Les Bases de données & Langage SQL

Sidati KHABID

Table des matières

Table des matières 1
I. Base de données 4
1. Notions de base 4
Définitions : 4
Système de Gestion de Base de Données (SGBD) 4
2. Base de données = Fichiers ? 4
3. Objectifs des SGBD 5
4. Fonctions d'un SGBD 6
5. Définition et description des données 6
Niveau logique (conceptuel) 7
Niveau physique 7
Niveau externe 7
6. Manipulation et restitution des données 7
7. Contrôle 7
8. Modèles de SGBD 8
Le modèle hiérarchique 8
Le modèle réseau 9
Le modèle relationnel (SGBDR) 10
Le modèle déductif 11
Le modèle objet (SGBDO) 11
9. Principaux SGBD du marché 12
10. L'architecture des SGBD 12
II. Le modèle relationnel 12
1. Généralités 12
Notions de modèle de données : 12
Modèle relationnel 13
Concepts du modèle 13
Notions de clé primaire 14
Schéma d'une table 14
Problème de Redondance des données 14
Éliminer les redondances 14
Clé Étrangère 16
Contraintes d'intégrités 16

Exemple 17
Schéma d'une base de données 17
III. Algèbre relationnelle 18
1. Principe de l'algèbre relationnelle : 18
2. Opérations relationnelles 18
Projection $\boldsymbol{\pi}$ 18
Sélection $\boldsymbol{\sigma}$ 19
Renommage $\boldsymbol{\rho}$ 20
Opérateurs ensemblistes 20
La jointure $\bowtie$ 25
$\boldsymbol{\Theta}$ −jointure 26
Equi-jointure 26
Jointure naturelle 26
Jointure externe 26
La formulation de contraintes 27
Groupement et agrégation 27
IV. Le langage S.Q.L. 30
1. Introduction 30
Historique 30
Définition 30
2. Le langage SQL 32
Langage de Description de Données 32
Langage de Manipulation des Données 33
3. La sélection 35
DISTINCT 35
Fonctions intégrées 36
La Jointure 37
Opérateur de partitionnement 38
Opérateurs du WHERE 39
Syntaxe générale de la commande SELECT 40
4. Requêtes pivots 41
5. Introduction aux contraintes d'intégrité 41
Contrainte d'intégrité de domaine 41
Contrainte d'intégrité de relation (ou d'entité) 42
Contrainte d'intégrité de référence 42

6. Créer une table : CREATE TABLE 42
Introduction 42
Création simple 42
Les types de données 43
Création avec Insertion de données 44
7. Contraintes d'intégrité 44
Syntaxe 44
Contraintes de colonne 44
Contraintes de table 45
Complément sur les contraintes 45
8. Supprimer une table : DROP TABLE 45
9. Modifier une table : ALTER TABLE 45
Ajout ou modification de colonnes 45
Renommer une colonne 46
Renommer une table 46
10. Exemples 46
V. Les Jointures et leurs types 48
1. Principe 48
2. Exemple 48
3. Types de Jointures 48
4. Définitions et exemples 48
Cross Jointure 48
Jointure Interne 49
Jointure Externe 49
5. Performance 50

I. Base de données

1. Notions de base

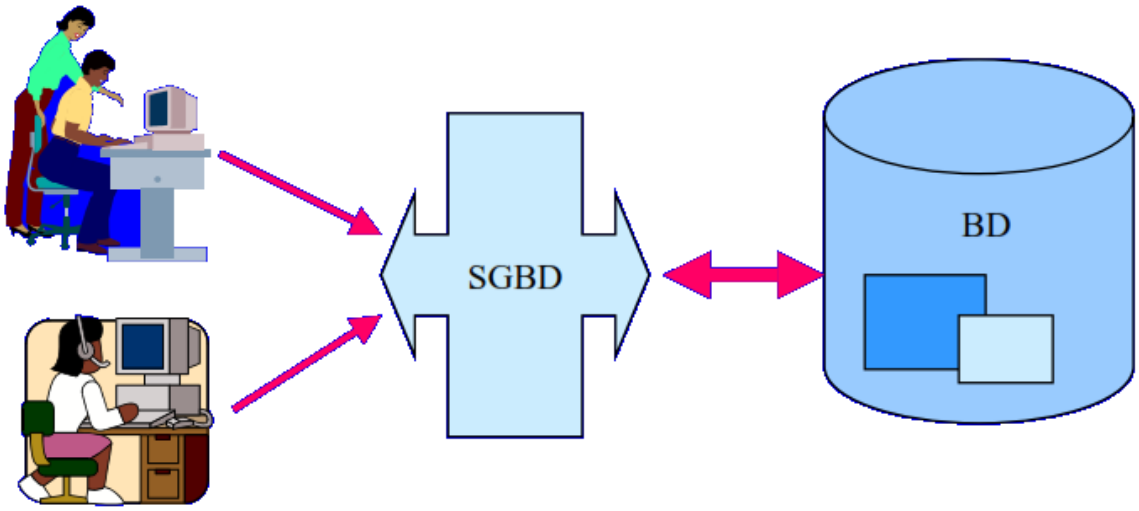
Définitions :

Une base de données est un ensemble structuré de données[1] enregistrées sur des supports accessibles par l'ordinateur[2] pour satisfaire simultanément plusieurs utilisateurs[3] de manière sélective[4] en un temps opportun[5] .

Système de Gestion de Base de Données (SGBD)

Le logiciel qui permet d'interagir avec une BD est un Système de Gestion de Base de Données (SGBD)

Il permet à des utilisateurs de créer et maintenir une base de données. Les activités supportées sont la défi-



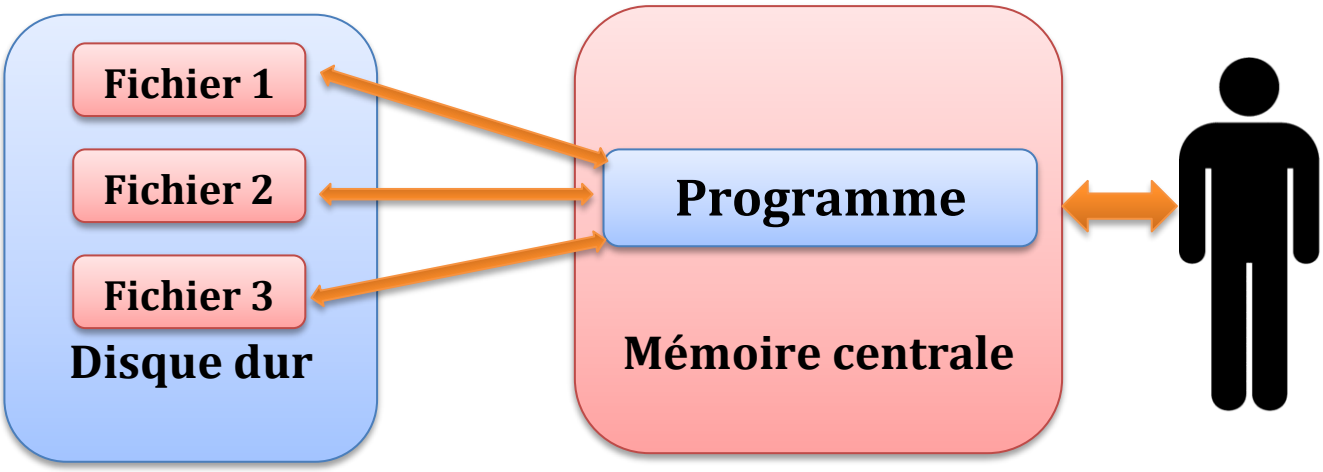
nition d'une base de données (spécification des types de données à stocker), la construction d'une base de données (stockage des données proprement dites) et la manipulation des données (principalement ajouter, supprimer, retrouver des données).

