DICJ

Département d'informatique

Révision bases de données et SQL

420-JQA-JQ

Les bases de données

- Les bases de données sont apparues afin de résoudre les problèmes de doublons, d'intégrité et de sécurité des données!
- Au fil des années différents modèles ont existés: hiérarchique, réseau puis…. relationnel
- Le modèle relationnel est maintenant le plus largement répandu
- Par modèle, on entend la façon utilisée pour conserver les données et les relier entre elles.

Qu'est-ce qu'une base de données

- C'est une collection de données enregistrées sans redondances inutiles, pouvant être utilisées par plusieurs applications.
- Les données y sont enregistrées de telle manière qu'elles soient indépendantes des programmes qui les utilisent.
- Ces données sont structurées pour permettre des opérations, parfois très complexes, de lecture, suppression, déplacement, tri, comparaison, etc.
- Une base de données peut être locale, sur un ordinateur, ou répartie, sur un ou plusieurs serveurs.

Qu'est-ce qu'un SGBD

- Toutes les opérations sur une base de données sont permises grâce au SGBD: Système de Gestion de Bases de Données.
- C'est un ensemble de programmes (logiciel) qui gèrent la base de données et qui permet de définir, de mémoriser, de manipuler les données et d'en assurer la sécurité et la confidentialité dans un environnement multi usagers.
- Différents modèles de SGBD existent, mais depuis de nombreuses années, c'est le modèle relationnel (SGBDR) qui s'est imposé comme standard.
- Les logiciels les plus connus du marchés sont tous de ce type (Access, Oracle, SQLServer, Informix, Sybase, Filemaker, MySQL, DB2, Paradox, etc.).

Les avantages d'un SGBD

- **Indépendance physique**
  Accès aux données indépendant du support matériel où elles sont stockées.
- **Indépendance entre les données et traitements**
  Évolution des structures de données sans répercussion sur les programmes utilisant ces données.
- **Diminution de la redondance**
  Chaque donnée n'est stockée qu'une seule fois. Diminue l'espace mémoire sur disque.
- **Intégrité et cohérence des données**
  Cohérence des données puisqu'elles ne sont stockées une seule fois.
- **Partage des données**
  Accessibilité des données par plusieurs utilisateurs en même temps. Le système gère ces demandes à l'aide d'une file d'attente optimisée.
- **Sécurité et confidentialité des données**
  Gestion de profils utilisateur associés à des droits d'actions sur la base.

L'organisation d'une base de données

- **Table :** est un ensemble de données.
- **Enregistrement (ligne)** Un groupe de champs reliés entre eux, telles que toutes les données sur un client.
- **Champ (colonne)** Le plus petit élément d'information d'une base de données, comme le nom, la ville, le pays ou le numéro de téléphone d'un client.
- **Clé primaire (Primary Key PK)** Champ qui contient une information identifiant uniquement chaque enregistrement.
- **Clé étrangère (Foreign Key FK)** Champ d'une table représentant une relation entre celle-ci et la clé primaire d'une autre table.

Par où commencer?

- Les bases de données ont donc plusieurs avantages mais pour en profiter…. elles doivent être bien structurées.
- Pour s'assurer d'un bon résultat, il faut donc passer par l'étape qu'on appelle la modélisation.
- Cette modélisation ressemble un peu aux plans d'un architecte et comporte plusieurs niveaux.

Rappel - Modèle conceptuel

- Le modèle conceptuel est une représentation de la réalité pour laquelle on désire développer une BD.
- Il doit être utilisable avec un client.
- Par conséquent, il doit être exempt de détails techniques.
- On utilise un sous-ensemble des possibilités du diagramme de classes de UML qu'on appelle le diagramme de classe d'analyse.

Rappel – Conceptuel au relationnel

- Transformer le modèle conceptuel pour être en mesure de le convertir en BD.
- Les bases de données relationnelles ne permettent pas de représenter tel quel un diagramme de classe.
- On doit convertir les associations et les classes d'association.

Modèle relationnel

- C'est un modèle de données découlant d'un modèle conceptuel.
- Il se rapproche de la représentation des données dans la base de données.
- On le réalise en appliquant de simples règles sur le modèle conceptuel.

En résumé!

- Les entités deviennent des tables.
- Les identifiants deviennent des clés primaires (PK).
- Les attributs deviennent les champs des tables.
- Les relations de type un à plusieurs et un à un deviennent des clés étrangères (FK).
- Les relations de type plusieurs à plusieurs deviennent les tables d'intersection contenant une clé primaire composée (PK) et des clés étrangères (FK).

Le langage SQL

- Le langage SQL (Structured Query Language) est un langage normalisé servant à exploiter des bases de données relationnelles
- Il contient des instructions de définition de données (DDL), de manipulation de données (DML) et de contrôle de données et de contrôle des transactions (DCL).
- Pour ce cours, nous nous en tiendrons au DDL et au DML.
- SQL utilise très peu de mots-clés.

