

## Cours 4 et 5: Le langage SQL

- Origine
- Définition des structures
- Définition des contraintes
- Modification des structures
- Création d'index
- Création de vues
- Définition d'un schéma



Parallèle avec le monde objet

## Origine

### Structured Query Language

- caractéristiques des langages déclaratifs
- origine : IBM, System R, milieu des années 70
- implémenté dans de nombreux SGBD

Plusieurs versions :

- SQL1 initial : ANSI\* —1986
- SQL1 avec intégrité référentielle, ANSI —1989
- SQL2 ANSI —1992
- SQL3 ANSI — 1999 incorpore la notion d'objet

\* ANSI = American National Standard Institute

## Caractéristiques de SQL

- Fonctionnalités :
  - Définition des objets de la base de données (LDD)
  - Manipulation de données (LMD)
  - Contrôle des accès aux données
  - Gestion de transactions
- Utilisé par : DBA, développeurs, quelques utilisateurs



Parallèle avec MySQL

## Principales Instructions

- Définitions (LDD)  
*CREATE, DROP, ALTER*
- Mises à jour (LMD)  
*INSERT, UPDATE, DELETE*
- Interrogations (LMD)  
*SELECT*
- Contrôle d'accès aux données  
*GRANT, REVOKE*
- Gestion de transactions  
*COMMIT, ROLLBACK*



## Consultation des données

- Hypothèse:
  - un schéma de base de données est créé
  - Une base de données a été remplie
- La création a été faite par une interface QBE (mySQL)
- On expérimente la consultation avec SQL
- On expérimentera la création du schéma et le remplissage de la base avec SQL plus tard



**Langage algébrique en SQL**



**Requêtes mono et multi table(s)**

## Exemple pour les requêtes

### CLIENT

numéro	nom	adresse	numéro_téléphone
101	Durand	NICE	0493456743
108	Fabre	PARIS	NULL
110	Prosper	PARIS	NULL
125	Antonin	MARSEILLE	NULL

### PRODUIT

référence	marque	Prix HT
153	BMW	1000
589	PEUGEOT	1800
158	TOYOTA	1500

### VENTE

numéro	référence_produit	numéro_client	date
00102	153	101	12/10/04
00809	589	108	20/01/05
11005	158	108	15/03/05
12005	589	125	30/03/05

## Opérateur *Projection*

**SELECT xx yy** : définition du format du résultat  
+ spécifier la cible

- Afficher le nom et l'adresse des clients

```
SELECT nom , adresse  
FROM Client ;
```

- Afficher toutes les informations des clients

```
SELECT *  
FROM Client ;
```

## Elimination des doublons

- Afficher l'adresse de tous les clients

```
SELECT adresse  
FROM Client ;
```

```
SELECT ALL adresse  
FROM Client ;
```

- Afficher toutes les adresses existantes (sans doublons)

```
SELECT DISTINCT adresse  
FROM Client ;
```

Par défaut

Projection



Relationnel : On doit pouvoir distinguer chaque tuple !

SQL : le résultat peut ne pas être une relation

## Opérateur Sélection

....

+ spécifier la cible +

**where expression** : définition des tuples du résultat

- Quels sont les clients dont l'adresse est *Paris*

```
SELECT *  
FROM Client  
WHERE adresse = 'Paris';
```

- Quels sont les produits dont le prix TTC est supérieur à 1000

```
SELECT *  
FROM Produit  
WHERE prix_HT + prix_HT * 0.195 > 1000;
```

## Opérations possibles (mySQL)

- Booléennes

and, or, xor, =, !=, <, >, <=, >=

- Arithmétiques

+, -, \*, /

+, -

*opérateurs unaires*

- Fonctions numériques

abs, log, cos, sin, mod, power ...

- Arithmétiques sur date

+, -

- Fonctions sur chaînes

length, concat, ...