2. Base de données = Fichiers ?

Travailler directement sur un fichier présente plusieurs inconvénients :

- Manipulation de données lourde et compliquée. Il faut être expert en programmation
- Le programmeur doit connaître la localisation physique des fichiers, la structure physique des enregistrements, le mode d'accès à ces fichiers
- Toute modification de la structure des enregistrements (ajout d'un champ par exemple) entraîne la réécriture de tous les programmes qui manipulent ces fichiers

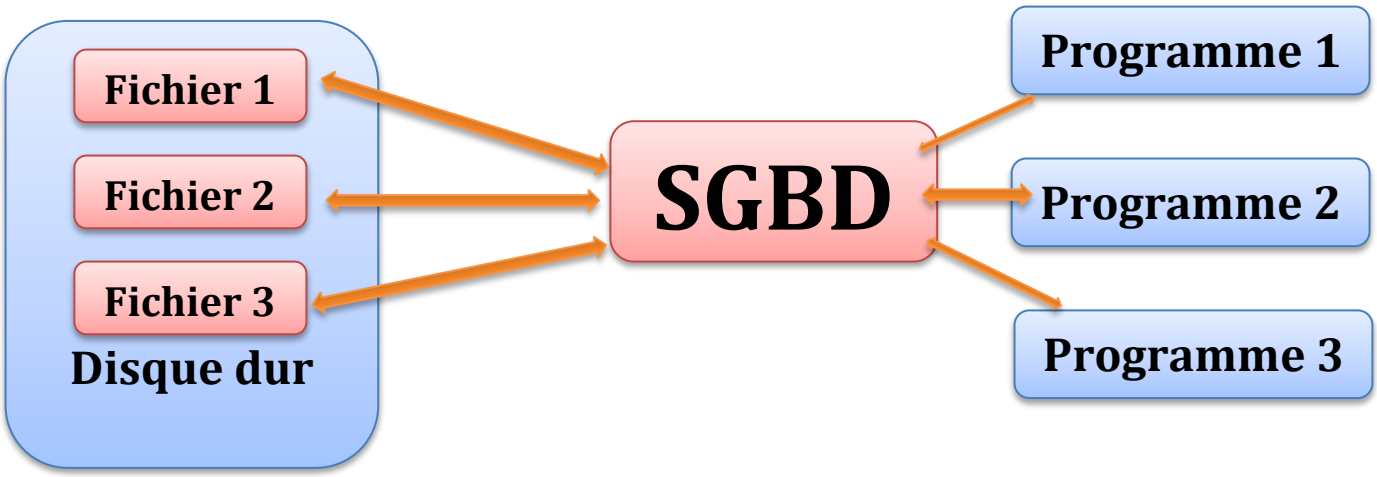
[1] Organisation et description de données
[2] Stockage sur disque
[3] Partage des données
[4] Confidentialité
[5] Performance



Un SGBD est un intermédiaire entre les utilisateurs et les fichiers physiques.

Un SGBD facilite :

- La gestion de données, avec une représentation intuitive simple sous forme de tables.
- La manipulation de données. On peut insérer, modifier les données et les structures sans modifier les programmes qui manipulent la base de données.



3. Objectifs des SGBD

- Faciliter la représentation et la description de données. Plus besoin de travailler directement sur les fichiers physiques (tels qu'ils sont enregistrés sur disque). Un SGBD nous permet de décrire les données et les liens entre elles d'une façon logique sans se soucier du comment cela va se faire physiquement dans les fichiers. On parle alors d'image logique de la base de données, (ou aussi description logique ou conceptuelle ou encore de schéma logique). Ce schéma est décrit dans un modèle de données par exemple le modèle de tables, appelé le modèle relationnel.

- Faciliter la manipulation en travaillant directement sur le schéma logique. On peut insérer, supprimer, modifier des données directement sur l'image logique. Le SGBD va s'occuper de faire le travail sur les fichiers physiques Permettre l'ajout des contraintes permettant d'avoir à tout instant des données cohérentes par exemple l'âge d'une personne supé rieur à zéro, salaire supérieur à zéro, etc. Dès que l'on essaie de saisir une valeur qui ne respecte pas cette contrainte, le SGBD le refuse
- Efficacité des Accès (Temps de réponse et débit global)

4. Fonctions d'un SGBD

- Description des données : codification structuration, grâce à un Langage de Description de Données(LDD)
- Rechercher des informations utiles en interrogeant la base de données, à l'aide d'un Langage d'Interrogation de Données (LID)
- Manipulation et restitution des données (insertion, mise à jour, interrogation)
  - ⇨ mise en œuvre à l'aide d'un Langage de Manipulation de Données (LMD)
  - ⇨ SQL (Structured Query Language):Langage standard
- Contrôle (partage, intégrité, confidentialité, sécurité) : permet de définir les droits d'accès pour les différents utilisateurs de la base de données, donc il permet de gérer la sécurité de la base et de confirmer et d'annuler les transactions grâce à un Langage de Contrôle de données (LCD)

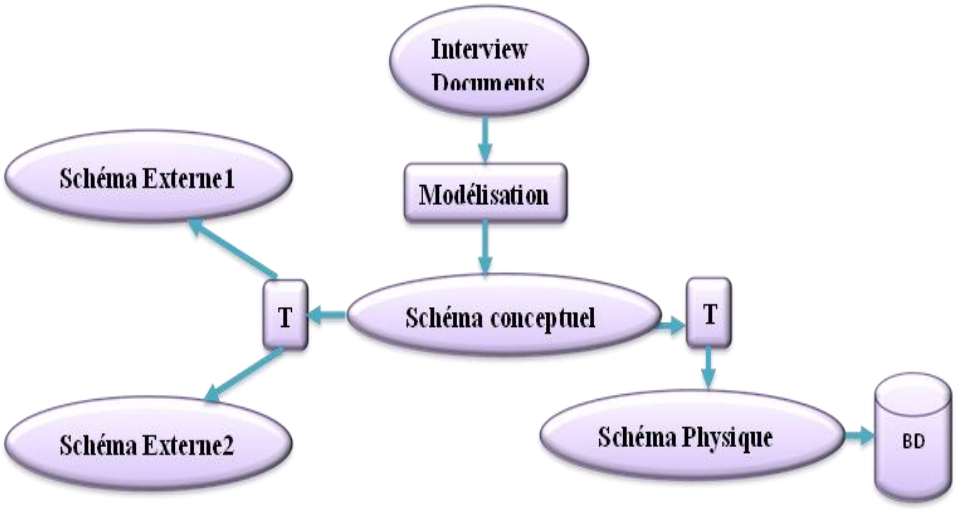
5. Définition et description des données

Les niveaux d'abstraction :

Il s'agit de simplifier la vision des utilisateurs des données physiques sur le disque.

Il y en a 3 définis par le groupe ANSI/X3/SPARC[6]

Architecture fonctionnelle de référence :



3 niveaux de description des données :

[6] the American National Standard Institute Standard Planning and Requirements Committee

Niveau logique (conceptuel)

- Permetla description :
  - ⇨ des objets : exemple OUVRAGES, ETUDIANTS
  - ⇨ des propriétés des objets (attributs) : exemple cote de OUVRAGES, Titre de OUVRAGES, nombre d'exemplaires etc.
  - ⇨ des liens entre les objets : un OUVRAGE peut être emprunté par un ETUDIANT
  - ⇨ des Contraintes : le nombre d'exemplaires d'un OUVRAG E est supérieur à zéro

Cette description est faite selon un modèle de données.

Un modèle de données est un ensemble de concepts permettant de décrire la structure d'une base de données. La plupart des modèles de données incluent des opérations permettant de mettre à jour et questionner la base. Le modèle de données le plus utilisé est **le modèle relationnel**,

Cette description va donner lieu à un schéma de base de données. Un schéma de base de données se compose d'une description des données et de leurs relations ainsi que d'un ensemble de contraintes d'intégrité.

