LES BASES DE DONNEES

BTS SNIR

LYCEE NEWTON-ENREA

Prenons l'exemple d'un bon de commande

Commande No : *14010* **Date :** *24/09/2014*

Numéro client : *BD2014*
Nom : *LECOIN*
Adresse : *24 RUE SAINT-JEAN*
Localité : *CAEN*

REF PRODUIT	LIBELLE	PRIX	QUANTITE	SOUS-TOTAL
190464K	Calcium chlorure 1 mol/l	77	10	770
31535.292	Sodium chlorure 1 mol/l (1N)	105	15	1575
30024.290	Acide chlorhydrique 1 mol/l (1 N)	41	3	123
30917.320	Iode 0,05 mol/l (0,1 N)	117	8	936
			TOTAL COMMANDE	3404

Comment stocker le bon de commande ?

Image numérisée ★☆☆☆☆

Tableau dans un traitement de texte ★☆☆☆☆

Feuille de calcul dans un tableur ★★☆☆☆

Base de données ★★★★★

Quelles données extraire du bon de commande ?

Commande No : 14010 Date : 24/09/2014

Numéro client : BD2014
Nom : LECOIN
Adresse : 24 RUE SAINT-JEAN
Localité : CAEN

données du client

REF PRODUIT	LIBELLE	PRIX	QUANTITE	SOUS-TOTAL
190464K	Calcium chlorure 1 mol/l	77	10	770
31535.292	Sodium chlorure 1 mol/l (1N)	105	15	1575
30024.290	Acide chlorhydrique 1 mol/l (1 N)	41	3	123
30917.320	Iode 0,05 mol/l (0,1 N)	117	8	936
			TOTAL COMMANDE	3404

données d'un détail

données de la commande

Comment structurer les données ?

Les données peuvent être présentées dans des **tables**

Nom_la_table

Ville	Département	Région	Population
Strasbourg	Bas-Rhin	Alsace	271782
Bordeaux	Gironde	Aquitaine	239157
Dijon	Côtes-d'Or	Bourgogne	151212
Brest	Finistère	Bretagne	141303

Attribut

Enregistrement

Associons, par exemple, une table à chaque entité

COMMANDES

NCOM	DATECOM	TOTAL-COM
14010	24/09/2014	3404

CLIENTS

NCLI	NOM	ADRESSE	LOCALITE
BD2014	LECOIN	24 RUE SAINT-JEAN	CAEN

PRODUITS

REFPROD	LIBELLE	PRIX	QCOM	SOUS-TOTAL
190464K	Calcium chlorure 1 mol/l	77	10	770
31535.292	Sodium chlorure 1 mol/l (1 N)	105	15	1575
30024.290	Acide chlorhydrique 1 mol/l (1 N)	41	3	123
30917.320	Iode 0,05 mol/l (0,1 N)	117	8	936

Il manque des liens entre tables

La table `commandes` n'a aucune information sur le client

La table `produits` n'a aucune information sur la commande

... Et avec références

COMMANDES

NCOM	NCLI	DATECOM
14010	BD2014	24/09/2014

CLIENTS

NCLI	NOM	ADRESSE	LOCALITE
BD2014	LECOIN	24 RUE SAINT-JEAN	CAEN

PRODUITS

NCOM	REFPROD	QCOM	LIBELLE	PRIX
14010	190464K	10	Calcium chlorure 1 mol/l	77
14010	31535.292	15	Sodium chlorure 1 mol/l (1 N)	105
14010	30024.290	3	Acide chlorhydrique 1 mol/l (1 N)	41
14010	30917.320	8	Iode 0,05 mol/l (0,1 N)	117

Peut-on encore améliorer ?

La table `produits` contient les numéros de commande et les quantités

Est-ce vraiment nécessaire ?

Distribution optimale des données de bons de commandes

COMMANDES

NCOM	NCLI	DATECOM
14010	BD2014	24/09/2014

CLIENTS

NCLI	NOM	ADRESSE	LOCALITE
BD2014	LECOIN	24 RUE SAINT-JEAN	CAEN

DETAILS

NCOM	REFPROD	QCOM
14010	190464K	10
14010	31535.292	15
14010	30024.290	3
14010	30917.320	8

PRODUITS

REFPROD	LIBELLE	PRIX
190464K	Calcium chlorure 1 mol/l	77
31535.292	Sodium chlorure 1 mol/l (1 N)	105
30024.290	Acide chlorhydrique 1 mol/l (1 N)	41
30917.320	Iode 0,05 mol/l (0,1 N)	117

Que peut-on faire de ces données ?