Définir la structure d'une BD

- Créer une BD
- Créer une table et ses colonnes
  - Les types de données
  - Les contraintes:
    - Colonne obligatoire/facultative
    - Clés primaires
    - Numéro séquentiel généré
    - Clés étrangères
    - Unique
    - Valeurs par défaut
- Modifier une BD
- Modifier une table
- Supprimer une table
- Supprimer une BD

Créer une base de données

Syntaxe:

CREATE DATABASE BDNom;

Exemple:

CREATE DATABASE BDPubs;

Se positionner sur une base de données

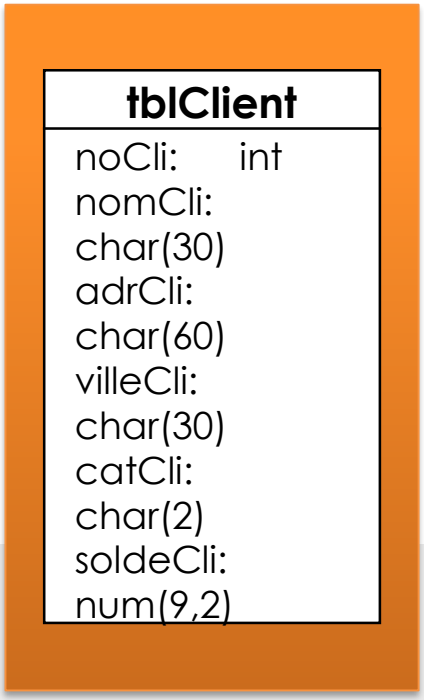
Syntaxe:

```
USE BDNom;
```

Exemple:

```
USE BDPubs;
```

Créer une table et ses colonnes




```
CREATE TABLE tblClient
(     noCli             int,
      nomCli            varchar(30),
      adrCli            varchar(60),
      villeCli          varchar(30),
      catCli            char(2),
      soldeCli          decimal(9,2),
      PRIMARY KEY (noCli)
);
```

TYPE DE DONNÉES

Les contraintes – obligatoire/facultative

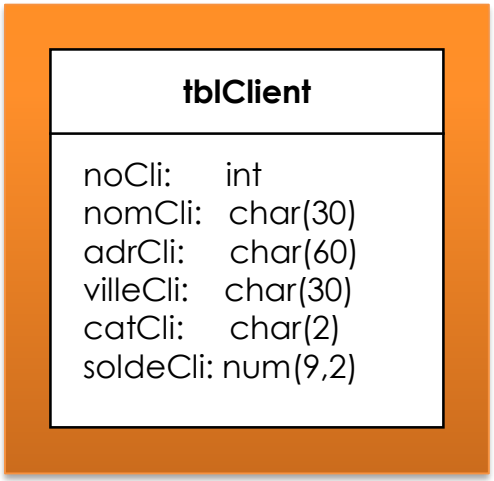
- Une colonne est **facultative** par défaut.
- Il faut donc déclarer explicitement les colonnes **obligatoires.**

```
CREATE TABLE tblClient
(   noCli           int             NOT NULL,
    nomCli          varchar(30)     NOT NULL,
    adrCli          varchar(60)     NOT NULL,
    villeCli        varchar(30),
    catCli          char(2)         NULL,
    soldeCli        decimal(9,2)    NOT NULL,
    PRIMARY KEY (noCli)
);
```

Colonnes facultatives

Colonne obligatoire

Les contraintes - Clé primaire (PK)




```
CREATE TABLE tblClient
(   noCli           int             NOT NULL,
    nomCli          varchar(30)     NOT NULL,
    adrCli          varchar(60)     NOT NULL,
    villeCli        varchar(30)     NOT NULL,
    catCli          char(2)         NULL,
    soldeCli        decimal(9,2)    NOT NULL,
    PRIMARY KEY (noCli)
);
```

Numéro séquentiel généré

- AUTO_INCREMENT permet de générer automatiquement un numéro séquentiel pour la « *clé primaire* ».

```
CREATE TABLE tblPersonne
(  noPers      smallint unsigned AUTO_INCREMENT,
   nomPers     varchar(40),
   prenPers    varchar(40),
   adrPers     tinytext,
   telPers     char(12),
   PRIMARY KEY(noPers)
);
```

- La numérotation débute à 1 et le pas est de 1.

PERSONNE

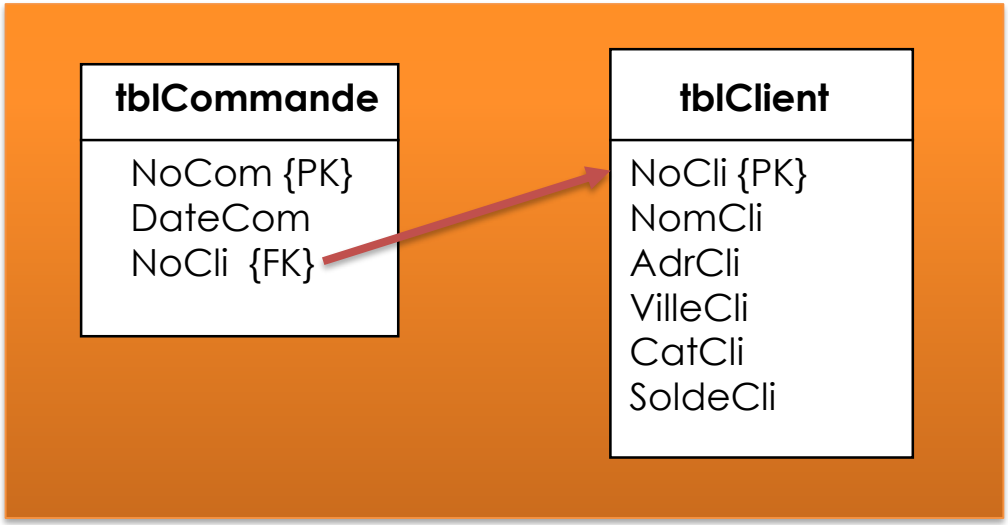
NoPers	NomPers	PrenPers	AdrPers	TelPers
1	Dupond	Marc	8 rue du Pont	418-544-5454
2	Dupont	Pierre	14 boul Dupond	418-555-4444