Niveau physique

Description informatique des données et de leur organisation : en termes de fichiers, d'index, de méthodes d'accès, …

Passage du modèle logique au modèle physique tend à être assisté par le SGBD : transparent et /ou semi-automatique

Objectifs : optimiser les performances

Niveau externe

Description des données vues par un utilisateur (ou un groupe d'utilisateurs)

Objectifs : simplification, confidentialité

Exemple : OUVRAGES édités par des éditeurs français

6. Manipulation et restitution des données

Permet :

- Insertion : saisir des données
- Supprimer
- Modifier
- Interroger : rechercher des données via des requêtes

La manipulation des données est mise en œuvre à l'aide d'un Langage de Manipulation de Données (LMD). S.Q.L (Structured Query Language) est le Langage standard de manipulation de BD

7. Contrôle

Un SGBD permet :

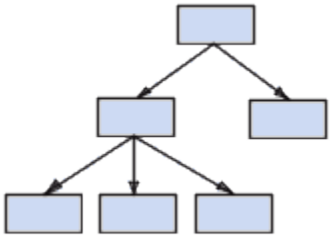
- Partage de données : accès à la même information par plusieurs utilisateurs en même temps. Le SGBD inclut un mécanisme de **contrôle de la concurrence** basé sur des techniques de verrouillage des données (pour éviter par exemple qu'on puisse lire une information qu'on est en train de mettre à jour)
- Intégrité des données grâce à la définition de contraintes sur les données. Le SGBD veille à ce que toutes les contraintes soient vérifiées à chaque insertion, suppression, ou modification d'une donnée.
- Confidentialité : plusieurs utilisateurs peuvent utiliser en même temps une base de données, se pose le problème de **la confidentialité des données**. Des droits doivent être gérés sur les données, droits de lecture, mise à jour, création, ... qui permettent d'affiner.
- Sécurité : une base de données est souvent vitale dans le fonctionnement d'une organisation, et il n'est pas tolérable qu'une panne puisse remettre en cause son fonctionnement de manière durable. Les SGBD fournissent des mécanismes pour assurer cette sécurité.

8. Modèles de SGBD

Il existe cinq modèles de SGBD, différenciés selon la représentation des données qu'elle contient.

Le modèle hiérarchique

Les données sont classées hiérarchiquement, selon une arborescence descendante. Ce modèle utilise des pointeurs entre les différents enregistrements. Il s'agit du premier modèle de SGBD



Il est employé par des systèmes très répandus comme IMS (IBM)

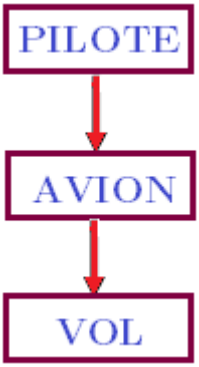
Dans ce modèle, la structure des données est décrite par deux concepts :

1. L'enregistrement logique ou article qui regroupe l'ensemble des propriétés ou champs constituant les caractéristiques d'une entité
2. Le lien orienté associant les articles selon une arborescence : un article père est relié à N articles fils

La BD est constituée d'une collection d'arbres (forêt)

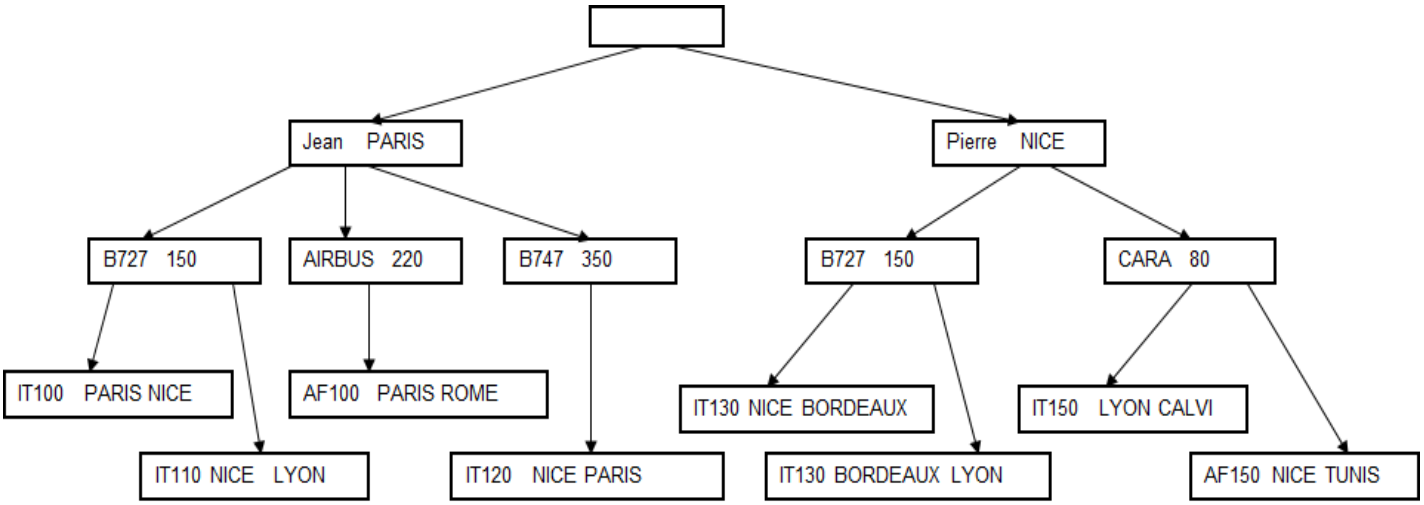
Exemple:

Arbre TRANSPORT_AERIEN
Article PILOTE Racine
    PNOM caractère (20)
    ADDRESSE caractère (30)
    ...
Article AVION Parent = PILOTE
    ANOM caractère (10)
    CAPACITE entier
Article VOL Parent = AVION
    VNOM caractère (5)



ORIGINE caractère (20)
DESTINATION caractère (20)

Occurrences du Schéma:



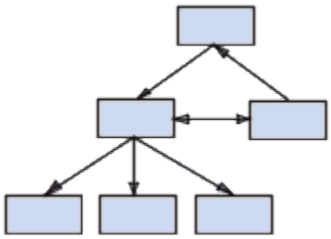
Inconvénients:

Le modèle est simple mais présente quelques faiblesses due à la structure arborescente :

1. Seule une association [1-N] sera naturellement représentée
2. Il y a duplication d'enregistrements communs à deux arborescences

Le modèle réseau

Comme le modèle hiérarchique, ce modèle utilise des pointeurs vers des enregistrements. Toutefois la structure n'est plus forcément arborescente dans le sens descendant.



Proposé par le groupe DBTG (Data Base Task Group): auparavant CODASYL Consortium

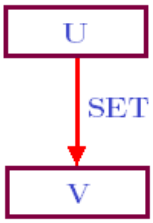
Son inventeur original était Charles Bachman.

Idée : permettre une modélisation plus naturelle des relations entre entités.

Différence entre un modèle hiérarchique et un modèle de réseau :

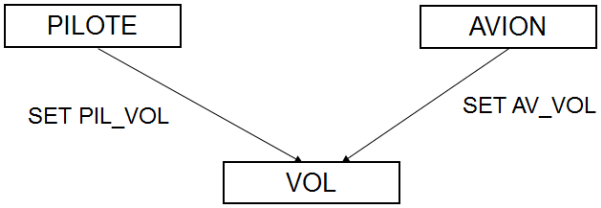
1. Le modèle hiérarchique structure les données comme un arbre d'articles (chaque enregistrement ayant un article parent et de nombreux enfants)
2. Le modèle de réseau permet à chaque enregistrement d'avoir de multiples enregistrements parents et enfants, formant une structure en treillis.

Les liens entre articles d'un modèle réseau appelés SET matérialise une association entre deux types d'articles distincts.