Les conserver

Les interroger

Quel est le numéro, le nom et l'adresse des clients de Caen ?

```
select NCLI, NOM, ADRESSE
from   CLIENTS
where  LOCALITE = 'CAEN';
```

Quelles sont les commandes des clients de Caen ?

```
select NCOM
from   COMMANDES
where  NCLI in (select NCLI
                from   CLIENTS
                where  LOCALITE = 'CAEN');
```

Requêtes réalisées dans le langage SQL

Que peut-on faire de ces données ?

Et encore ...

Vérifier une commande
Produire une facture
Gérer les stocks
Calculer le chiffre d'affaire (par produit, par localité, etc.)
Étudier la répartition géographique des ventes
etc.

Types de bases de données

1960s Navigationnelle
- exploite les possibilités des disques durs récemment inventés

1970s Relationnelle
- ensemble de tables structurées
- langage SQL standardisé en 1986

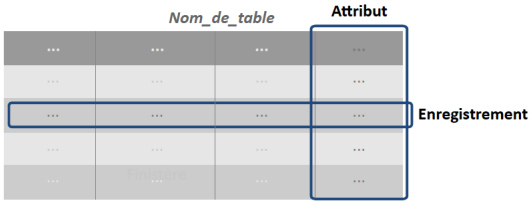
1980s Orientée objets
- collection d'objets persistants

2000s NoSQL (*Not Only SQL*)
- représentation arborescente
- modèle navigationnel plus structuré
- adapté aux très grandes bases (Internet)

Généralités sur les bases de données (relationnelles)

Le modèle relationnel

L'information est représentées par un ensemble de **tables**



Une table établit une **relation** entre ses attributs

Le modèle relationnel

Chaque attribut a un **nom** et un **type**

Villes

...	...	...	Population
...	...	...	...
Bordeaux	...	...	239157
...	...	...	...
...	...	...	...

nombre entier

Le modèle relationnel

Chaque attribut a un **nom** et un **type**

Hydrocarbures

...	...	PM	...
...	...	...	...
...	...	...	...
...	...	...	...
Butane	...	58,123	...

nombre réel

Le modèle relationnel

Chaque attribut a un **nom** et un **type**

Chimistes

...	Nom	...	...
...	...	...	...
...	Volta	...	...
...	...	...	...
...	...	...	...

chaîne de caractères

Le modèle relationnel

Chaque attribut a un **nom** et un **type**

Chimistes

...	...	Naissance	...
...	...	...	...
...	Volta	18-02-1745	...
...	...	...	...
...	...	...	...

date

Le modèle relationnel : un exemple

Villes

Ville	Departement	Region	Population
Strasbourg	Bas-Rhin	Alsace	271782
Bordeaux	Gironde	Aquitaine	239157
Dijon	Côte-d'Or	Bourgogne	151212
Brest	Finistère	Bretagne	141303
Amiens	Somme	Picardie	133448
...	...	...	...

4 attributs

- 3 textuels : `Ville`, `Departement`, `Region`
- 1 numérique : `Population`

261 enregistrements

Le modèle relationnel : la notion de requête

Une base de données permet de répondre à des **requêtes**

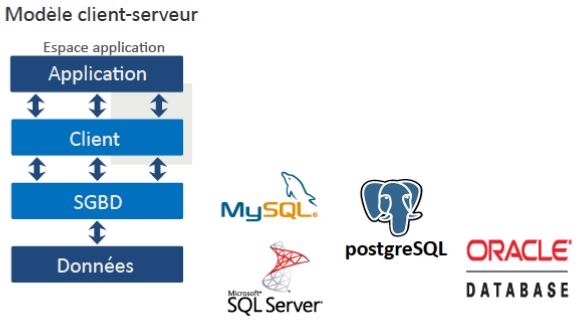
Formulées en **SQL** (*Structured Query Language*)

- Prononcer SeQueL
- En français : langage de requête structurée
- Créé en 1974 et normalisé en 1986