Un grand nombre



Dans **Select**: attribut calculé (résultat)

Dans **Where**: participer à la sélection

## Précédence des opérateurs

- :=
- ||, OR, XOR
- &&, AND
- BETWEEN, CASE, WHEN, THEN, ELSE
- =, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
- |
- &
- <<, >>
- -, +
- \*, /, DIV, %, MOD
- ^
- - (unary minus), ~ (unary bit inversion)
- !, NOT
- BINARY, COLLATE

+ faible

SELECT 1+2\*3;  
7

+ forte

## Utilisation des opérateurs

- SELECT ABS (-32); 32 *Bien sur on peut utiliser des attributs*
- SELECT FLOOR (1.23); 1
- SELECT MOD (234, 10); 4 *Peut se trouver dans Where*
- SELECT 253 % 7; 1
- SELECT ROUND (1.298, 1); 1,3
- SELECT ROUND (1.298, 0); 1
- SELECT SIGN (234) SIGN (-32) SIGN (0); 1 / -1 / 0
- SELECT 3 / 5; 0,60

WHERE 3 / 5 < 1;

## Utilisation des opérateurs (Chaînes)

- **SELECT** **CONCAT** ('My', 'S', 'QL');    **'MySQL'**
- **SELECT** **CHAR\_LENGTH** ('MySQL');    **5**
- **SELECT** **LOCATE** ('bar', 'foobarbar');    **4**
- **SELECT** **LOCATE** ('bar', 'foobarbar', 5);    **7**
- **SELECT** **INSERT** ('Quadratic', 3, 4, 'What');    **'QuWhattic'**
- **SELECT** **LOWER** ('MySQL');    **'mysql'**
- **SELECT** **SUBSTRING** ('Quadratically', 5, 6);    **'ratica'**
- **SELECT** 'David!' **LIKE** 'David\_';    **1**
- **SELECT** 'David!' **LIKE** '%D%v%';    **1**
- **SELECT** **STRCMP** (S1, S2);    **-1, 0, 1**

**\_ et %**

**0 (false), 1 (true)**

## Combinaison Sélection + projection

**SELECT** liste d'attributs ou \*

**FROM** liste des relations

**WHERE** condition ;

Attributs recherchés / calculés

Si plusieurs: il faut une jointure

Test sur chaque tuple: résultat vrai ou faux

Utilise des fonctions, opérateurs...

Autres clauses possibles

## Requêtes simples (1)

- Y a-t-il des produits dont le nom est « XBOX »  
**SELECT** \*  
**FROM** Produit  
**WHERE** nom = 'XBOX' ;
- Quels sont les ventes réalisés il y a plus de 30 jours?  
**SELECT** \*  
**FROM** Vente  
**WHERE** CURRENT\_DATE () > 30 + date ;
- Quels sont les ventes faites après le 1er janvier 2007?  
**SELECT** \*  
**FROM** Vente  
**WHERE** date > DATE ('2007-01-01') ;

## Requêtes simples (2)

- Quels sont les ventes dont le montant HT est entre 1000 et 3000 euros et dont le client n'est pas le numéro 101?  
**SELECT** \*  
**FROM** Vente  
**WHERE** (prix\_ht between 1000 and 3000) and  
(numero != 101);
- Quels sont les clients dont le nom est soit Prosper, soit Durand, soit Anthonin ?  
**SELECT** \*  
**FROM** Client  
**WHERE** nom in ('Prosper', 'Durand', 'Anthonin') ;



## Requêtes simples (3)

- Quels sont les clients dont le nom commence par 'P'

```
SELECT *  
FROM Employe  
WHERE nom LIKE 'P%';
```

- Quels sont les clients dont le nom commence par 'P' et a un 'S' comme 4<sup>ème</sup> lettre

```
SELECT *  
FROM Client  
WHERE nom LIKE 'P__S%';
```

## Requêtes et valeurs nulles (1)

- Quels sont les ventes dont la date de réalisation est inconnue?

```
SELECT *  
FROM Vente  
WHERE date is null ;
```

Ou : IS NOT NULL

### Attention :

WHERE date = NULL



Whatever comparé avec NULL  
→ Ni vrai ni faux

COUNT, MIN, SUM



Ignore les valeurs NULL  
→ sauf COUNT (\*)

Toute opération appliquée à NULL donne pour résultat NULL

## Requêtes et valeurs nulles (2)

```
SELECT nom  
FROM Produit  
WHERE prix_ht > 1000 ;
```

```
SELECT nom  
FROM Produit  
WHERE prix_ht > 1000 or prix_ht <= 2500 ;
```

Que se passe-t-il si certains produits  
ont un prix HT inconnu?