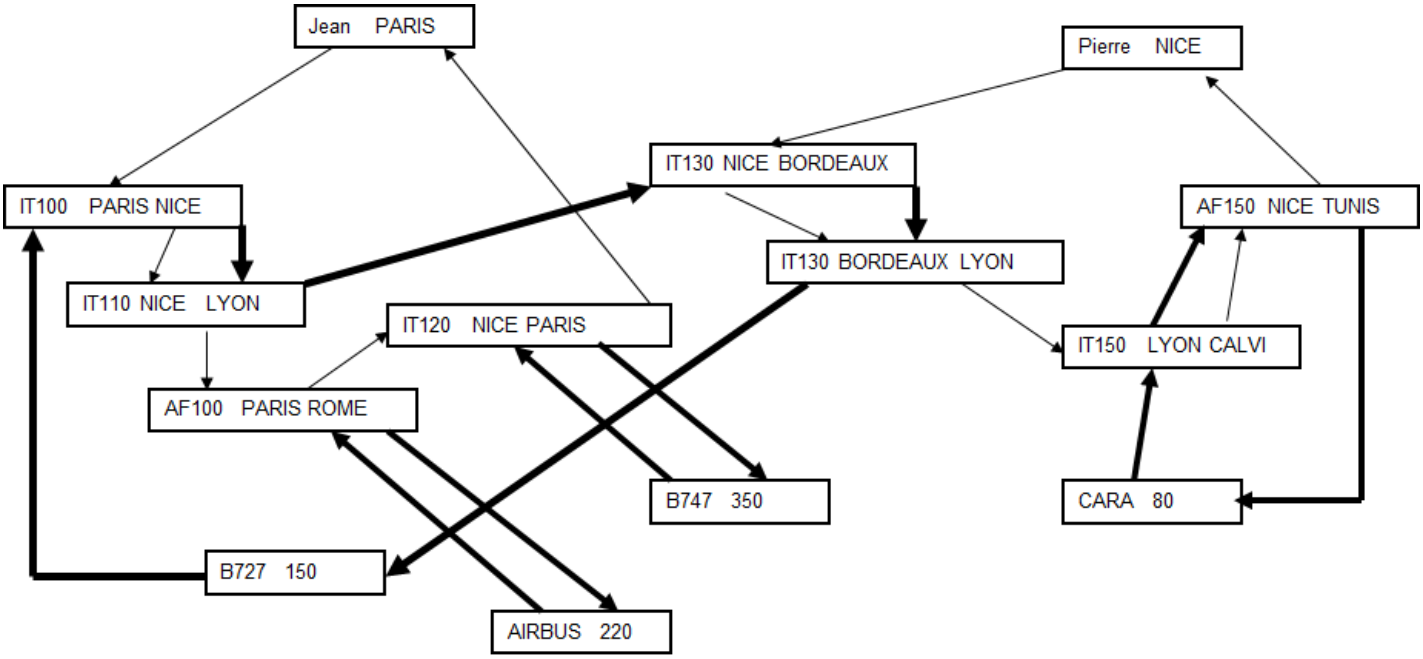


U est dit propriétaire, tout article V relié à U est dit membre
U et l'ensemble des V constitue une occurrence du SET
Un SET est une relation 1-N, mais le type V peut aussi être membre pour une autre relation SET : le graphe des occurrences permet donc des liens N-N entre articles.

Exemple:



Occurrences du Schéma:

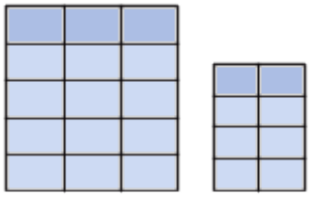


Le modèle était largement implémenté et utilisé. Mais il a échoué à devenir dominant pour deux raisons principales :

1. IBM a choisi de garder le modèle hiérarchique dans leurs produits établis.
2. Il fut surpassé par le modèle relationnel, qui offrait une interface de plus haut niveau et plus déclarative.

Le modèle relationnel (SGBDR)

Le plus utilisée, les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). La manipulation de ces données se fait selon la théorie mathématique des relations et l'algèbre relationnelle.



Le modèle déductif

Les données sont représentées sous forme de table, mais leur manipulation se fait par calcul de prédicats.

Un système de base de données déductive peut générer de nouveaux faits et règles, basés sur des règles et des faits stockés dans la base de données (déductive).

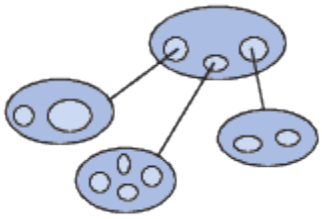
De tels systèmes :

- Gèrent principalement des règles et des faits.
- Utilisent un langage déclaratif (prolog ou datalog) pour spécifier ces règles et faits.
- Utilisent un moteur d'inférence qui peut déduire de nouveaux faits et règles à partir de ceux données.

Aussi connues comme : bases de données logiques, systèmes de connaissances, et bases de données référentielles.

Le modèle objet (SGBDO)

Les données sont stockées sous forme d'objets, c'est-à-dire de structures appelées *classes* présentant des données membres. Les champs sont des instances de ces classes, utilisation de ce modèle est plutôt rare.



A la fin des années 90 les bases de données relationnelles sont les bases de données les plus répandues (environ trois quarts des bases de données).

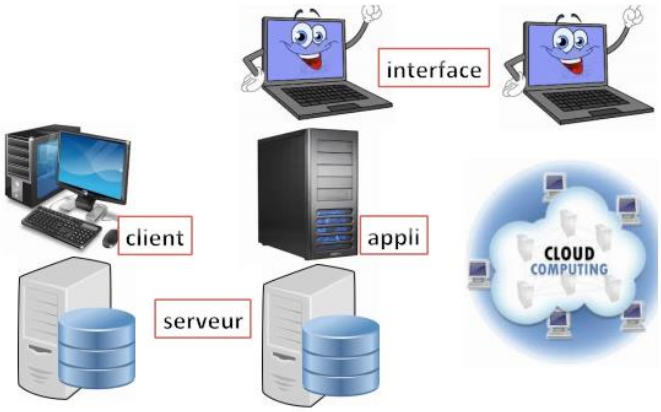
9. Principaux SGBD du marché

Les grands SGBD	Les SGBD personnels	Les open sources	Les SGBD objets
Oracle IBM DB2 Microsoft SQL Server Sybase SQL Server Ingres Informix	Borland Paradox Filemaker Interbase Microsoft Access Microsoft FoxPro	MySQL PostgreSQL	Objectivity Object Store Versant O2

10. L'architecture des SGBD

Elle est basée sur une architecture Client-Serveur :

- Données sur le serveur partagé entre plusieurs clients
- Interfaces graphiques sur la station de travail personnelle
- Communication par des protocoles standardisés
- Clients et serveurs communiquant par des requêtes avec réponses



(a) Client–Serveur (b) Trois Tiers (c) Base de données dans les nuages

Nous pouvons noter différentes évolutions générées par des améliorations dans les matériels disponibles :

- L'accroissement des performances notamment fondées sur les mémoires vives de plus en plus massives, et des mémoires flash encore plus massives ;
- L'utilisation de plus en plus de parallélisme massif dans des grappes de machines pour traiter d'énormes volumes de données. On parle parfois de "big data" ;
- Pour simplifier la gestion de données, on tend à la déporter dans les nuages (le cloud), c'est-à-dire à mettre ses données dans des grappes de machines gérées par des spécialistes comme Apple, Google, Amazon,…

II. Le modèle relationnel

1. Généralités

Notions de modèle de données :

Un modèle est un ensemble d'outils utilisés pour décrire et manipuler des données

Modèle relationnel

Créé par CODD (IBM 69 /70).

- La majorité des SGBD actuels sont basés sur ce modèle.
- Dispose d'un Langage de Description des Données (LDD) et d'un Langage de Manipulation des Données (LMD).
- Principe simple : 1 seul concept (relation ou table) pour décrire les données et les liens entre ses données.
- Rigoureusement défini par la notion d'ensemble
- SQL : langage standard de description et de manipulation des données

Concepts du modèle

Table (relation) : Vision tabulaire du relationnel

⇨ Les données (le schéma logique) sont représentées dans une table

Exemple : la table OUVRAGES décrit un ouvrage

Cote	Titre	Editeur	Année	NbExempl aire	Thème
12TA1	Réseau informatiques	Eyrolles	998	10	Interconnexion, Réseau, Internet
13GO1	Algorithmes génétiques	Addison Wesley	1994	5	AG, Informatique évolutionniste
15TA2	Système d'exploitation	Eyrolles	1993	6	UNIX, SE, Windows

⇨ Attribut : nom donné à une colonne d'une table (exemple cote, Titre, Editeur, etc.). La première ligne de la table comporte ses attributs.

⇨ Nom de la table (ou de la relation) : OUVRAGES

⇨ Tuple(ou n-uplet) : nom donné à une ligne comportant des valeurs saisies. (tuple: 12TA1 ; Réseaux informatiques ; Eyrolles ; 1998 ; 10 ; Interconnexion réseau Internet).