Les contraintes - Clé primaire COMBINÉE

- La clé primaire peut être composée de plusieurs champs

```
CREATE TABLE tblPersonne
(   nomPers     varchar(40),
    prenPers    varchar(40),
    adrPers     tinytext,
    telPers     char(12),
    PRIMARY KEY(nomPers, prenPers)
);
```

Les contraintes - Clé étrangère (FK)




```
CREATE TABLE tblCommande
(   noCom       char(12)  NOT NULL,
    noCli       char(10)  NOT NULL,
    dateCom     date      NOT NULL,
    PRIMARY KEY (noCom),
    FOREIGN KEY (noCli) REFERENCES tblClient(noCli)
);
```

Les contraintes - UNIQUE

- Pour interdire l'apparition de doublons pour un champ, on utilise la contrainte UNIQUE.

UNIQUE (champ ou *liste de champs*)

UNIQUE (**NAS**)

nomPers	prenPers	NAS
Dupond	Marc	**221-445-789**
Martin	Marc	**784-555-123**

```
CREATE TABLE tblPersonne
(  noPers      smallint unsigned AUTO_INCREMENT,
   nomPers     varchar(40),
   prenPers    varchar(40),
   NAS         char(11) UNIQUE,
   PRIMARY KEY(noPers)
);
```

Les contraintes – UNIQUE combiné

- Pour interdire les doublons d'une combinaison de champs, on liste l'ensemble des champs dans une seule commande **UNIQUE**.

  Exemple: pour interdire tout doublon du couple *nom*, *prénom*:

  UNIQUE(nomPers,prenPers)

nomPers	prenPers
Dupond	Marc
Dupont	Pierre
Martin	**Marc**
Martin	Pierre
Martin	**Marc**

Enregistrement interdit car le couple ('Martin', 'Marc') existe déjà

Contrainte Valeur par défaut

- La contrainte DEFAULT permet de déterminer la valeur qui sera assignée à la colonne si on ne donne pas de valeur spécifique
- Les valeurs par défaut sont généralement déclarées directement sur la ligne de définition du champ

```
CREATE TABLE tblClient
(  noClient     char(10)     NOT NULL,
   nomClient    varchar(30)  NOT NULL,
   adrClient    varchar(60)  NOT NULL,
   villeClient varchar(30)  NOT NULL DEFAULT 'Québec',
   catClient    char(2),
   soldeClient decimal(9,2)NOT NULL DEFAULT 0.0,
   PRIMARY KEY (noCli),
   UNIQUE (nomCli)
);
```

Un exemple

- Il est possible de mettre la contrainte sur la ligne de déclaration de la colonne, mais attention aux clés combinées.

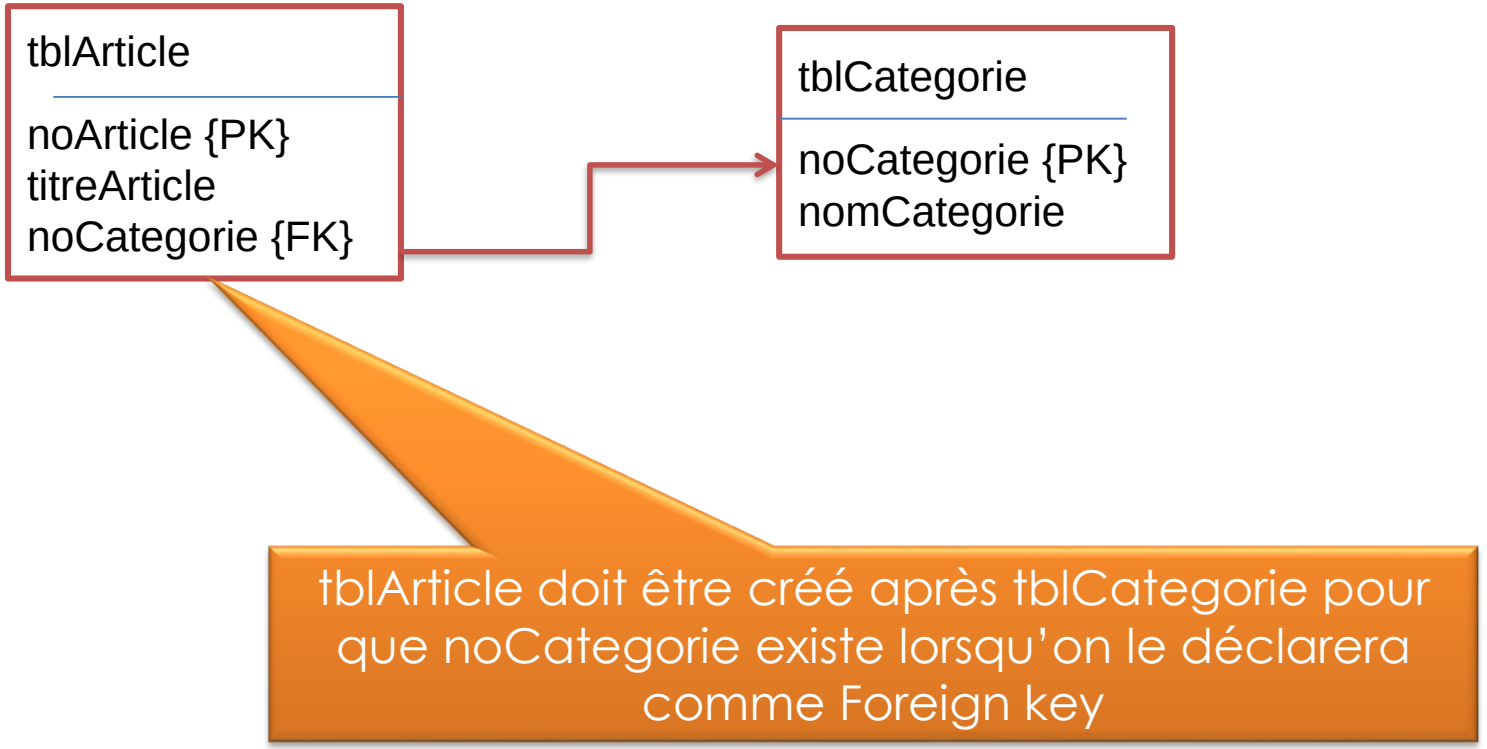
```
CREATE TABLE tblCompagnie
(  noCie         int                NOT NULL PRIMARY KEY AUTO_INCREMENT,
   nomCie        varchar(30)        NOT NULL UNIQUE,
   adrCie        varchar(60)        NOT NULL,
   villeCie      varchar(30)        NULL DEFAULT 'Québec',
   catCie        char(2),
   valeurBourse decimal(9,2)        NOT NULL
);

CREATE TABLE tblCompagnie
(  noCie      int                   NOT NULL AUTO_INCREMENT,
   nomCie     varchar(30)           NOT NULL,
   adrCie     varchar(60)           NOT NULL,
   villeCie   varchar(30)           NULL DEFAULT 'Québec',
   catCie     char(2),
   valeurCie decimal(9,2)           NOT NULL,
   PRIMARY KEY (noCie),
   UNIQUE (nomCie)
);
```

