# Le langage SQL

- Langage de définition de données (LDD) et de manipulation de données (LMD) des bases de données relationnelles
- S'appuie sur l'algèbre relationnel
- Langage de définition de données permet de créer, modifier, supprimer les éléments du schéma relationnel
- Langage de manipulation de données permet l'interrogation, l'ajout, la modification, la suppression de données

# Les instructions SQL

- SELECT : Extraction de données

- INSERT
- UPDATE : Manipulation de données (LMD)
- DELETE
- MERGE

- CREATE
- ALTER
- DROP : Langage de définition de données (LDD)
- RENAME
- TRUNCATE

- COMMIT
- ROLLBACK :Contrôle des transactions
- SAVEPOINT

- GRANT : Langage de contrôle des données
- REVOKE

# SELECT élémentaire

```
SELECT liste_colonnes
   FROM liste_tables;
```

- Les expressions arithmétiques
- Les alias
- La valeur NULL
- Les doublons

# Limiter et trier les données

```
SELECT liste_colonnes
   FROM liste_tables
   [WHERE condition(s)];
```

- La clause WHERE limite l'interrogation aux lignes qui remplissent les conditions mentionnées.
- Les opérateurs : =, >, >=, <, <=, <>

# La condition BETWEEN

- Permet d'afficher les lignes en fonction d'une plage de valeurs

```
SELECT nom, salaire
   FROM employés
   WHERE salaire BETWEEN 2500 AND 3500;
```

# La condition IN

- Vérifie l'appartenance d'une donnée à une liste de valeur

```
SELECT nom
   FROM employés
   WHERE ville IN ('TOULOUSE', 'BLAGNAC');
```

# La condition LIKE

- Recherche les chaînes de caractères valides à l'aide de caractères génériques
- % représente n'importe quelle séquence de 0 ou plusieurs caractères
- _ représente un seul caractère

```
SELECT nom
   FROM employés
   WHERE ville LIKE 'T%';
```

# Les conditions NULL

- Les conditions NULL comprennent les conditions IS NULL et IS NOT NULL
- Recherche des valeurs non attribuées

```
SELECT nom
   FROM employés
   WHERE ville IS NULL;
```

# Les conditions logiques

- AND : Renvoie TRUE si les deux conditions sont vraies
- OR : Renvoie TRUE si une des conditions est vraie
- NOT : Renvoie TRUE si la condition qui suit est FALSE

```
SELECT nom
   FROM employés
   WHERE salaire >= 1000
   AND ville LIKE '%ou%'
```

# Le tri

- La clause ORDER BY permet de trier les lignes

```
SELECT liste_colonnes
   FROM liste_tables
   [WHERE condition]
   [ORDER BY liste_colonnes2 [ASC|DESC]];
```

```
SELECT nom
   FROM employés
   WHERE salaire >= 1000
   ORDER BY salaire DESC;
```

# Les fonctions monolignes

- Ces fonctions renvoient un résultat par ligne, elles permettent de manipuler des éléments de données.
- Elles acceptent des paramètres et renvoient une seule valeur
- Elles peuvent être imbriquées
- On distinguent des :
  - Fonctions alphanumériques
  - Fonctions numériques
  - Fonction de dates
  - Fonction de conversions
  - Fonctions générales

# Fonctions alphanumériques

- LOWER(exp) : Convertie en minuscules
- UPPER(exp) : Convertie en majuscules
- CONCAT(exp1, exp2, ...) : Concatène
- SUBSTRING(exp, m, [n]) : Retourne une partie de la chaîne
- LENGTH(exp) : Nombre de caractères
- INSTR(exp, 'chaine') : Renvoie la position d'une chaîne

# Fonctions alphanumériques

- LPAD(exp, n, 'chaine') : Ajoute des caractères à gauche
- RPAD(exp, n, 'chaine') : Ajoute des caractères à droite
- TRIM(exp) : Retire les espaces avant et après
- REPLACE(exp, 'ch1', 'ch2') : Remplace ch1 par ch2

# Fonctions numériques

- ROUND(exp, n) : Arrondit à la décimale spécifiée
- TRUNCATE(exp, n) : Tronque à la décimale spécifiée
- MOD(exp1, exp2) : Renvoie le reste de la division de exp1 par exp2

# Fonctions de dates

- CURDATE() : Renvoie la date système au format 'yyyy-mm-dd'
- CURTIME() : Renvoie l'heure système au format 'hh:mm:ss'
- DATE_FORMAT(date, format) : Renvoie la date selon le format indiqué
  - %d : jour du mois (01-31)
  - %M : Mois en anglais
  - %Y : année (2006)

  %m : mois (01-12)

  %y : année (06)
- YEAR(date) :

# Les jointures

- Permettent d'afficher des données de plusieurs tables
- Sans conditions de jointure le produit cartésien est effectué

```
SELECT nom, libel
   FROM employés, services;
```

# Les jointures

- La condition de jointure s'écrit dans la clause WHERE

```
SELECT liste_colonnes
   FROM liste_tables
   WHERE table1.colonne1 = table2.colonne2;
```

- Le nom de table est nécessaire si le nom des colonnes est identique

```
SELECT nom, libel
   FROM employes, services
   WHERE employes.numService = services.numService;
```