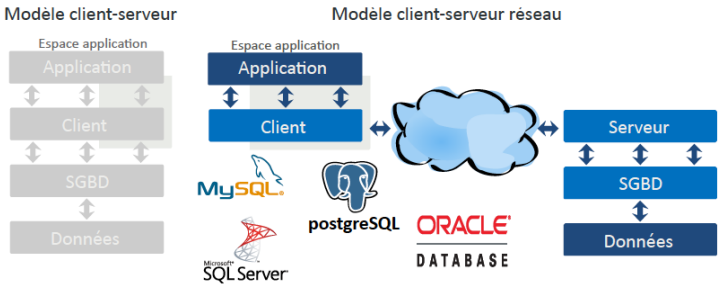
Interprétées et exécutées par le SGBD

La plupart des SGBDs fonctionnent en mode client/serveur

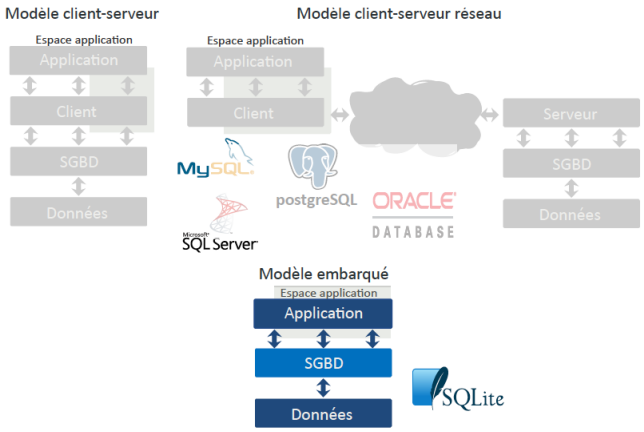
Quelques SGBDs relationnels



Quelques SGBDs relationnels



Quelques SGBDs relationnels



Langage SQL

Le langage SQL

Permet de gérer une base de données et d'effectuer des requêtes

Définir les éléments d'une base de données

Gérer les droits d'accès aux données

Manipuler les données

Définir les éléments d'une base de données

`CREATE TABLE` : Créer une table

Syntaxe

```
CREATE TABLE nom_table(
    colonne1 datatype  PRIMARY KEY ,
    colonne2 datatype,
    colonne3 datatype,
    ...
    colonneN datatype,
);
```

Définir les éléments d'une base de données

`CREATE TABLE` : Créer une table

Exemple

```
CREATE TABLE Etudiants(
   "Numero"          INTEGER PRIMARY KEY NOT NULL,
   "Nom"             TEXT                NOT NULL,
   "Prenom"          TEXT                NOT NULL,
   "Date_naissance" INTEGER              NOT NULL
);
```

Définir les éléments d'une base de données

`CREATE TABLE` : Créer une table
`DROP TABLE` : Supprimer une table

Exemple

```
DROP TABLE Produits;
```

Définir les éléments d'une base de données

`CREATE TABLE` : Créer une table
`DROP TABLE` : Supprimer une table
`ALTER TABLE` : Modifier une table

Exemple

```
ALTER TABLE Villes ADD COLUMN "CodePostal";

ALTER TABLE Villes DROP COLUMN "CodePostal";

ALTER TABLE Produits RENAME TO Ordinateurs;
```

Gérer les droits d'accès aux données

`GRANT` : Attribuer des droits sur une table

`REVOKE` : Supprimer des droits d'accès sur une table

Manipuler les données

`INSERT` : Insérer des données dans une table

Syntaxe

```
INSERT INTO Nom_Table [(champ1,champ2,...,champN)]
       VALUES (valeur1,valeur2,...,valeurN);
```

Manipuler les données

`INSERT` : Insérer des données dans une table

Exemple

```
INSERT INTO Villes (Ville,Departement,Region,
                    Population)
       VALUES ("Castelmoron-d'Albret","Gironde",
               "Aquitaine",54);
```

Manipuler les données

INSERT : Insérer des données dans une table
UPDATE : Modifier des données dans une table

Syntaxe

```
UPDATE Nom_Table
       SET champ1 = valeur1 , champ2 = valeur2 ,
           .... , champN = valeurN
       WHERE [condition];
```

Manipuler les données

INSERT : Insérer des données dans une table
UPDATE : Modifier des données dans une table

Exemple

```
UPDATE Villes SET Population = 56
              WHERE Ville = "Castelmoron-d'Albret";
```

Manipuler les données

`INSERT` : Insérer des données dans une table
`UPDATE` : Modifier des données dans une table
`DELETE` : Supprimer des données dans une table