SAUF POUR LES CLÉS COMBINÉES

- Le champ déclaré UNIQUE peut accepter ou non les valeurs nulles.

Ordre de création

- L'ordre de création des tables est important.
- On ne peut référencer un champ qui n'est pas encore existant.



Supprimer une table

Syntaxe:

```
DROP TABLE NomTable;
```

Exemples:

```
DROP TABLE tblCommande;
DROP TABLE tblPersonne;
```

Attention, opération sous haute surveillance !

La table ne doit plus être référencée par une clé étrangère

Supprimer une base de données

Syntaxe:

```
DROP DATABASE NomBD;
```

Exemples:

```
DROP DATABASE BDPubs;

DROP DATABASE BDInventaire;
```

Attention ! **Tout** sera perdu!

S'assurer d'avoir un script de création

Modifier la structure d'une table

- Il est possible de modifier la structure et les contraintes d'une table déjà créée par la commande **ALTER TABLE**.
- Voici ce qu'il est possible de réaliser :
  - ajouter/supprimer une colonne
  - créer/supprimer une clé primaire
  - créer/supprimer une clé étrangère
  - changer la valeur par défaut d'un champ
  - ajouter une contrainte d'unicité (interdire les doublons)
  - modifier un champ (type de données, longueur, contraintes)
  - changer totalement la définition d'un champ

Ajouter un champ

- Il est possible d'ajouter une colonne à une table après sa création.

Syntaxe:

ALTER TABLE tblNomTable **ADD** NomColonne *typededonnees*

- Exemple : ajout d'une colonne FAX qui est une chaîne de 10 caractères

ALTER TABLE tblPersonne **ADD** FaxPers char(10)

Supprimer un champ

- Attention, supprimer un attribut implique la suppression des valeurs qui se trouvent dans la colonne qui correspond à cet attribut.

Syntaxe:

ALTER TABLE tblNomTable **DROP** NomChamp

Exemple : enlevez la colonne FAX

ALTER TABLE tblPersonne **DROP** FaxPers

Ajouter/Supprimer une clé primaire

- Il est possible d'ajouter une contrainte clé primaire à une colonne existante

```
ALTER TABLE tblNomTable ADD PRIMARY KEY (NomColonne,...)

Exemple:
ALTER TABLE tblPersonne ADD PRIMARY KEY (NoPers)
```

- Il est aussi possible de retirer la contrainte de clé primaire. Cette commande n'est pas si fréquemment utilisée et n'est pas aussi simple qu'elle peut le paraître car on doit respecter différentes règles quant au champ auto-incrémentée, aux foreign key, etc…

Ajouter une clé étrangère

- Il est possible d'ajouter une contrainte clé étrangère à une colonne existante

```
ALTER TABLE tblNomTable
ADD FOREIGN KEY (NomColonne) REFERENCES tblNomTable(NomColonne)

Exemple:
ALTER TABLE tblPersonne
ADD FOREIGN KEY (NoEmploi) REFERENCES tblEmploi(NoEmploi)
```

- Cette commande n'ajoute pas le champ NoEmploi à la table tblPersonne, elle ne fait qu'ajouter la contrainte de clé étrangère. Le champ NoEmploi doit déjà exister dans la table tblPersonne et dans la table tblEmploi.