# Alias de table

- Utilisation de préfixes désignant les tables
- Permet de simplifier les interrogations
- Améliore les performances
- Evite les ambiguïtés sur les noms de colonnes

```
SELECT e.nom, s.libel
   FROM employes e, services s
   WHERE e.numService = s.numService;
```

# Jointures externes

- Permettent de visualiser également les lignes qui ne correspondent pas à la condition de jointure
- On distingue la jointure externe gauche (LEFT JOIN) et la jointure externe droite (RIGHT JOIN)

```
SELECT e.nom, s.libel
   FROM employes e RIGHT JOIN services s
   ON e.numService = s.numService;
```

# Agréger des données

- L'objectif est de donner des informations statistiques sur un ensemble de ligne
- Les lignes seront regroupées grâce à la clause GROUP BY
- Les groupes pourront être exclus à l'aide à la clause HAVING
- Les fonctions de groupe donneront des résultats par goupe

# Les fonctions de multilignes

- AVG(exp) : moyenne des valeurs de l'expression
- COUNT(exp) : nombre de valeurs non nulles
- MAX(expr) : Valeur maximale
- MIN(expr) : Valeur minimale
- SUM(expr) : Somme des valeurs de l'expression

# Création de groupes

- Les groupes sont créés par la clause GROUP BY

```
SELECT liste_colonnes
   FROM liste_tables
   [WHERE condition]
   [GROUP BY liste_colonnes3];
```

```
SELECT idService, count(nomEmp)
   FROM Employes
   GROUP BY idService;
```

# Exclure des groupes

- La clause HAVING limite les groupes de la même manière que les lignes pour WHERE

```
SELECT liste_colonnes
   FROM liste_tables
   [WHERE condition]
   [GROUP BY liste_colonnes3]
   [HAVING condition];
```

```
SELECT idService, sum(salaire)
   FROM Employes
   GROUP BY idService
   HAVING sum(salaire) > 1000;
```

# Sous-interrogation

- Instruction SELECT  imbriquée dans une clause d'une autre instruction SELECT
- Elle s'exécute en premier et son résultat est utilisé par la requête principale

```
SELECT liste_colonnes
   FROM liste_tables
   WHERE expr operateur (
      SELECT ....)
```

# Création d'une table

```
CREATE TABLE nom_table
   (nom_col1 type_col1 [DEFAULT valeur1] [contrainte_col1],
    nom_col2 type_col2 [DEFAULT valeur2] [contrainte_col2],
      ...
    contrainte_table1,
    contrainte_table2,
      ...);
```

```
CREATE TABLE tblClient
   (idClient type_col1 [DEFAULT valeur1] [contrainte_col1],
    nom_col2 type_col2 [DEFAULT valeur2] [contrainte_col2],
      ...
    contrainte_table1,
    contrainte_table2,
      ...);
```

# Contraintes

- Contraintes de colonnes et contraintes de tables matérialisent les contraintes d'intégrités posés sur le schéma relationnel et que le SGBD devra prendre en charge
- Contraintes de colonnes portent sur un attribut
- Contraintes de table portent sur plusieurs colonnes

# Contraintes

- Contrainte d'unicité (clause UNIQUE) permet de s'assurer qu'il n'existe pas de valeur dupliquée dans la colonne
- Contrainte d'obligation (clause NOT NULL) permet de vérifier qu'il existe une valeur pour chaque élément de la colonne
- Contrainte de clé primaire (clause PRIMARY KEY) a le même rôle que la clause UNIQUE mais ne peut être spécifié qu'une seule fois dans la table

# Contraintes

- Contrainte d'intégrité référentielle « colonne » (clause REFERENCES) matérialise une dépendance entre 2 colonnes de la table
- Contrainte d'intégrité référentielle «table» (clause FOREIGN KEY) matérialise une dépendance entre 2 colonnes de 2 tables
- Contrainte sémantique (clause CHECK) permet de spécifier des conditions logiques portant sur une ou plusieurs colonnes d'une même table

# Modification, suppression de table

```
ALTER TABLE nom_table modification;
```

- Permet de modifier la structure de la table

```
DROP TABLE nom_table;
```

- Permet de supprimer une table

```
RENAME ancien_nom TO nouveau_nom;
```

- Permet de renommer une table

# Définition de vues

```
CREATE VIEW nom_vue [alias1, alias2, ...] AS requete;
```

- Création d'une vue correspondant à la requête

```
DROP VIEW nom_vue;
```

- Suppression d'une vue

# Définition d'index

```
CREATE [UNIQUE] INDEX nom_index ON table
    (nom_col1 [ASC/DESC], nom_col2 [ASC/DESC], ...);
```

- Création d'un index

```
DROP INDEX nom_index;
```

- Suppression d'un index

# Manipulation de données

```
INSERT INTO nom_table
   [(nom_col1 [,nomcol2] ..)]
   VALUES (valeur1[,valeur2] ...);
```

- Insertion de ligne

```
UPDATE nom_table
   SET nom_col1 = expression1
   [,  nom_col2 = expression2] ...
   [WHERE condition];
```

- Modifications de lignes

```
DELETE FROM nom_table
   [WHERE condition];
```

- Suppression de lignes