⇨ Extension d'une table : le contenu de la table tous les tuples

⇨ Cardinalité : nombre de tuples de la relation. (Exemple la cardinalité dans OUVR AGES est 3)

⇨ L'ordre des lignes et des colonnes n'est pas significatif

⇨ Pas de lignes identiques

⇨ Une case une valeur

- Attribut : nom donné à une colonne d'une relation il prend ses valeurs dans un domaine
- Domaine : ensemble de valeurs possibles prises par les attributs

Exemples:

⇨ ENTIER, REEL, CHAINES DE CARACTERES

⇨ SALAIRE = {4 000. . 1 00 000}

⇨ COULEUR = {BLEU, BLANC , ROUGE}

- ⇨ POINT = {(X: REEL ,Y:REEL)}
- ⇨ TRIANGL E = {(P1:POINT,P2: POINT,P3:POINT)}

Notions de clé primaire

Clé primaire : Groupe d'attributs minimum qui détermine un tuple d'une manière unique dans la table

Exemple : le numéro de la CIN, CNE

La clé de la table OUVRAGES est l'attribut «cote», car la cote permet de déterminer de façon unique une ligne de la table.

ATTENTION : la clé se détermine par rapport à toutes les valeurs possibles de l'attribut (ou les attributs) formant la clé primaire, et surtout pas par rapport aux valeurs déjà saisies

Remarque : toute table doit obligatoirement avoir une clé primaire

Schéma d'une table

Le schéma d'une table, appelé aussi le schéma en intention, comporte le nom de la relation, ses attributs + format et la clé primaire.

La clé primaire est souvent soulignée (et/ou mise en gras)

Exemple : le schéma de la table OUVRAGES est OUVRAGES (cote: texte, Titre: Texte, Editeur: Texte, NbExemplaire: Numérique, Année: Date, Thème: Texte)

Problème de Redondance des données

La redondance = répétition des informations.

Exemple :

Table OUVRAGES

Cote	Titre	Editeur	Année	NbExemplaire	Thème	Nom Auteur	PréNom Auteur
12TA1	Réseau informatiques	Eyrolles	1998	10	Interconnexion, Réseau, Internet	Tenenbaum	Henri
13GO1	Algorithmes génétiques	Addison Wesley	1994	5	AG, Informatique évolutionniste	Goldberg	Stephen
13GO1	Algorithmes génétiques	Addison Wesley	1994	5	AG, Informatique évolutionniste	Holland	John
15TA2	Système d'exploitation	Eyrolles	1993	6	UNIX, SE, Windows	Cardy	Roanld
15TA2	Système d'exploitation	Eyrolles	1993	6	UNIX, SE, Windows	Dumar	Eric
15TA2	Système d'exploitation	Eyrolles	1993	6	UNIX, SE, Windows	Tenenbaum	Henri

Un des objectifs des SGBD est (de nous permettre) de représenter les données avec le moins de redondance possible.

Alors comment éliminer les redondances ?

Éliminer les redondances

Pour éliminer les répétitions dans notre exemple nous allons dans premier temps construire une table auteur comportant tous les auteurs.

La table auteur est décrit e par AUTEURS (NumAuteur, NomAuteur,PrénomAuteur). Nous avons rajouté l'attribut NumAuteur pour représenter la clé. Nu mAuteur est un numéro qui peut être donné automatiquement par le SGBD.

Table AUTEURS

NumAuteur	Nom Auteur	PréNom Auteur
1	Tenenbaum	Henri
2	Goldberg	Stephen
3	Holland	John
4	Cardy	Roanld
5	Dumar	Eric

La table OUVRAGES peut se réduire à cela

Cote	Titre	Editeur	Année	NbExemplaire	Thème	NumAuteur
12TA1	Réseau informatiques	Eyrolles	998	10	Interconnexion, Réseau, Internet	1
13GO1	Algorithmes génétiques	Addison Wesley	1994	5	AG, Informatique évolutionniste	2
13GO1	Algorithmes génétiques	Addison Wesley	1994	5	AG, Informatique évolutionniste	3
15TA2	Système d'exploitation	Eyrolles	1993	6	UNIX, SE, Windows	4
15TA2	Système d'exploitation	Eyrolles	1993	6	UNIX, SE, Windows	5
15TA2	Système d'exploitation	Eyrolles	1993	6	UNIX, SE, Windows	1

Cette représentation nous permet effectivement de réduire la table OUVRAGES il n y a que le numéro de l'auteur au lieu du nom et du prénom, mais Il y a toujours des redondances. La redondance provient du fait qu'un OUVRAG E peut avoir plusieurs auteurs.

Et pour éliminer ces redondances, nous allons construire une table ECRIT qui permet de relier les OUVRAGES et leurs AUTEURS.

Rappelons qu'un des intérêts d'un SGBD est sa possibilité de créer des liens entre les objets.

Le schéma de la table ECRIT est : ECRIT (cote, NumAuteur), il suffit donc de prendre les clés primaires des tables OUVRAGES et AUTEURS et former une nouvelle table, en l'occurrence ECRIT.

Table ECRIT

NumAuteur	Cote
1	12TA1
2	13GO1
3	13GO1
4	15TA2
5	15TA2
1	15TA2

La base de données décrivant les OUVRAGES sera composée des tables suivantes:

AUTEURS (NumAuteur, Nom, Prénom)

OUVRAGESS (cote, Titre, NbExemplaire, Année, Editeur,Thème)

ECRIT (cote, NumAuteur)

Noter que nous avons supprimé l'attribut NumAuteur de la table OUVRAGES

Clé Étrangère

Définition : Nous appelons Clé étrangère toute clé primaire apparaissant dans une autre table.

Dans notre exemple les attributs cote et NumAuteur de la table ECRIT proviennent en fait respectivement des tables OUVRAGES et AUTEURS. Ces deux Attributs sont clés primaires dans chacune de ces tables.

Exemple : NumAuteur est une clé étrangère dans la table ECRIT, cote est aussi une clé étrangère dans ECRIT

Par convention, Une clé étrangère est soulignée en pointillé (et/ou mise en italique)

Attention : la notion de clé est toujours liée à une table, un attribut (ou groupe d'attributs) est clé primaire, ou clé étrangère dans une table donnée.

Contraintes d'intégrités

Un des avantages des bases de données par rapport à une gestion de fichiers traditionnelle réside dans la possibilité d'intégrer des contraintes que doivent vérifier les données à tout instant.

Définition : Contraintes d'intégrité «sont des assertions qui doivent être vérifiées à tout moment par les données contenues dans la base de données»

Exemple : on souhaite poser les contraintes suivantes :

- Le nombre d'exemplaire de chaque OUVRAGE doit être supérieur à 0
- Chaque OUVRAGE doit avoir au moins un auteur
- Etc.

Ceci est possible grâce à la notion de contraintes d'intégrité

Il y a trois types de C.I. obligatoires :

- Contrainte de clé : une relation doit posséder une clé primaire
- Contrainte d'entité : un attribut d'une clé ne doit pas posséder de valeurs nulles (vides)
- Contrainte de référence (pour les clés étrangères), c'est une contrainte exprimée entre deux tables. Tout tuple d'une relation faisant référence à une autre relation doit se référer à un tuple qui existe :
  - ⇨ Intuitivement, cela consiste à vérifier que l'information utilisée dans un tuple pour désigner un autre tuple est valide, notamment si le tuple désigné existe bien

En d'autre terme, quand on désigne un attribut comme clé étrangère, les seules valeurs que peut prendre cet attribut sont celles qui sont déjà saisies dans la table qu'il référence

Contrainte optionnelle :

Contrainte de domaine : liée au domaine de définition d'un attribut.

Exemple : NbExemplaire > 0

Les contraintes d'intégrité sont vérifiées (exécutées) à chaque mise à jour de la base de données (ajout, suppression ou modification d'un tuple). Si, lors d'une mise à jour une contrainte n'est pas satisfaite, cette mise à jour ne peut pas avoir lieu.