## Les autres clauses (tri)

```
SELECT attribut1..  
FROM  
WHERE expression  
ORDER BY attribut1:[ASC] [DESC]...
```

Affichage!

- Donner le numero, le prix HT et la marque des produits selon l'ordre décroissant des marques et l'ordre croissant des prix HT

```
SELECT marque, prix_ht, numero  
FROM Produit  
ORDER BY marque DESC, prix_ht ASC;
```

Ordre d'affichage!

Mais aussi:

```
ORDER BY 1 DESC, 2 ASC;
```

## Requêtes multi- tables (Opérateur *Jointure*)

### Deux points de vue:

- exécuter des **boucles imbriquées** (une table par boucle)  
appliquer la clause WHERE dans les boucles
- calculer le **produit cartésien** (une nouvelle table)  
Appliquer la clause WHERE sur chaque ligne

*Donner le nom d'un produit et le montant de la vente*

```
SELECT Produit.nom, Vente.prix_HT  
FROM Produit, Vente  
WHERE Vente.reference_produit = Produit.numero ;  
          Critères de jointure
```

***Il faut joindre les tables (jointure)***

## Définition de la jointure

```
SELECT Client.nom, Vente.prix_ht  
FROM Client, Vente ;
```

→ tous les tuples (*nom*, *prix\_ht*)

→ *nom* est un nom de client et *prix\_ht* est un prix de vente

**Pas de critère de jointure → Produit cartésien**

**Intérêt?**



**Mots-clés pour exprimer le critère de sélection**

## Jointure (compléments)

- Donner la marque des produits dont le prix HT est supérieur à celui d'une BMW

```
SELECT Produit.marque
FROM Produit Produit_reference Produit
WHERE Produit_reference.marque = 'BMW' AND
      Produit.prix_HT > Produit_reference.prix_HT ;
```

Renommage

Ou de même prix HT:

```
SELECT Produit.marque
FROM Produit Produit_reference, Produit
WHERE Produit_reference.marque = 'BMW' AND
      Produit.prix_ht = Produit_reference.prix_ht AND
      Produit.marque != Produit_reference.marque ;
```

## SQL et les différentes jointures

Jointure interne	SELECT ... FROM <table gauche> <b>[INNER] JOIN</b> <table droite> ON <condition de jointure>
Jointure externe	SELECT ... FROM <table gauche> <b>LEFT   RIGHT   FULL OUTER JOIN</b> <table droite> ON condition de jointure <i>dans mysql pas reconnu</i>
Jointure naturelle	SELECT ... FROM <table gauche> <b>NATURAL JOIN</b> <table droite> [USING <noms de colonnes>]
Jointure croisée	SELECT ... FROM <table gauche> <b>CROSS JOIN</b> <table droite> <i>dans mysql = INNER JOIN</i>
Jointure d'union	SELECT ... FROM <table gauche> <b>UNION JOIN</b> <table droite>

## Quelques jointures

- Donner le nom des clients et le prix HT

```
SELECT Client.nom, Vente.prix_ht  
FROM Vente, Client  
WHERE Vente.numero_client = Client.numero
```

Jointure naturelle

- Donner la marque des produits dont le prix HT est supérieur à celui d'une BMW

```
SELECT Produit.marque  
FROM Produit, Produit_reference, Produit  
WHERE Produit_reference.marque = 'BMW' AND  
Produit.prix_ht > Produit_reference.prix_ht ;
```

Theta Jointure

## Quelques jointures (compléments)

### Que deviennent les tuples non sélectionnés de R1 ou R2?

- dans la jointure (interne **INNER JOIN**) les tuples qui ne peuvent pas être joints sont éliminés du résultat
- dans la jointure externe (**OUTER JOIN**) les tuples qui ne peuvent pas être joints sont conservés dans le résultat

Pour les tuples de la relation de gauche (R1) et / ou de droite (R2)

Défaut

R1 **FULL OUTER JOIN** R2 : Remplit R1.\* et R2.\*

R1 **LEFT OUTER JOIN** R2 : Remplit R1.\*

R1 **RIGHT OUTER JOIN** R2 : Remplit R2.\*

avec **NULL** si nécessaire.