Ajouter/Supprimer une valeur par défaut

- Ajouter une valeur par défaut

ALTER TABLE tblNomTable **ALTER** NomColonne **SET DEFAULT** *valeur*

Exemple:

ALTER TABLE tblPersonne **ALTER** provPers **SET DEFAULT** 'Québec'

- Supprimer une valeur par défaut

ALTER TABLE tblNomTable **ALTER** NomColonne **DROP DEFAULT**

Exemple:

ALTER TABLE tblPersonne **ALTER** provPers **DROP DEFAULT**

Ajouter/Supprimer une contrainte d'unicité

- Ajouter une contrainte d'unicité

```
ALTER TABLE tblNomTable ADD UNIQUE (NomColonne)

Exemple:
ALTER TABLE tblPersonne ADD UNIQUE (NomPers)
```

- Puisqu'une contrainte UNIQUE est d'abord un index, pour la supprimer il faut supprimer l'index en spécifiant son nom.

```
ALTER TABLE tblNomTable DROP INDEX NomContrainte

Exemple:
ALTER TABLE tblPersonne DROP INDEX NomPers
```

- Vous pouvez voir le nom donné à l'index en cliquant sur "+index" dans l'onglet structure. Ce nom est souvent le nom de la colonne.

Modifier un type de données

- Modifier seulement le type de données

ALTER TABLE tblNomTable **MODIFY** NomColonne *nouveautype*

Exemple:

- **ALTER TABLE** tblPersonne **MODIFY** NomPers varchar(40)

ALTER TABLE tblNomTable **MODIFY** NomColonne *nouveautype contrainte*

Exemple:
ALTER TABLE tblPersonne **MODIFY** NomPers varchar(40) NOT NULL

Modifier complètement un champ

- Modifier du même coup le nom, le type et les contraintes.

```
ALTER TABLE tblNomTable CHANGE AncienNomColonne
                    NouveauNomColonne NouveauType NouvellesContraintes

Exemple:
ALTER TABLE tblPersonne
          CHANGE Prenom PrenPers varchar(40) NOT NULL
```

- La partie NouvellesContraintes est facultative, mais même si nous souhaitons changer seulement le nom, il faut absolument fournir le type même s'il est semblable.

```
ALTER TABLE tblPersonne  CHANGE Nom NomPers varchar(25)
```

Renommer une table

- Nous pouvons modifier le nom d'une table.

```
RENAME TABLE tblNomTableTO tblNouveauNom

Exemple:
RENAME TABLE Personne TO tblPersonne
```

- Le nom est automatiquement modifié dans les références des FOREIGN KEY pointant sur la table.

Réinitialiser un champ auto incrémenté

- Pour réinitialiser la valeur d'un champ auto incrémenté, il existe une commande ALTER TABLE

```
ALTER TABLE tblPersonne  AUTO_INCREMENT = 1;
```

Ajouter une contrainte CHECK

- La contrainte CHECK permet de spécifier des valeurs ou des plages de valeurs pour un champ donné.

ALTER TABLE nom de la table
ADD CHECK (nom du champ condition)

Exemple
ALTER TABLE tblPersonne
ADD CHECK (taille < 190);

Ajout de données

- La commande permettant d'ajouter une ligne dans une table est **INSERT.**

```
INSERT INTO tblNomTable(liste des champs)
        VALUES(liste des valeurs);
```

```
INSERT INTO tblPersonne(nom, prenom, age)
        VALUES('Gagnon', 'Jean', 22);
```

- La liste des champs contient les champs de la table pour lesquels on souhaite fournir des valeurs.
- Les caractères et les dates doivent être entre apostrophes mais pas les valeurs numériques.

Deux formats

- On effectue une commande **INSERT** complète pour chaque ligne à ajouter.

```
INSERT INTO tblPersonne(nom, prenom, age)
        VALUES('Gagnon', 'Jean', 22);
INSERT INTO tblPersonne(nom, prenom, age)
        VALUES('Perron', 'Julie', 25);
```

- On peut utiliser la version abrégée en séparant les groupes de valeurs par des virgules

```
INSERT INTO tblPersonne(nom, prenom, age) VALUES
    ('Gagnon', 'Jean', 22),
    ('Perron', 'Julie', 25);
```

Règles pour l'ajout

- Si certains champs sont omis dans la liste des champs, alors les valeurs par défaut définies dans le script de création seront utilisées (DEFAULT).
- S'il n'y a pas de valeurs par défaut et que NULL est permis, la donnée contiendra NULL.
- S'il n'y a pas de valeurs par défaut et que le champ est NOT NULL, la valeur suivra les règles suivantes:

Valeur	Type donnée
0	pour un nombre
' '	pour une chaîne
0000-00-00	pour une date
00:00:00	pour une heure

Chaînes de caractères et dates

- Lorsqu'un champ contient une chaîne de caractères on doit utiliser des apostrophes.
- Si cette chaîne contient elle-même un apostrophe, on le double.

```
INSERT INTO tblCateg(CodeCateg, DescCateg, dateInscr)
VALUES('Da', 'Demande d''admission', '2019-03-06');
```

- **Attention**: il s'agit bien de 2 apostrophes de suite et non de guillemets.
- Ou le mettre entre guillemets

```
INSERT INTO tblCateg(CodeCateg, DescCateg, dateInscr)
VALUES('Da', "Demande d'admission", "2019-03-06");
```

Ajout sans noms de colonnes

- Il existe une syntaxe plus courte mais avec laquelle des valeurs doivent être fournies pour tous les champs de la table.

INSERT INTO tblNomTable **VALUES**(*valeurs dans l'ordre des champs*);

- Soit la définition de table suivante:

```
CREATE TABLE tblBallon
(       idBallon        TINYINT         NOT NULL,
        taille          TINYINT         NOT NULL,
        couleur         VARCHAR(20)     NULL,
        PRIMARY KEY(idBallon)
);
```

```
INSERT INTO tblBallon VALUES(1, 20, 'rouge');          bon
INSERT INTO tblBallon VALUES(1, 'rouge', 20);          mauvais
INSERT INTO tblBallon VALUES('rouge' );                mauvais
INSERT INTO tblBallon VALUES(1, 20, NULL);             bon
```

Ajout avec une clé auto incrémentée

- Lorsque la clé primaire est auto incrémentée, deux méthodes peuvent être utilisées pour les chargement initial de la base de données:
  - On peut laisser le système générer automatiquement la clé en utilisant un INSERT sans fournir de valeurs pour la clé
  - On peut forcer le numéro de la clé en le fournissant dans les instructions INSERT

Des exemples suivent!