Exemple

Schéma de la relation AUTEURS :

AUTEURS (NumAuteur , Nom, Prénom)

Schéma de la relation OUVRAGES :

OUVRAGES(cote , Titre, NbExemplaire, Année, NumEditeur, Thème)

Clé primaire : cote

Contrainte de domaine : NbExemplaire >0

Contrainte référentielle : OUVRAGES.NumEditeur est une clé étrangère et fait référence à EDITEURS.NumEditeur

Il suffit d'écrire :

Contrainte référentielle : OUVRAGES .NumEditeur REFERENCE EDIEURS.NumEditeur

Schéma de la table ECRIT :

ECRIT (NumAuteur,cote)

Clé primaire : NumAuteur, cote

Contraintes référentielles :

ECRIT.NumAuteur REFERENCE AUTEURS.NumAuteur

ECRIT.cote REFERENCE OUVRAGES.cote

Le fait d'écrire ECRIT.cote REFERENCE OUVRAGES.cote, c'est à dire définir l'attribut cote dans ECRIT comme clé étrangère, implique une contrainte référentielle. Ceci se traduit par ; les seules valeurs que peut prendre cote dans ECRIT sont celles qui sont déjà saisies dans cote d'OUVRAGES (c'est à dire 12TA1, 13GO1, 15TA2)

Schéma d'une base de données

Le schéma d'une base de données est composé de l'ensemble des schémas des tables (relations) définies dans cette BD

Exemple :

Schéma de la base de données permettant la gestion de notices bibliographiques est :

AUTEURS (NumAuteur , Nom, Prénom)

EDITEURS(NumEditeur, Editeur)

OUVRAGES(cote, Titre, NbExemplaire, Année, NumEditeur, Thème)

Contrainte de domaine : NbExemplaire>0

Contrainte référentielle : OUVRAGES .NumEditeur REFERENCE EDITEURS.NumEditeur

ECRIT (NumAuteur, cote)

Clé primaire : NumAuteur, cote

Contraintes référentielles :

ECRIT.NumAuteurR EFERENCE AUTEURS.NumAuteur

ECRIT.cote REFERENCE OUVRAGES.cote

III. Algèbre relationnelle

1. Principe de l'algèbre relationnelle :

Ensemble d'opérateurs s'appliquant sur l'ensemble des lignes (ou tuples) d'une (ou plusieurs) table(s)

Le résultat d'une opération (ou d'une requête) est une nouvelle table qui est exploitable à son tour par une nouvelle opération.

2. Opérations relationnelles

Une opération relationnelle agit sur une ou plusieurs tables et a pour résultat une table

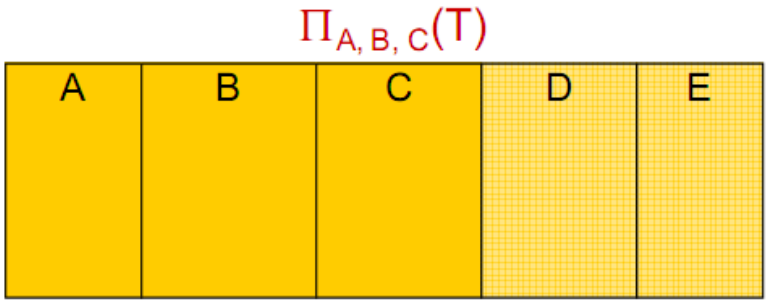
La projection et la sélection sont des opérations qui s'appliquent à une table

Les opérations ensemblistes (union, intersection, différence) ne peuvent être utilisées qu'avec deux tables ayant les mêmes attributs et fournissent une troisième table ayant les mêmes attributs

Le produit cartésien et la jointure fournissent une troisième table à partir de deux tables quelconques.

Projection π

Projeter sur un ensemble de colonnes d'une table T, revient à supprimer de la table celles qui ne sont pas dans l'ensemble :



Exemple:

Magasin	Produit	Prix
Monoprix	Pomme	2,20
Monoprix	Cerise	4,80
Franprix	Poire	3,75
Franprix	Abricot	2,95
Champion	Fraise	3,75
Monoprix	Abricot	3,50
Franprix	Pomme	2,20
Champion	Pomme	2,40

VENTES

Produit	Prix
Pomme	2,20
Cerise	4,80
Poire	3,75
Abricot	2,95
Fraise	3,75
Abricot	3,50
Pomme	2,40

$\pi_{Produit,Prix}$ (VENTES)

Magasin
Monoprix
Franprix
Champion

$\pi_{Magasin}$(VENTES)

Sélection σ

On appelle condition une assertion valant vraie ou fausse sur une ligne de table.

La sélection sur la condition cond notée σ_{cond} (T) correspond à l'algorithme suivant :

pour chaque ligne de T faire

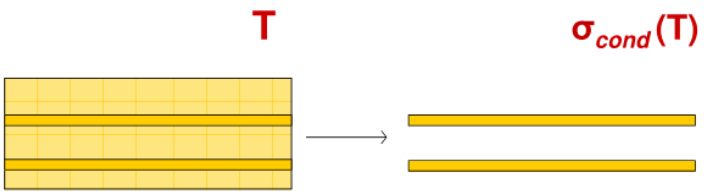
si cond (ligne) = vrai

alors garder la ligne

sinon ne pas la garder

fin pour

La sélection sur une condition consiste donc à garder les lignes de la table vérifiant la condition



Exemple:

Magasin	Produit	Prix
Monoprix	Pomme	2,20
Monoprix	Cerise	4,80
Franprix	Poire	3,75
Franprix	Abricot	2,95
Champion	Fraise	3,75
Monoprix	Abricot	3,50
Franprix	Pomme	2,20
Champion	Pomme	2,40

VENTES

Magasin	Produit	Prix
Monoprix	Pomme	2,20
Monoprix	Cerise	4,80
Monoprix	Abricot	3,50

$\sigma_{Magasin = 'Monoprix'}(VENTES)$

Magasin	Produit	Prix
Monoprix	Cerise	4,80
Franprix	Poire	3,75
Champion	Fraise	3,75
Monoprix	Abricot	3,50

$\sigma_{Prix > 3}(VENTES)$

Renommage ρ

But:

Résoudre des problèmes de compatibilité entre noms de relations ou noms d'attributs de deux relations opérandes d'une opération binaire.

Syntaxe :

Renommage d'une relation : $\rho_{nouveau_nom_Relation}\ R$

Renommage d'un attribut : $\rho_{nom_attribut:nouveau_nom}\ R$

Condition :

Il faut que le nouveau nom n'existe pas déjà dans R

Exemple:

Relation *Produits*

Produit	Prix
Pomme	2,20
Cerise	4,80
Poire	3,75
Abricot	2,95
Tomate	2,75

Relation $\rho_{Produit:Fruits}$ *Produits*

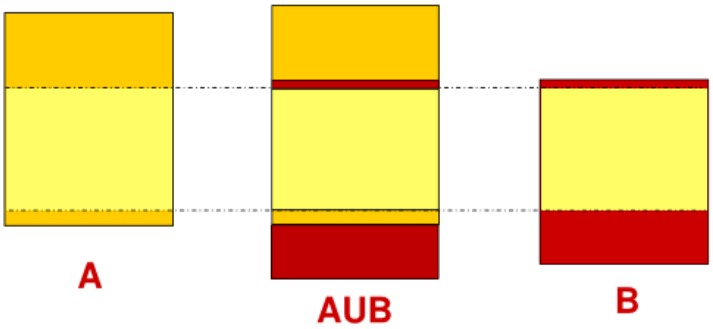
Fruits	Prix
Pomme	2,20
Cerise	4,80
Poire	3,75
Abricot	2,95
Tomate	2,75

Opérateurs ensemblistes

Union

$A \cup B$ est la table contenant toutes les lignes de A et toutes les lignes de B sans doublon.

Pour l'union il faut une structure identique.



Exemple :

Produit	Prix
Pomme	2,20
Cerise	4,80
Poire	3,75
Abricot	2,95
Tomate	2,75

FRUIT

Produit	Prix
Carotte	1,40
Poireau	1,25
Salade	2,05
Tomate	2,75

LEGUME

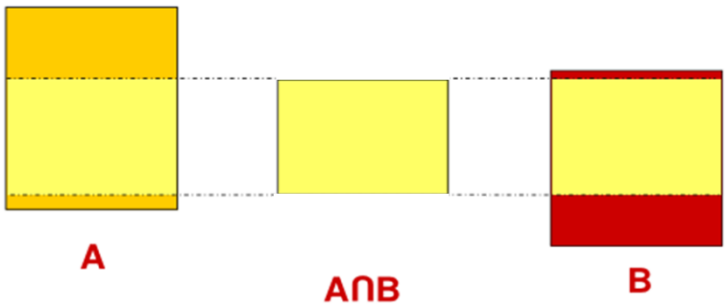
Produit	Prix
Pomme	2,20
Cerise	4,80
Poire	3,75
Abricot	2,95
Tomate	2,75
Carotte	1,40
Poireau	1,25
Salade	2,05

FRUIT U LEGUME

Intersection

$A \cap B$ contient toutes les lignes communes aux deux tables A et B.