## Exercice (1)

### Relations :

- Journal (code-j, titre, prix, type, périodicité)
- Dépôt (no-dépôt, nom-dépôt, adresse)
- Livraison (no-dépôt, code-j, date-liv, quantité-livrée)

### Requêtes : donner...

- le prix des journaux livrés le 15/01/07 ?
- tous le nom des hebdomadaires reçus par le dépôt de Paris..
- les titre des journaux livrés à Nice.
- le nom des dépôts qui reçoivent des hebdomadaires dont la quantité livrée excède 100.

## Exercice (2)

```
SELECT *  
FROM LIVRAISON RIGHT JOIN (DEPOT, JOURNAL)  
ON (LIVRAISON.no-depot = DEPOT.no-depot AND  
    LIVRAISON.code-j = JOURNAL.code-j)
```

*Toutes les lignes de DEPOT et JOURNAL seront présentes  
Avec éventuellement rien pour la partie livraison*

```
SELECT *  
FROM LIVRAISON, DEPOT, JOURNAL  
WHERE LIVRAISON.no-depot = DEPOT.no-depot AND  
    LIVRAISON.code-j = JOURNAL.code-j)
```

*Seulement les lignes de DEPOT et JOURNAL qui  
correspondent à une livraison*

## Opérateur Union

Afficher la liste des numéros d'employé des responsables de département et des directeurs

Département

numéro-département	...	responsable
--------------------	-----	-------------

Employé

numéro-employé	...	fonction
----------------	-----	----------

```
SELECT responsable
FROM Département
UNION
SELECT numéro-employé
FROM Employé
WHERE fonction = 'Directeur';
```

## Opérateurs Intersection, Différence

Afficher les numéros d'employé des responsables de département qui sont aussi des directeurs

```
SELECT responsable
FROM Département
INTERSECT
SELECT numéro-employé
FROM Employé
WHERE fonction = 'Directeur';
```

```
SELECT responsable
FROM Département
EXCEPT
SELECT numéro-employé
FROM Employé
WHERE fonction = 'Directeur';
```

Afficher les numéros d'employé des responsables de département  
Sauf ceux qui sont aussi des directeurs

## Requêtes (Fonctions statistiques)

- SUM (*nom d'attribut*)
- COUNT(\*)
- COUNT(DISTINCT *nom d'attribut*)
- MAX (*nom d'attribut*)
- MIN (*nom d'attribut*)
- AVG (*nom d'attribut*)
- AVG (DISTINCT *nom d'attribut*)



Dans les clauses :

- SELECT
- HAVING TO

## Clause Group by

**SELECT** *attributs recherchés*  
**FROM** *liste des relations*  
**WHERE** *condition*  
**GROUP BY** *attributs de regroupement*  
[**HAVING** *condition sur le groupe*];

**SELECT** COUNT (\*)  
**FROM** Produit  
**GROUP BY** marque  
**SELECT** AVG (prix\_ht), ~~nom~~  
**FROM** Produit  
**GROUP BY** marque

**SELECT** COUNT (\*)  
**FROM** Produit  
**WHERE** marque <> 'BMW'  
**GROUP BY** marque  
**HAVING** AVG (prix\_ht) > 10.5



## Utilisation des fonctions statistiques

- Donner la moyenne des prix HT et les prix min. et max.

```
SELECT AVG (prix_ht), MIN (prix_ht), MAX (prix_ht)
FROM Produit
```

- Même chose mais par marque

```
SELECT AVG (prix_ht), MIN (prix_ht), MAX (prix_ht)
FROM Produit
GROUP BY marque
```

- Même chose mais par marque si la moyenne > 10,5

```
SELECT AVG (prix_ht), MIN (prix_ht), MAX (prix_ht)
FROM Produit
GROUP BY marque HAVING AVG (prix_ht) > 10.5
```

## Requêtes imbriquées (1)

```
SELECT liste d'attributs recherchés
FROM liste_des_relations
WHERE bloc SFW dans la condition
```

Intérêt: indiquer qu'un attribut doit prendre ses valeurs dans une liste de valeurs définies par un autre bloc SFW

- Itérations imbriquées
- autre façon de faire certaines formes de jointure
- Sous-requêtes indépendantes ou pas

## Requêtes imbriquées (2)

Principaux connecteurs: = != > < IN EXISTS ANY ALL

**EXISTS** *R* : retourne TRUE si *R* n'est pas vide, FALSE sinon

*t* **IN** *R* : retourne TRUE si *t* appartient à *R*, FALSE sinon

valeur comp **ANY** *R* : retourne TRUE si la comparaison avec **au moins un** des tuples de *R* renvoie TRUE

valeur comp **ALL** *R* : retourne TRUE si la comparaison avec **tous** les tuples de *R* renvoie TRUE

## Sous-requêtes indépendantes

Quelles sont les références produit dont le prix HT est supérieur au prix HT moyen des produits ?