Sans fournir le nom du champ de la clé

- La liste des champs de l'instruction INSERT ne contient pas la clé
- Un numéro séquentiel sera généré automatiquement

```
CREATE TABLE tblMembre
(           noMembre        tinyint     AUTO_INCREMENT,
            nomMembre       varchar(25),
            prenMembre      varchar(25),
            PRIMARY KEY(noMembre)
);

INSERT INTO tblMembre(nomMembre, prenMembre)
            VALUES('Bouchard', 'Julie');

INSERT INTO tblMembre(nomMembre, prenMembre) VALUES
            ('Fortin', Lucie'), ('Tremblay', 'Jonathan'), ('Bouchard', 'Annie');
```

noMembre n'est pas ici

ni là

Sans fournir le nom des colonnes

- L'instruction INSERT ne contient pas les noms des colonnes.
- Vous devez utiliser le mot-clé DEFAULT à la place de la valeur de la clé.

```
CREATE TABLE tblMembre
(           noMembre        tinyint     AUTO_INCREMENT,
            nomMembre       varchar(25),
            prenMembre      varchar(25),
            PRIMARY KEY(noMembre)
);
INSERT INTO tblMembre
VALUES(DEFAULT, 'Bouchard', 'Julie');
```

En fournissant la valeur de la clé

- La liste des champs de l'instruction INSERT contient la clé
- La clé prendra la valeur fournie, à condition qu'elle n'existe pas déjà dans la table, mais, par la suite, si on ajoute des données via un programme, le numéro augmentera automatiquement

```
CREATE TABLE tblMembre
(           noMembre        tinyint     AUTO_INCREMENT,
            nomMembre       varchar(25),
            prenMembre      varchar(25),
            PRIMARY KEY(noMembre)
);
INSERT INTO tblMembre(noMembre, nomMembre, prenMembre)
            VALUES(2, 'Bouchard', 'Julie');
```

noMembre est ici

et là

Attention aux FOREIGN KEY

- Vous devez vous même lier les données en inscrivant des valeurs appropriées dans les champs des FOREIGN KEY
- Soit un membre relié à une seule succursale

```
CREATE TABLE tblMembre
(       noMembre        smallint        NOT NULL AUTO_INCREMENT,
        nom             varchar(25)     NOT NULL,
        prenom          varchar(25)     NOT NULL,
        noSucc          tinyint         NOT NULL,
        PRIMARY KEY(noMembre),
        FOREIGN KEY(noSucc) REFERENCES tblSuccursale(noSucc)
);

INSERT INTO tblMembre(nom, prenom, noSucc)
        VALUES( 'Bouchard', 'Julie', 107);
```

La succursale 107 doit déjà exister dans tblSuccursale

Les tables d'intersection

- Si un membre peut être relié à plusieurs succursales, nous aurons une table d'intersection donc deux FK à associer

```
CREATE TABLE tblMembreSuccursale
(       noMembre        smallint            NOT NULL,
        noSucc          tinyint             NOT NULL,
        PRIMARY KEY(noMembre, noSucc),
        FOREIGN KEY(noMembre) REFERENCES tblMembre(noMembre),
        FOREIGN KEY(noSucc) REFERENCES tblSuccursale(noSucc)
);

INSERT INTO tblMembreSuccursale VALUES
            ( 1, 107),
            ( 3, 79);
```

Le membre 1 sera associé à la succursale 107 et le membre 3 à la succursale 79

Extraction de données

- La commande permettant d'extraire une ou plusieurs lignes d'une table se nomme **SELECT**.

- Nous pouvons faire de l'extraction simple, c'est –à-dire de l'extraction sur une seule table ou de l'extraction complexe avec des opérateurs, des jointures permettant de faire des requêtes sur plusieurs tables et des regroupements de données.

- Nous nous concentrerons sur des requêtes simples pour commencer.

Extraction simple

- Une extraction simple utilise une seule table
- La requête **SELECT** renvoie un résultat sous la forme d'une table
- La syntaxe:

```
SELECT {liste_champs}
FROM {TABLE}
WHERE {conditions};
```

Les champs de la table qui seront présents dans le résultat de la requête

Spécifie dans quelle table la recherche doit être effectuée

Clause optionnelle où sont posées les conditions auxquelles les enregistrements doivent satisfaire pour être inclus dans le résultat.

