, >=, <=, BETWEEN, LIKE, IN) dans la clause de jointure
    - ✓ Auto-jointure : cas particulier de l'équijointure qui met en œuvre deux fois la même table. L'utilisation d'un alias devient nécessaire pour distinguer les tables sources des lignes jointes
    - ✓ Jointure externe : favorise une table dite *dominante* par rapport à l'autre dite *subordonnée*. Les lignes de la table dominante sont retournées même si elles ne répondent pas à la condition de jointure
  - *Jointure procédurale* : requêtes qui contiennent des sous-interrogations (`SELECT` imbriqué).

Jointure en forme procédurale : Sous-interrogations

- Forme générale

```
SELECT colonnesTable1 ..... Niveau 1
FROM Table1
WHERE colonnes |expression(s) {IN|=|opérateur}
(SELECT colonne(s)Table2 FROM Table2 ..... Niveau 2
WHERE colonnes |expression(s) {IN|=|opérateur}
    (SELECT…) ..... Niveau 3
    [AND (conditionsTable2)]
    )
    … ..... Niveau i
[AND (conditionsTable1)];
```

Jointure en forme procédurale : Sous-interrogations

- Principes
  - Chaque niveau *i (i>1)* est une sous-requête de la requête de niveau *i-1* (niveau supérieur)
  - Le résultat d'une sous-requête de niveau *i* peut être à l'entrée d'un opérateur dans la requête de niveau *i-1 (supérieur)*
  - Deux types de jointures procédurales :
    - Sous-interrogations non synchronisées
    - Sous-interrogations synchronisées

Sous-interrogations **non synchronisées**

- Principe d'évaluation
  - Seules les colonnes du SELECT de premier niveau sont extraites
  - Une sous-interrogation est exécutée avant la requête de *niveau supérieur*. Le résultat qu'elle retourne est utilisé par la requête de niveau supérieur
  - Une sous interrogation peut être
    - ✓ **Monoligne** : retourne une seule ligne grâce aux opérateurs =, >, <, >=
    - ✓ **Multilignes** : retourne plusieurs lignes grâce aux opérateurs [NOT] IN, ANY, ALL

Sous-interrogations non synchronisées **mono-lignes**

Prix du produit de catégorie A ayant le plus grand stock, **retourne une seule ligne**

```
SELECT ----------
FROM ----------------
WHERE ------------------- = (--------------
                            --------------
                            ----------------
```

RefProd	Catégorie	Prix	Stock
18	A	34	58
42	B	20	200
45	A	60	609
12	C	50	50
23	B	10	100

Cette requête est monoligne seulement dans le cas où un seul client a fait la commande la plus petite➔ privilégier IN

Sous-interrogations non synchronisées **mono-lignes**

Le code du client qui a a fait la plus petite commande d'un produit donné

```
SELECT CodCli
FROM Commande
WHERE RefComm = (----------------
                 ----------------
                 ----------------=
                        (----------
                         ----------------));
```

Cette requête est monoligne seulement dans le cas ou un seul Client a fait la commande la plus petite➔ privilégier IN