**SELECT** référence

**FROM** *Produit*

**WHERE** *Produit.prix\_ht* > (**SELECT** AVG(*P.prix\_ht*)  
**FROM** *Produit P*);



Le bloc SFW imbriqué peut être évalué séparément du bloc principal

## Sous-requêtes indépendantes (1)

Quels sont les marques de produits vendus le 1/1/2007 ?

```
SELECT marque
FROM Produit, Vente
WHERE Vente.ref-produit = Produit.reférence
      AND date = 1/1/2007;
```

ou

```
SELECT Produit.marque
FROM Produit
WHERE Produit.reférence IN
      (SELECT Vente.ref-produit
       FROM Vente
       WHERE date = 1/1/2007);
```

*Jointure ou requête imbriquée*

## Sous-requêtes indépendantes (2)

Quelles sont les marques de produits qui n'ont pas le prix HT le plus élevé ?

```
SELECT marque
FROM Produit
WHERE Produit.prix-ht < ANY
      (SELECT P.prix-ht
       FROM Produit P;)
```

*< SELECT max (P.prix\_ht) ...*

Quelles sont les marques de produits qui ont le prix HT le plus élevé ?

```
SELECT marque
FROM Produit
WHERE Produit.prix_ht >= ALL
      (SELECT P.prix_ht FROM Produit P;)
```

*> SELECT max (P.prix\_ht) ...*

## Sous-requêtes corrélées (1)

Quels produits dont le prix HT est supérieur au prix HT moyen des produits de la même marque ?

```
SELECT Produit.référence
FROM Produit
WHERE Produit.prix-ht > (SELECT AVG(P.prix-ht)
                        FROM Produit P
                        WHERE Produit.marque = P.marque);)
```

Le bloc *SFW* principal et le bloc *SFW* imbriqué doivent être évalués simultanément

## Sous-requêtes corrélées (2)

Quels sont les marques dont aucun produit n'a été vendu le 1/01/2007 ?

```
SELECT Produit.marque
FROM Produit
WHERE NOT EXISTS
    (SELECT Vente.ref-produit
     FROM Vente, Produit P
     WHERE Vente.ref-produit = P.référence and
           P.marque = Produit.marque and
           date = 1/01/2007;)
```

## Opérateur Division (1)

$R1(a,b) \div R2(b)$

Afficher les valeurs de R1.a pour lesquelles :  
il n'existe pas de valeur de R2.b telle que :  
(R1.a, R2.b) n'appartienne pas à R1

Département

numéro-département	...	responsable
--------------------	-----	-------------

Employé

numéro-employé	département	...	fonction
----------------	-------------	-----	----------

Donner le numéro du département, s'il existe, qui emploie tous les ingénieurs système. (cas particulier: un employé ∈ un seul dept.)

```
SELECT D.numéro-département
FROM Département D
WHERE not exists (SELECT * FROM Employe E
                  WHERE E.fonction = 'Ingénieur Système'
                  AND E.département != D.numéro-département);
```

## Opérateur Division

$R1(a,b) \div R2(b)$  : Afficher les valeurs de R1.a pour lesquelles : il n'existe pas de valeur de R2.b telle que (R1.a, R2.b) n'appartienne pas à R1

Département-Employé

num-département	...	num-employé
-----------------	-----	-------------

Employé

num-employé	...	...	fonction
-------------	-----	-----	----------

Donner le numéro du département, s'il existe, qui emploie tous les ingénieurs système.

```
SELECT distinct DE1.num-département
FROM Département-Employé DE1
WHERE not exists
(SELECT * FROM Employe E
 WHERE E.fonction = 'Ingénieur Système' AND NOT EXISTS
 (SELECT * FROM Département-Employé DE2
  WHERE DE1.num-département = DE2.num-département
  AND DE2.num-employé = E.num-employé));
```