Pour l'intersection il faut une structure identique.



Exemple :

Produit	Prix
Pomme	2,20
Cerise	4,80
Poire	3,75
Abricot	2,95
Fraise	3,75

FRUIT

Produit	Prix
Cerise	4,80
Abricot	3,50
Fraise	3,75

PRIMEUR

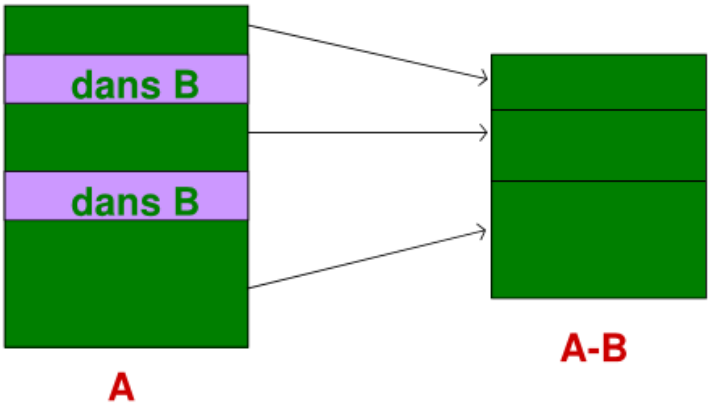
Produit	Prix
Cerise	4,80
Fraise	3,75

FRUIT ∩ PRIMEUR

Différence

$A - B$ est la table contenant toutes les lignes de A qui ne se trouvent pas dans B.

Pour la différence il faut une structure identique.



Exemple :

Produit	Prix
Pomme	2,20
Cerise	4,80
Poire	3,75
Abricot	2,95
Fraise	3,75

FRUIT

Produit	Prix
Cerise	4,80
Abricot	3,50
Fraise	3,75

PRIMEUR

Produit	Prix
Pomme	2,20
Poire	3,75
Abricot	2,95

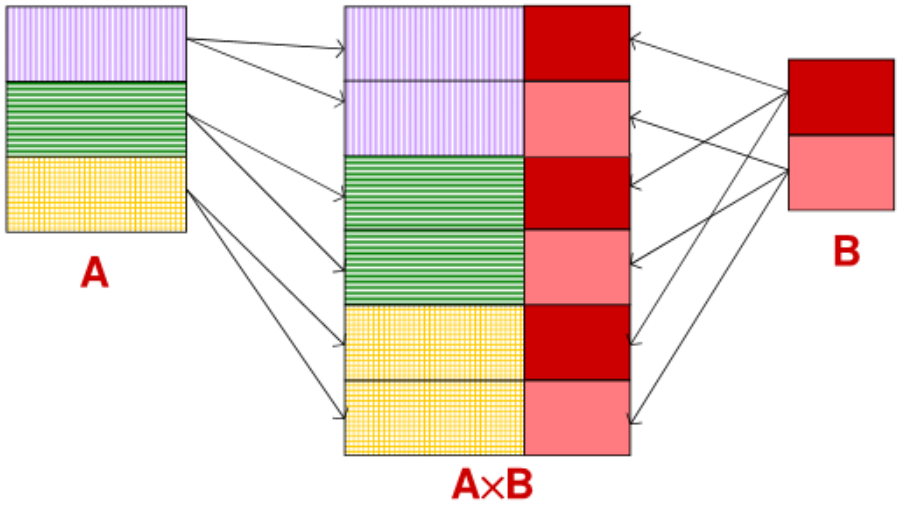
FRUIT-PRIMEUR

Produit	Prix
Abricot	3,50

PRIMEUR-FRUIT

Le produit cartésien

Pour chaque ligne de A fabriquer autant de lignes qu'il y a de lignes dans B par concaténation.



Pour deux tables T_1 et T_2 , la table$T_1 \times T_2$ est le résultat de l'algorithme suivant :

Pour chaque ligne de T_1 faire

Pour chaque ligne de T_2 faire

concaténer la ligne de T_1 avec la ligne de T_2

fin pour

fin pour

nblignes($T_1 \times T_2$) = nblignes(T_1) * nblignes(T_1)

Exemple :

Magasin	Adresse	Horaire	Offre	Prix
Monoprix	Gobelins	20H	Pomme	2,20
Franprix	Mouffetard	20H45	Abricot	2,95
Champion	Monge	22H	Fraise	3,75

SUPERMARCHE5EME

Produit	Prix
Cerise	4,80
Abricot	3,50
Fraise	3,75

PRIMEUR

Magasin	Adresse	Horaire	Offre	Prix	Produit	Prix
Monoprix	Gobelins	20H	Pomme	2,20	Cerise	4,80
Monoprix	Gobelins	20H	Pomme	2,20	Abricot	3,50
Monoprix	Gobelins	20H	Pomme	2,20	Fraise	3,75
Franprix	Mouffetard	20H45	Abricot	2,95	Cerise	4,80
Franprix	Mouffetard	20H45	Abricot	2,95	Abricot	3,50
Franprix	Mouffetard	20H45	Abricot	2,95	Fraise	3,75
Champion	Monge	22H	Fraise	3,75	Cerise	4,80
Champion	Monge	22H	Fraise	3,75	Abricot	3,50
Champion	Monge	22H	Fraise	3,75	Fraise	3,75

SUPERMARCHE5EME × PRIMEUR

La division

But :

Traiter les requêtes de style «les … tels que TOUS les …» (Exemple. "Quels sont les références des produits achetés par tous clients ?")

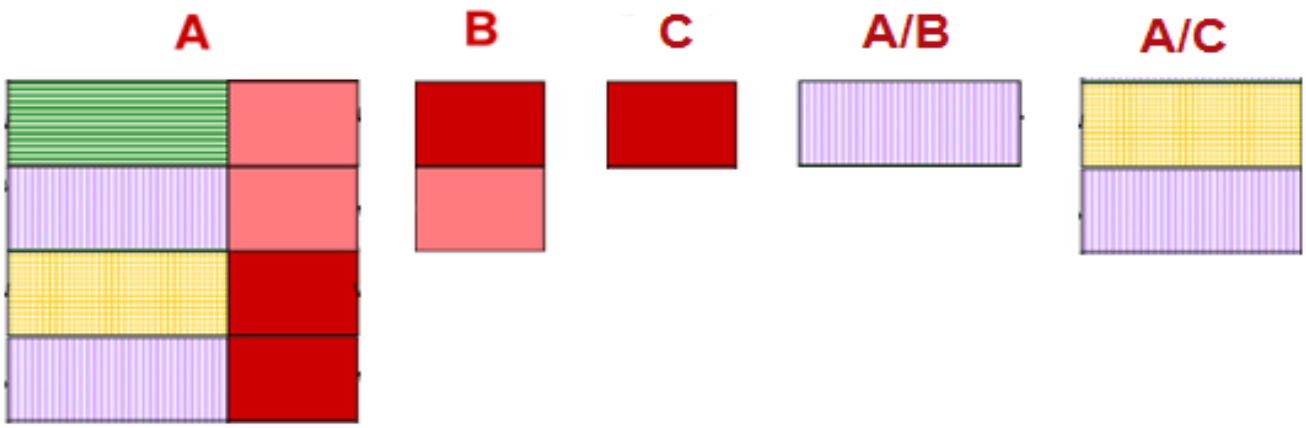
Soient les relations $R(A_1, \dots, A_n)$ et $S(A_1, \dots, A_m)$ avec $n > m$ et $A_1, \dots, A_m$ des attributs de même nom dans R et S, on définit alors la relation R/S par:

$$\mathrm{R/S} = \{< a_{m+1}, a_{m+2}, \dots, a_n > / \forall < a_1, a_2, \dots, a_m > \in S, \exists < a_1, a_2, \dots, a_m, a_{m+1}, a_{m+2}, \dots, a_n > \in R\}$$

Et on a la formule suivante :

$$R/S = R_1 - \pi_{A_1,\dots,A_n}(R_1 \times S - R), \text{ avec } R_1 = \pi_{A_1,\dots,A_n} R$$

Et $R \times S / S = R$



Exemple :

R

A	B	C
1	1	1
1	2	0
1	2	1
1	3	0
2	1	1
2	3	3
3	1	1
3	2	0
3	2	1

S

B	C
1	1
2	0

S'

B	C
1	1

R/S

A
1
3

R/S'

A
1
2
3

S"

B	C
3	5

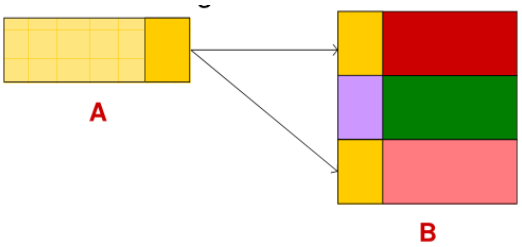
R/S"

A
/

La jointure ⋈

C'est l'opération permettant de « coller » au bout des lignes de la table A toutes les lignes de la table B vérifiant la condition de jointure.