Syntaxe

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

Exemple

```
DELETE FROM Villes WHERE ( Population < 100
        AND Ville IS NOT "Castelmoron-d'Albret");

DELETE FROM Villes WHERE ( Population < 100
        AND Ville != "Castelmoron-d'Albret");
```

Manipuler les données

`INSERT` : Insérer des données dans une table
`UPDATE` : Modifier des données dans une table
`DELETE` : Supprimer des données dans une table
`SELECT` : Extraire des données depuis une table

Syntaxe

```
SELECT champ1,champ2,...,champN FROM Nom_Table
       WHERE conditions
       GROUP BY expressions
       HAVING expression
       ORDER BY ordonnancement [DESC]
       LIMIT expression [OFFSET expression]
```

Manipuler les données avec SELECT

Projeter les données sur un sous-ensemble de colonnes

Exemple

```
SELECT Ville, Population FROM Villes
```

Résultat

Ville	Population
Strasbourg	271782
Bordeaux	239157
Dijon	151212
Brest	141303
Amiens	133448
...	...

Commande SELECT : clause WHERE

Projeter sur un sous-ensemble des enregistrements

Exemple

```
SELECT Ville, Population FROM Villes
    WHERE Departement = "Bas-Rhin";
```

Résultat

Ville	Population
Strasbourg	271782
Haguenau	34280
Schiltigheim	30952

Commande SELECT : clause WHERE

Projeter sur un sous-ensemble des enregistrements

Exemple

```
SELECT * FROM Villes
        WHERE Region = "Alsace";
```

Résultat

Ville	Departement	Region	Population
Strasbourg	Bas-Rhin	Alsace	271782
Mulhouse	Haut-Rhin	Alsace	109588
Colmar	Haut-Rhin	Alsace	67615
Haguenau	Bas-Rhin	Alsace	34280
Schiltigheim	Bas-Rhin	Alsace	30952

Commande SELECT : clause ORDER BY

Trier les enregistrements extraits

Exemple

```
SELECT Ville, Population FROM Villes
       ORDER BY Population DESC;
```

Résultat

Ville	Population
Strasbourg	271782
Bordeaux	239157
Dijon	151212
Brest	141303
Amiens	133448
...	...

Commande SELECT : clause LIMIT

Limiter le nombre d'enregistrements extraits

Exemple

```
SELECT Ville, Population FROM Villes
       ORDER BY population DESC
       LIMIT 4;
```

Résultat

Ville	Population
Strasbourg	271782
Bordeaux	239157
Dijon	151212
Brest	141303

Commande SELECT : clauses LIMIT et OFFSET

Ignorer les premiers enregistrements

Exemple

```
SELECT Ville, Population FROM Villes
       ORDER BY Population DESC
       LIMIT 4 OFFSET 3;
```

Résultat

Ville	Population
Brest	141303
Amiens	133448
Metz	120738
Orléans	114167

Comment identifier un enregistrement ?

Exemple

```
SELECT * FROM Villes WHERE Ville = "Saint-Denis";
```

Résultat

Ville	Departement	Region	Population
Saint-Denis	La Réunion	La Réunion	141303
Saint-Denis	Seine-Saint-Denis	Île-de-France	106785

Problème : Quelle est la population de la ville Saint-Denis ?

Clé primaire

Définition

Champ ou ensemble de champs qui permettent d'identifier un enregistrement de façon unique

Dans la pratique

On utilise un **champ spécifique entier** et de valeur **unique** gérée par le SGBD

Exemple

```
CREATE TABLE Villes(
    Id      INTEGER PRIMARY KEY AUTOINCREMENT,
    Ville   TEXT,
    ...
);
```

Clé primaire

La modification d'une table est facilitée

Exemple

```
SELECT * FROM Villes WHERE Ville = "Saint-Denis";
```

Résultat

Id	Ville	Departement	Region	Population
20	Saint-Denis	La Réunion	La Réunion	141303
37	Saint-Denis	Seine-Saint-Denis	Île-de-France	106785

Exemple

```
UPDATE Villes SET Population = 145000 WHERE Id = 20;
```

Clé primaire : Conséquences pratiques

L'insertion de valeurs est gérée par le SGBD

Exemple

```
INSERT INTO Villes (Ville, Departement, Region,
                    Population)
       VALUES ("Saint-Denis", "La Réunion",
               "La Réunion", 145022);
```

Clé primaire : Conséquences pratiques

Premier cas : Le SGBD n'a pas été notifié de la clé primaire

(oubli de `PRIMARY KEY`)