## Opérateur *Division*

```
SELECT distinct département
FROM EMPLOYE
WHERE fonction = 'Ingénieur Système';
GROUP BY département
HAVING count(numéro-employé) =
  (SELECT count(numéro-employé) FROM EMPLOYE E)
  WHERE E.fonction = 'Ingénieur Système');

SELECT *
FROM EtudiantUE
GROUP BY etudiant
HAVING count(uniteValeur) =
  (SELECT count(code) FROM UE);
```

## Exercices

Donner le nom des employés qui ont le même salaire que Dupont

*Employé*

numéro-employé	nom	...	salaire
----------------	-----	-----	---------

```
SELECT Y.nom
FROM Employé X , Employé Y
WHERE
  X.nom='Dupont'
  and Y.salaire = X.salaire
  and X.nom != Y.nom;
```

*Remplacer la jointure par une requête imbriquée*

## Manipulation des structures de données

- Schéma logique / Base de données  
→ CREATE/DROP SCHEMA/DATABASE ...
- Schéma de **tables** (*relations*) et de leur contenu  
→ CREATE/ALTER/DROP TABLE ...
- Définition des **contraintes** qui assurent des contrôles sur l'intégrité des données
- Mais encore :
  - les index
  - les utilisateurs et les privilèges

## Création/Suppression d'un schéma

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] *db\_name*  
DROP DATABASE [IF EXISTS] *db\_name*

- Dans la norme SQL: création de schéma
- MySQL: *schema* = *database*
- Il faut avoir les droits de le faire (administrateur?)
- Création d'un compte s'il n'existe pas
- Association des droits

*Pas de message  
d'erreur si la BD  
existe!*

**CREATE DATABASE** *lahire06-07*

**GRANT ALL PRIVILEGES**

**ON** *lahire06-07.\**

**TO** ' *lahire06-07*'@localhost

**IDENTIFIED BY** ' *lahire06-07*';

**DROP DATABASE** *lahire06-07*

*Attention !*

## Types de la norme ANSI

INTEGER	Entiers relatifs	4 octets
SMALLINT	Entiers relatifs	2 octets
BIGINT	Entiers relatifs	8 octets
FLOAT	Flottants	4 octets
DOUBLE	Flottants	8 octets
REAL	Flottants	4 ou 8 octets
NUMERIC(N,D)/ DECIMAL(N,D)	Décimaux à précision fixe	N octets
CHAR(M)	Chaînes de longueur fixe	M octets
VARCHAR(M)	Chaînes de longueur variable	au plus M octets
BIT VARYING	Chaînes d'octets	longueur de la chaîne
DATE	date(hour, mois, an)	4 octets
TIME	heure(h, mn, sh)	4 octets
DATETIME	date et heure	8 octets
YEAR	année	2 octets

## Création et gestion des relations

- Structures de données (relations, attributs, tuples)

relation → **table**  
 attribut → **column**  
 tuple → **row**

- Instructions sur les relations

**CREATE TABLE**

*créer une relation*

**DROP TABLE**

*supprimer une relation*

**ALTER TABLE**

*modifier la structure d'une relation*



## Création de table/relation (1)

```
CREATE TABLE Produit (  
  référence CHAR(4),  
  nom VARCHAR(20),  
  marque VARCHAR(15),  
  constructeur CHAR(20)  
);
```

```
CREATE TABLE Client (  
  numéro INT(10),  
  nom VARCHAR(20),  
  adresse VARCHAR(30),  
  téléphone CHAR(10)  
);
```

```
CREATE TABLE Vente (  
  numéro INT(10),  
  ref-produit INT(10),  
  ref-client INT(10),  
  date DATE  
);
```

- Compléter la description
- Voir avec les types MySQL

## Création de table (valeurs nulles, défaut)

- NOT NULL : l'attribut correspondant doit toujours avoir une valeur  
nom VARCHAR(20) NOT NULL
- DEFAULT : définir la valeur d' un attribut par défaut  
adresse VARCHAR (30) DEFAULT 'Inconnue'

```
CREATE TABLE Produit (  
  référence CHAR(4) NOT NULL,  
  nom VARCHAR(20) NOT NULL,  
  marque VARCHAR(15),  
  constructeur CHAR(20) DEFAULT 'Renault'  
);
```