Sous-interrogations non synchronisées **mono-lignes**

Commande

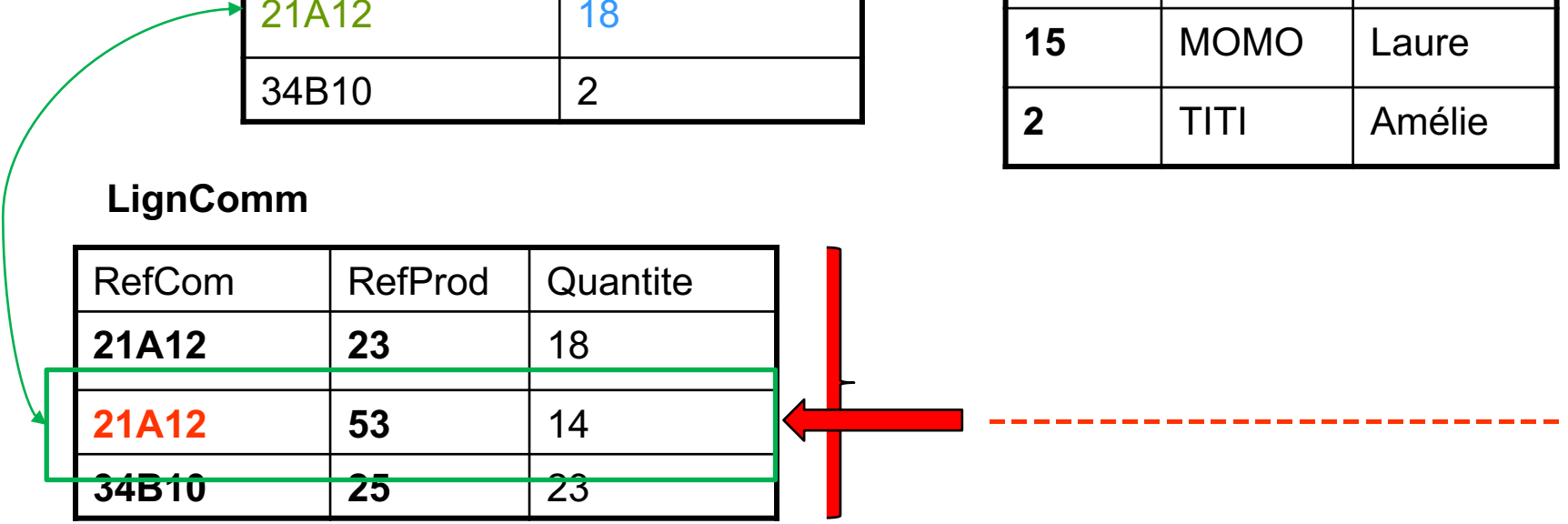
RefCom#	**CodCli#**
21A12	18
34B10	2

Client

CodCli	Nom	Prenom
18	TOTO	Martin
15	MOMO	Laure
2	TITI	Amélie

LignComm

RefCom	RefProd	Quantite
21A12	**23**	18
21A12	**53**	14
34B10	**25**	23



Sous-interrogations non synchronisées **multi-lignes** (1)

Prix moyen des produits commandés dans la commande 12A

Produits dont le stock est inférieur à tous les stocks des produits de catégorie A

Attention, l'opérateur NOT IN retourne FALSE si un membre ramené par la sous-interrogation est NULL

Sous-interrogations dans la clause FROM

- Possibilité de construire de manière dynamique la table sur laquelle porte le `SELECT`

Catégories comprenant plus de 2 produits

```
SELECT categorie
FROM (SELECT categorie, COUNT(*) as nb
      FROM Produit
      GROUP BY categorie)
WHERE nb>2;
```

Sous-interrogations **synchronisées** (1)

- Principe
  - Sous-interrogation mono ou multi-lignes
  - Exécutée pour chaque ligne retournée de la sous-interrogation vers l'interrogation de niveau supérieur
  - Exécution équivalente à des boucles imbriquées
- Forme générale

```
Select alias1.col1
FROM nomTable1 alias1
WHERE col(s) opérateur (SELECT alias2.col2
                        FROM nomTable2 alias2
                        WHERE alias1.x opérateur alias2.y)
[AND (condition)];
```

Sous-interrogations synchronisées (2)

Produits ayant un stock supérieur au stock moyen des produits de leur catégorie

Dans chaque commande, les produits commandés avec la plus grande quantité

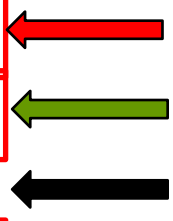
```
SELECT ---------------
FROM Produit p, LignComm ---
WHERE p. RefProd=------. RefProd
AND   Quantite = (SELECT MAX(Quantite)
                  FROM LignComm cl
                  WHERE cl.RefComm=---.RefComm);
```