- L'insertion (`INSERT`) d'un nouvel enregistrement produit un doublon si sa clé primaire est déjà utilisée
- `INSERT OR REPLACE` n'a aucun effet
- Plusieurs requêtes nécessaires pour éviter les doublons
  - `SELECT` pour vérifier la présence de la clé primaire
  - `UPDATE` si la clé est déjà utilisée
  - `INSERT` dans le cas contraire

Conclusion

Le SGBD n'empêche pas la production de doublons, puisqu'il ne sait pas que l'un des attributs est une clé primaire. C'est au programmeur de s'assurer de l'absence de doublons

Clé primaire : Conséquences pratiques

Premier cas : Le SGBD n'a pas été notifié de la clé primaire

Exemple

```
INSERT INTO Villes (Ville, Departement, Region,
                    Population)
       VALUES ("Saint-Denis", "La Réunion",
               "La Réunion", 145022);
```

Résultat

Id	Ville	Departement	Region	Population
20	Saint-Denis	La Réunion	La Réunion	145000
37	Saint-Denis	Seine-Saint-Denis	Île-de-France	106785
262	Saint-Denis	La Réunion	La Réunion	145022

Clé primaire : Conséquences pratiques

Deuxième cas : Le SGBD a été notifié de la clé primaire

L'insertion (`INSERT`) d'un nouvel enregistrement échoue si sa clé primaire est déjà utilisée

Cet échec peut-être détecté

L'instruction `UPDATE` permet de modifier les valeurs de l'enregistrement existant pour lui donner les nouvelles valeurs

L'instruction `INSERT OR REPLACE` permet de réaliser cela en une seule instruction

Conclusion

Le SGBD empêche la production de doublons

Un problème de conception de la table `Villes`

Id	Ville	Departement	Region	Population
7	Strasbourg	Bas-Rhin	Alsace	271782
35	Mulhouse	Haut-Rhin	Alsace	109588
...	...	...	...	...

Problèmes

- Répétition du nom de la région pour chaque département
- 36700 communes et 101 départements $\Rightarrow$ 36599 répétitions
- Toute information supplémentaire sur les départements et régions doit être répétée inutilement
- Il y a dépendance fonctionnelle {`Departement`} $\rightarrow$ {`Region`}

Solution : créer une table `Departements`

Table Departements

Informations sur les départements

Numero	Nom	...	Region
2A	Corse-du-Sud	...	Corse
973	Guyane	...	Guyane
976	Mayotte	...	Mayotte

Clé primaires possibles

{Numero} : Concise

{Nom} : Moins concise

{Numero, Nom} : Beaucoup moins concise

Ce sera {Numero}

Nouvelle table Villes

Nouvelle table Villes

Id	Ville	n_Departement	Population
77	Quimper	29	63550
78	Valence	26	63405
127	Albi	81	48916

- Noms des régions déplacés dans la table Departements
- Le champ n_Departement dans Villes correspond au champ Numero de Departements

Notion de clé étrangère

Définition (Clé étrangère)

Champ, ou un ensemble de champs, qui correspondent à une clé primaire dans une autre table

Exemple

- `n_Departement` dans `Villes` correspond à `Numero` dans `Departements`
- `Numero` est une **clé primaire** dans `Departements`
- `n_Departement` est une **clé étrangère** dans la table `Villes`

Notion de clé étrangère

Création de la nouvelle table `Villes`

```
CREATE TABLE Villes (
  Id INTEGER PRIMARY KEY,
  Ville VARCHAR(50),
  n_Departement INTEGER,
  FOREIGN KEY(n_Departement) REFERENCES
                    Departements(Numero),
  ...
);
```

Notion de clé étrangère

Propriété désirable

Pour toute valeur d'une clé étrangère, il est souhaitable qu'un enregistrement dont la clé primaire est cette valeur existe dans la table correspondante

Exemple

Si dans la table `Villes`, une ville est dans le département `2A`, il est souhaitable que ce département existe dans `Departements`

Propriété vérifiée automatiquement par certains SGBD

Clés primaires et étrangères : récapitulatif

Clé primaire : identifie un enregistrement dans une table

Ex : dans la table `Departements`, le champ `Numero` identifie les départements

Clé étrangère : identifie un enregistrement dans une autre table

Ex : dans la table `Villes`, le champ `n_Departement` identifie les départements de la table `Departements`