## Création de table (clé primaire)

### Identification des tuples

**PRIMARY KEY** (*att*)

**PRIMARY KEY** (*att1, ..., attn*)

→ mono attribut ou multi attributs

→ Attributs figurant dans une clé : déclaration NOT NULL

```
CREATE TABLE Produit (  
  référence CHAR(4) NOT NULL,  
  nom VARCHAR(20) NOT NULL,  
  marque VARCHAR(15),  
  constructeur CHAR(20) DEFAULT 'Renault',  
  PRIMARY KEY (référence)  
);
```

## Création de table (unicité)

### Unicité des valeurs dans une colonne

**UNIQUE** (nom, prenom)

→ Principalement pour les clés secondaires

```
CREATE TABLE Produit (  
  référence CHAR(4) NOT NULL,  
  nom VARCHAR(20) NOT NULL,  
  marque VARCHAR(15),  
  constructeur CHAR(20),  
  PRIMARY KEY (référence),  
  UNIQUE (nom)  
);
```

NOT NULL non spécifié

UNIQUE ne s'applique pas aux valeurs nulles

## Création de table (clé étrangère)

**FOREIGN KEY (att) REFERENCES ...**  
**FOREIGN KEY (att1, ..., attn) REFERENCES ...**

**FOREIGN KEY (ref-produit) REFERENCES Produit**  
→ ref-produit référence la clé primaire de la table **Produit**

```
CREATE TABLE Vente (  
  numéro INT(10) NOT NULL,  
  ref-produit INT(10) NOT NULL,  
  ref-client INT(10) NOT NULL,  
  date DATE,  
  PRIMARY KEY (numéro),  
  FOREIGN KEY (ref-produit) REFERENCES Produit,  
  FOREIGN KEY (ref-client) REFERENCES Client  
);
```

## Gestion de l'intégrité référentielle

Vérification, si insertion, suppression, mise à jour ...

Les valeurs prises par la clé étrangère correspondent-elles à la clé?

Action par défaut : **rejet** de l'opération

Autres actions si celle par défaut ne convient pas

- **ON UPDATE** (en cas de mise à jour)
- **ON DELETE** (en cas de suppression)

Objectif: faire respecter la contrainte

- **SET NULL** clé étrangère mise à NULL
- **CASCADE** application de la même opération
- **SET DEFAULT** valeur par défaut pour la clé étrangère

## Intégrité référentielle (exemple)

```
CREATE TABLE Vente (  
  numéro INT(10) NOT NULL,  
  ref-produit INT(10) NOT NULL,  
  ref-client INT(10) NOT NULL,  
  date DATE,  
  PRIMARY KEY (numéro),  
  FOREIGN KEY (ref-produit) REFERENCES Produit  
  ON DELETE SET NULL ON UPDATE CASCADE,  
  FOREIGN KEY (ref-client) REFERENCES Client  
  ON DELETE SET NULL ON UPDATE CASCADE  
);
```

- Si suppression dans Produit : ref-produit = NULL
- Si mise à jour dans Produit : ref-produit = mise à jour

## Mise à jour d'un schéma de BD

- Suppression d'une relation  
`DROP TABLE nom de relation ;`

- Modification d'une table

`ALTER TABLE nom de_relation` **ACTION** *description*

ADD, MODIFY, ALTER, DROP, RENAME

*description commande*

Mais aussi :

`ALTER TABLE Vente TYPE = innodb`

## Modification d'une table (structure)

```
ALTER TABLE Produit MODIFY nom VARCHAR(25);  
ALTER TABLE Produit ADD quantité-stock VARCHAR(20);  
ALTER TABLE Client ALTER adresse SET DEFAULT 'NICE';  
ALTER TABLE Client DROP adresse;  
ALTER TABLE Client  
  ADD CONSTRAINT CT_UN UNIQUE (nom) ;  
ALTER TABLE Client  
  ADD CONSTRAINT CT_PR PRIMARY KEY (numéro) ;  
ALTER TABLE Vente ADD CONSTRAINT CT_ET  
  FOREIGN KEY (ref-client) REFERENCES Client ;
```