Sous-interrogations synchronisées (2)

Produits ayant un stock supérieur au stock moyen des produits de leur catégorie

```
SELECT *
FROM ----------
WHERE Stock > (SELECT --------  FROM ----------
                WHERE ---------- = --------- );
```

RefProd	Catégorie	Prix	Stock
18	A	34	58
42	B	20	200
45	A	60	609
12	C	50	50
23	B	10	100



Sous-interrogations synchronisées et l'opérateur [NOT] EXISTS (1)

- Principe de l'opérateur [NOT] EXISTS
  - Retourne une valeur VRAI ou FAUX
    - **Vrai** : si la sous-interrogation retourne au moins un tuple
    - **Faux** : si la sous-interrogation ne retourne aucun tuple
  - NOT EXISTS → ensemble vide
  - Permet de mettre en œuvre la **division**

```
SELECT col1, col2,…
FROM Table t1
WHERE [NOT] EXISTS
 (SELECT {col | valeur}
  FROM Table t2
  WHERE t1.col1=t2.col2);
```

Sous-interrogations synchronisées et l'opérateur [NOT] EXISTS (2)

Codes et noms des clients ayant passé au moins une commande

Références des produits n'ayant jamais été commandés

Retour sur les opérateurs ensemblistes : Division (1)

- Définition

Structure de T2 incluse dans T1

La division d'une (partie de) table *T1[a1,…,an,b1,…bm]* par la (partie de) table *T2 [b1,…bm]* donne une table *T3 [a1,…,an]* qui comprend les lignes *li* telles que, $\forall lj \in T3$, *on a* $li,lj \in T1$

- Deux cas de division
  - Division inexacte : inclusion des ensembles
  - Division exacte : égalité des ensembles

- N'existe pas comme opérateur prédéfini, peut être exprimée :
  - Des requêtes imbriquée et/ou
  - Opérateur EXISTS

Les opérateurs ensemblistes : Cas de la division (2)

- Les commandes ayant commandé de TOUS les produits

RefCom
21A12

RefCom	RefProd	Quantite
21A12	23	18
21A12	18	10
21A12	25	14
34B13	25	23

RefProd	Designation	Categorie	Prix	Stock
18	Tabouret	A	16,78	200
23	Chaise	B	23,00	350
25	Table	B	123,00	36

Retour sur les opérateurs ensemblistes : Division (3)

- *Division inexacte* : les commandes ayant commandé **de tous** les produits de la catégorie 'A'

Parcours de toutes les commandes

```
SELECT DISTINCT RefComm FROM Commande c
      WHERE NOT EXISTS
 (SELECT RefProd FROM Produit WHERE Categorie='A'
  MINUS
  SELECT DISTINCT(RefProd) FROM LignComm l
        WHERE l.Refcomm=c.Refcomm);
```

Ensemble à comparer

Ensemble des produits de référence

Retour sur les opérateurs ensemblistes : Division (4)

- *Division exacte* : les commandes ayant commandé **uniquement** de tous les produits de la catégorie 'A'

Parcours de toutes les commandes

```
SELECT RefCom FROM Commande c
       WHERE NOT EXISTS
 (SELECT RefProd FROM Produit WHERE Categorie='A'
  MINUS
SELECT DISTINCT(RefProd) FROM LignComm l
        WHERE l.Refcomm=c.Refcomm)
AND NOT EXISTS
(SELECT DISTINCT(RefProd) FROM LignComm l
        WHERE l.Refcomm=c.Refcomm
MINUS
SELECT RefProd FROM Produit WHERE Categorie='A')
```

A-B

B-A

Les opérateurs ensemblistes : Cas de la division (4)

- Les commandes ayant commandé de **tous** les produits

```
SELECT RefCom
FROM LignComm
GROUP BY Refcom
HAVING COUNT(DISTINCT(RefProd))=(SELECT COUNT(RefProd)
                                 FROM Produit);
```