Un exemple d'extraction Simple

```
SELECT noCli, nomCli, villeCli
FROM   tblClient;
```

noCli	nomCli	villeCli
B062	BERGERON	Québec
B112	HANSENNE	Gatineau
B332	MONTI	Alma
B512	GILLET	Montréal
C003	AVRON	Montréal
C123	MERCIER	Québec
C400	FERARD	Gatineau
D063	MERCIER	Montréal
F010	TOUSSAINT	Gatineau
F011	PONCELET	Montréal
F400	JACOB	Toronto
K111	VANBIST	Laval
K729	NEUMAN	Montréal
L422	FRANCK	Québec
S127	VANDERKA	Québec
S712	GUILLAUME	Lévis

```
SELECT *
FROM tblClient;
```

* liste tous champs de la table

Un exemple avec une condition

```
SELECT noCli, nomCli
FROM   tblClient
WHERE  villeCli = 'Montréal';
```

WHERE permet d'ajouter des conditions

noCli	nomCli
B512	GILLET
C003	AVRON
D063	MERCIER
F011	PONCELET
K729	NEUMAN

La liste des clients est plus courte

DISTINCT - Éliminer les doublons

```
SELECT villeCli
FROM   tblClient
WHERE  cat = 'C1';
```

villeCli
Gatineau
Québec
Gatineau
Québec
Québec

Certaines lignes sont en double

```
SELECT DISTINCT villeCli
FROM   tblClient
WHERE  cat = 'C1';
```

villeCli
Québec
Gatineau

DISTINCT permet d'enlever les doublons

ORDER BY - Mettre en ordre

- Ordre croissant ou **ASC**endant (par défaut)

```
SELECT DISTINCT villeCli
FROM tblClient
ORDER BY villeCli ASC
```

villeCli
Alma
Gatineau
Laval
Lévis
Montréal
Toronto
Québec

ASC est facultatif donc
`ORDER BY villeCli` donne le même résultat

- Ordre décroissant ou **DESC**endant

```
SELECT DISTINCT villeCli
FROM tblClient
ORDER BY villeCli DESC
```

villeCli
Québec
Toronto
Montréal
Lévis
Laval
Gatineau
Alma

Valeur Null

- L'absence de valeur est indiquée par un marqueur spécial, dit *valeur NULL*.
- Plusieurs situations sont possibles pour l'instant nous nous limiterons au cas où un champ est vide.
- Le champ a été prévu dans la table mais aucune valeur n'a été entrée pour un enregistrement donné.

Conditions utilisant des valeurs NULL

```
SELECT noCli
FROM   tblClient
WHERE  cat = NULL;
```

noCli

NULL ne peut être comparé à rien, même pas à lui-même ! C'est une valeur vide

```
SELECT noCli
FROM   tblClient
WHERE  cat IS NULL;
```

noCli
D063
K729

```
SELECT noCli
FROM   tblClient
WHERE  cat IS NOT NULL;
```

La bonne syntaxe

Conditions avec LIKE

```
SELECT noCli
FROM   tblClient
WHERE  cat LIKE 'B_';
```

'_' = un caractère quelconque

```
SELECT noProd
FROM   tblProduit
WHERE  nomProd LIKE '%SAPIN%';
```

'%' = une chaîne quelconque

'B_' ⟶ 'B1'
'Bd'
'B '

'%SAPIN%' ⟶ 'PL. SAPIN 200x20x2'
'Boite en SAPIN'
'SAPIN VERNI'

'B_' ⟶ 'xB'
'B'
'B12'

'%SAPIN%' ⟶ 'Boite en SA PIN'
'Achetez S A P I N !'

Modification de données

- La commande permettant de modifier des données dans une table est **UPDATE**.
  - **SET** permet de dire quels sont les champs à modifier et quelles sont les nouvelles valeurs;
  - **WHERE** permet de préciser les enregistrements à modifier.

```
UPDATE tblNomTable SET champ=valeur, ...
 [ WHERE condition ]
```

```
UPDATE tblPersonne  SET telPers = '418-555-2121'
WHERE nomPers = 'Gagnon' AND prenPers = 'Jean'
```

Modifications multiples

- Il est possible de modifier plusieurs champs par la même commande.

```
UPDATE tblPersonne SET telPers = '418-555-2121' ,
                       faxPers = '418-555-2020'
WHERE nomPers = 'Gagnon' AND prenPers = 'Jean'
```

```
UPDATE  tblEnfant  SET  ageEnfant = ageEnfant +1
```

Suppression de données

- La commande permettant de supprimer des données dans une table est **DELETE.**

```
DELETE FROM tblNomTable
[ WHERE condition ]
```

-

```
DELETE FROM tblPersonne
WHERE nomPers = 'Gagnon' AND prenPers = 'Jean'
```

```
DELETE FROM tblPersonne
```

Opérateurs

- les parenthèses **( )**.
- les opérateurs arithmétiques (**+**, **-**, *****, **/**, **%**).
- les opérateurs logiques qui retournent **0** (faux) ou **1** (vrai) (**AND**, **OR**, **NOT**, **BETWEEN**, **IN**).
- les opérateurs relationnels (**<**, **<=**, **=**, **>**, **>=**, **<>**).
- les opérateurs et les fonctions peuvent êtres composés entre eux pour donner des expressions très complexes.