## Mises à jour des données

- **INSERT INTO**  
pour insérer des tuple
- **DELETE FROM**  
pour supprimer des tuples
- **UPDATE**  
pour mettre à jour des tuples

## Insertion de tuples (1)

Insertion avec désignation explicite des colonnes

```
INSERT INTO Produit (nom, marque, constructeur)
VALUES ('mercedes 300 SL', 'mercedes benz',
       'daimler');
```

Insertion dans l'ordre des colonnes

```
INSERT INTO Produit
VALUES (DEFAULT, 'mercedes 300 SL', 'mercedes
       benz', 'daimler');
```

## Insertion de tuples (2)

Insertion à partir d'une autre table

```
INSERT INTO Produit (nom, marque, constructeur)
SELECT P.nom, P.marque, p.constructeur
FROM OldProduit P
WHERE P.marque = 'Peugeot'
```

➡ Evolution du schéma relationnel

## Suppression de tuples

Suppression de tous les tuples

```
DELETE FROM Produit
```

ou

```
DELETE FROM Produit WHERE 1 > 0    (retourne le nombre)
```

Suppression conditionnelle

```
DELETE FROM Produit
```

```
WHERE marque = 'peugeot'
```

```
DELETE FROM VENTE WHERE ref-produit IN  
  (SELECT référence  
   FROM Produit  
   WHERE Produit.marque = 'peugeot');
```

## Modification des tuples

```
UPDATE Produit
```

```
SET nom = 'mercedes 300 SLK'
```

```
WHERE numero =3;
```

*Modification d'un tuple*

```
UPDATE Produit
```

```
SET prix-ht = prix-ht * 0,9
```

```
WHERE référence NOT IN
```

```
  (SELECT ref-produit
```

```
   FROM VENTE
```

```
   WHERE quantité > 10);
```

*Modification d'une  
collection de tuple*

## Compléments: Définition de vues

Une vue est une table virtuelle dérivée de tables de base :

- On stocke seulement la définition de vue
- Son contenu est généré dynamiquement

```
CREATE VIEW ProduitPhare
AS
SELECT *
FROM Vente V, Produit P
WHERE
    V.ref-produit = P.référence
    AND V.quantité > 100 ;
```

```
CREATE VIEW VenteInfo
    (référence, date, marque)
AS
SELECT P.référence, V.date,
        P.marque
FROM Vente V, Produit P
WHERE P.référence = V.ref-produit;
```

➡ **Evolution du schéma relationnel**

**Requêtes fréquentes**

**DROP VIEW VenteInfo**

## Compléments: Création de contraintes

**CREATE ASSERTION**

**CHECK** *pour décrire des contraintes générales*

**CREATE ASSERTION** *QuantiteVendue*

**CHECK NOT EXISTS** (

**SELECT** V.quantité

**FROM** Vente V, Produit P

**WHERE**

V.ref-produit = P.référence AND

P.quantité-vendue < V.quantité);

*On suppose que la table Produit  
contient la colonne quantité-vendue*



## Compléments: Création d'index

### CREATE INDEX

Un index offre un chemin d'accès aux lignes d'une table

**CREATE INDEX index1 ON Client(nom,prenom);**

un index est systématiquement défini sur la clé primaire de chaque table

pour chaque clause UNIQUE utilisée dans la création de la table, un index permet de vérifier rapidement, au moment d'une insertion, que la clé n'existe pas déjà

## Compléments: Transactions

### Enlever la validation implicite:

SET AUTOCOMMIT=0 ;

### Déclencher une transaction et terminer par valider/annuler

START TRANSACTION;

SELECT @A:=SUM(salary) FROM table1 WHERE type=1;

UPDATE table2 SET summary=@A WHERE type=1;

COMMIT; ou ROLLBACK;



Un mécanisme complexe et sophistiqué

## Compléments: Gestion des privilèges

SHOW DATABASES ;

SHOW PRIVILEGES ;

GRANT *privilèges* ON ... TO ...

REVOKE *privilèges* ON ... FROM ...

ALL [PRIVILEGES]	Tous les droits sauf WITH GRANT OPTION
ALTER	Autorise l'utilisation de ALTER TABLE
CREATE	Autorise l'utilisation de CREATE TABLE
DELETE	Autorise l'utilisation de DELETE
DROP	Autorise l'utilisation de DROP TABLE
...	...