Conditions avec opérateurs

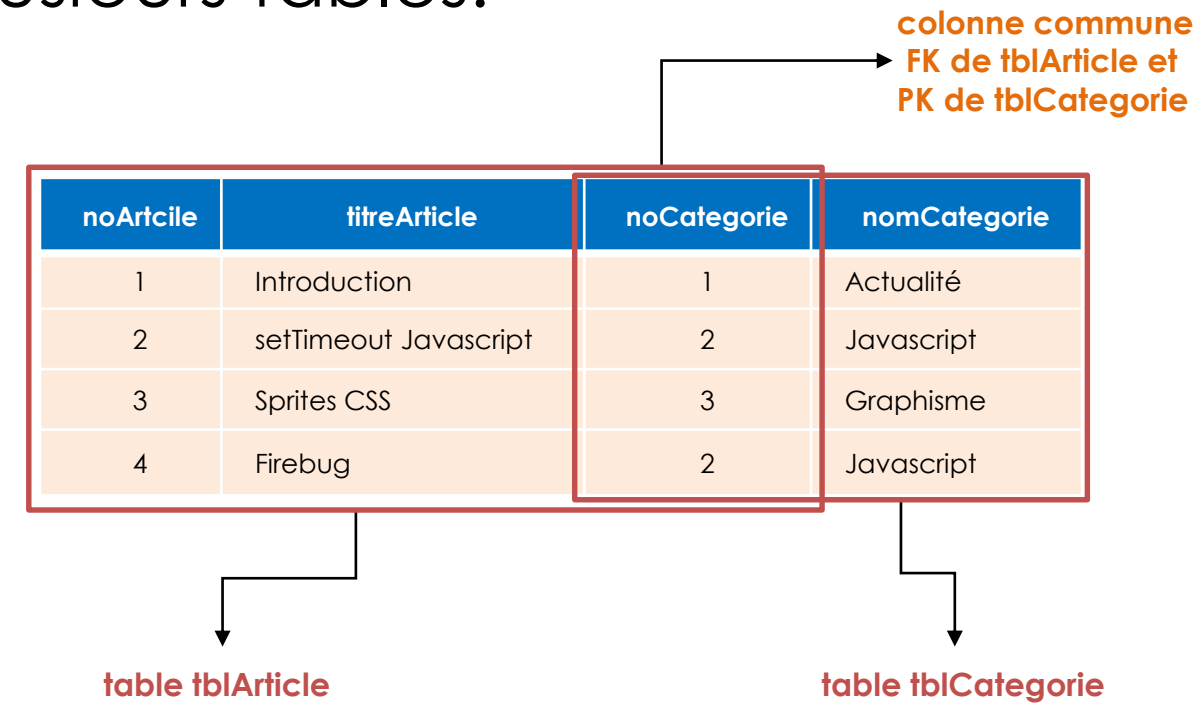
```
SELECT noCli
FROM   tblClient
WHERE  compte BETWEEN 1000 AND 4000;

SELECT nomCli, adrCli, compte
FROM   tblClient
WHERE  villeCli = 'Toulouse' AND compte  < 0;

SELECT nomCli, adrCli, compte
FROM   tblClient
WHERE  compte > 0 AND (cat = 'C1' OR villeCli =
  'Paris')
```

Les tables liées

- Dans un modèle relationnel, certaines tables sont liées entre elles à l'aide de colonnes ayant des valeurs communes.
- La clé étrangère d'une table est liée à la clé primaire d'une autre table.
- Par exemple, un article qui appartient à une catégorie

Le principe d'une jointure

- La jointure permet de produire une table temporaire constituée de données extraites de plusieurs tables.



Les différents types de jointures

- Nous avons deux tables : Table tblJeuxVideos et Table tblPropriétaires
- Les informations sont séparées dans des tables différentes et c'est bien. Cela évite de dupliquer des informations sur le disque. Cependant, lorsqu'on récupère la liste des jeux, si on souhaite obtenir le nom du propriétaire, il va falloir adapter la requête pour récupérer aussi les informations issues de la tblProprietaires. Pour cela, on doit faire ce qu'on appelle une jointure.
- Il existe plusieurs types de jointures, qui nous permettent de choisir exactement les données que l'on veut récupérer. Nous allons voir découvrir les deux plus importantes :
- les jointures internes : elles ne sélectionnent que les données qui ont une correspondance entre les deux tables ;
- les jointures externes : elles sélectionnent toutes les données, même si certaines n'ont pas de correspondance dans l'autre table.
- Il est important de bien comprendre la différence entre une jointure interne et une jointure externe.

Jointure interne - JOIN

- Cette façon de joindre les tables est réalisée en utilisant le mot réservé JOIN et en spécifiant la condition de jointure après le mot réservé ON

Les tables liées par le JOIN

```
SELECT *
FROM tblArticle
JOIN tblCategorie
     ON tblArticle.noCategorie = tblCategorie.noCategorie
```

La condition de jointure: les champs devant avoir la même valeur pour que l'enregistrement soit retenu

Alias

- Lorsqu'on fait une jointure il faut souvent préciser la table à laquelle on fait référence.
- Pour simplifier l'écriture on utilise souvent les alias pour renommer les tables.

```
SELECT *
FROM  tblArticle Ar
JOIN  tblCategorie Ca
      ON Ar.noCategorie = Ca.noCategorie
```

Ou

```
SELECT *
FROM  tblArticle AS Ar
JOIN  tblCategorie AS Ca
      ON Ar.noCategorie = Ca.noCategorie
```

Jointure externe – LEFT JOIN – RIGHT JOIN

- Les jointures externes permettent de garder l'une ou l'autre ou les deux tables complètes.

```
SELECT *
FROM personnes p
LEFT OUTER JOIN ingenieurs i ON i.personne_id =
p.personne_id;
```

Avec cette requête, on récupère toutes les données de la première table et celles conjointes de la deuxième table.

```
SELECT *
FROM ingenieurs i
RIGHT OUTER JOIN personnes p ON p.personne_id = i.
personne_id;
```

Avec cette requête, on récupère toutes les données de la deuxième table et celles conjointes de la première table.

Jointure externe – FULL OUTER JOIN

```
SELECT *
FROM personnes p
FULL OUTER JOIN ingenieurs i ON i.personne_id =
p.personne_id;
```

- Avec cette requête, on récupère l'intégralité des données des deux tables. Cependant, nous aurons plusieurs lignes incomplètes.