Le cas le plus fréquent est celui où la condition est l'égalité de deux attributs.



On appelle jointure de T_1 et T_2 sur la condition *cond* la sélection sur *cond* effectuée sur $T_1 \times T_2$:

$$T_1 \bowtie_{cond} T_2 = \sigma_{cond} (T_1 \times T_2)$$

La jointure est le résultat de l'algorithme suivant :

pour chaque ligne de T_1 faire

pour chaque ligne de T_2 faire

lig = concaténer ligne T_1 et ligne T_2

si cond(lig) = vrai garder lig

fin pour

fin pour

Exemple :

Magasin	Adresse	Horaire	Offre	Prix	Produit	Prix
Franprix	Mouffetard	20H45	Abricot	2,95	Abricot	3,50
Champion	Monge	22H	Fraise	3,75	Fraise	3,75

SUPERMARCHE5EME ⋈ PRIMEUR

Offre = Produit

SUPERMARCHE5EME.Prix <= PRODUIT.Prix

Θ –jointure

La Θ –jointure est une jointure dans laquelle l'expression logique *cond* est une simple comparaison en utilisant les opérateurs de condition $(>, \geq, <, \leq, \neq, =)$ entre un attribut A_1 de la relation R_1 et un attribut A_2 de la relation R_2

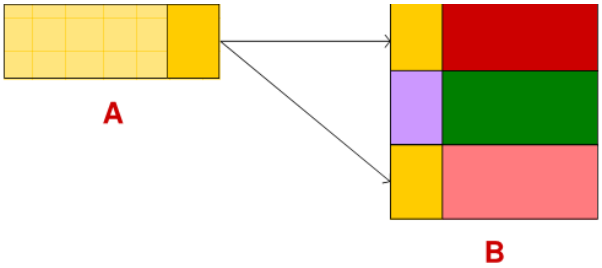
Equi-jointure

Une équi-jointure est une Θ –jointure dans laquelle l'expression logique *cond* est un test d'égalité entre un attribut A_1 de la relation R_1 et un attribut A_2 de la relation R_2. L'équi-jointure est notée :

$$R_1 \bowtie_{A_1=A_2} R_2$$

Jointure naturelle

Une jointure naturelle est une jointure dans laquelle l'expression logique *cond* est un test d'égalité entre les attributs qui portent le même nom et même domaine dans les relations R_1 et R_2. Dans la relation construite, ces attributs ne sont pas dupliqués, mais fusionnés en une seule colonne par couple d'attributs. La jointure naturelle est notée: $R_1 \bowtie R_2$. Si la jointure ne doit porter que sur un sous-ensemble des attributs communs à R_1 et R_2 il faut préciser explicitement ces attributs de la manière suivante : $R_1 \bowtie_{A_1,A_2,\ldots A_n} R_2$



Exemple :

numEtu	nom	groupeTP
10	Martin	211
11	Videau	221
22	Durand	221
32	Rossi	211

ETUDIANT

groupeTP	resProjet
211	Fournier
221	Astier

PROJET

numEtu	nom	groupeTP	resProjet
10	Martin	211	Fournier
11	Videau	221	Astier
22	Durand	221	Astier
32	Rossi	211	Fournier

ETUDIANT ⋈ PROJET

Jointure externe

Parfois, perdre les tuples défaillants n'est pas convenable...

Une jointure externe est alors une opération de jointure permettant de retrouver aussi les tuples défaillants, en mettant des valeurs "inconnu" (ou $\perp$) pour les attributs qui ne sont définis pas dans les résultats.

On a deux variantes : jointure externe gauche et droite. Formellement, en partant du cadre des jointures naturelles :

$$R_1 \overset{\circ}{\underset{L}{\bowtie}} R_2 = \{(t_1, \dots, t_n, u_{k+1}, \dots, u_m) | (t_1, \dots, t_n) \in R_1, si\ (u_{k+1}, \dots, u_m) \notin R_2 alors\ u_{k+1} = \dots = u_m = \perp\}$$

Une définition similaire pour la jointure externe droite $R_1 \overset{\circ}{\underset{R}{\bowtie}} R_2$, ainsi que pour celle bilatérale

Exemple :

$R_1=$

A	B	C
1	2	5
3	4	6
2	5	12
6	7	8

$R_2=$

B	C	D	E
2	5	6	3
4	6	8	10
5	7	10	3
4	6	7	8
3	4	6	8

$R_1 \overset{\circ}{\underset{L}{\bowtie}} R_2=$

A	B	C	D	E
1	2	5	6	3
3	4	6	8	10
3	4	6	7	8
2	5	12	⊥	⊥
6	7	8	⊥	⊥

$R_1 \overset{\circ}{\underset{R}{\bowtie}} R_2=$

A	B	C	D	E
1	2	5	6	3
3	4	6	8	10
3	4	6	7	8
⊥	5	7	10	3
⊥	3	4	6	8

$R_1 \overset{\circ}{\bowtie} R_2=$

A	B	C	D	E
1	2	5	6	3
3	4	6	8	10
3	4	6	7	8
2	5	12	⊥	⊥
6	7	8	⊥	⊥
⊥	5	7	10	3
⊥	3	4	6	8

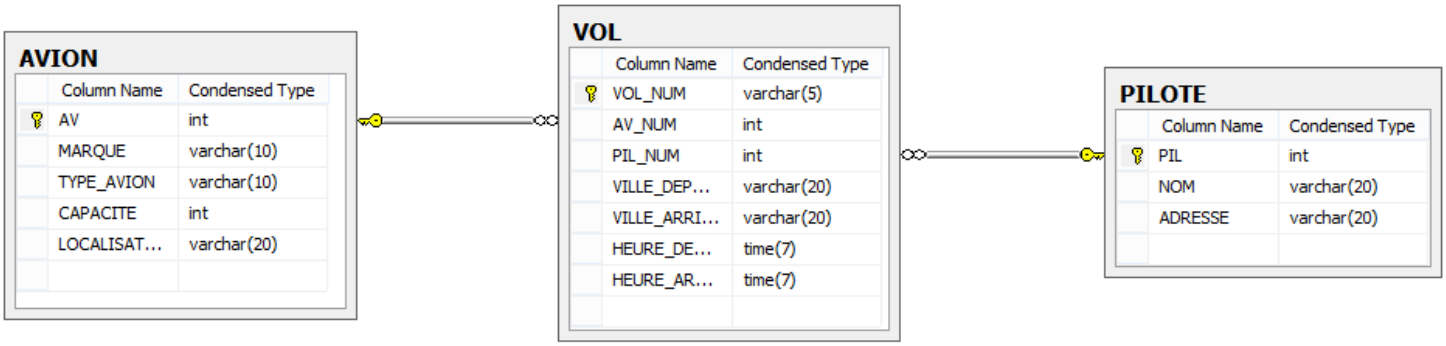
La formulation de contraintes

Une ***contrainte*** est une expression qui doit s'évaluer à l'ensemble vide avant et après toute opération :

$R = \emptyset$.

Exemples :

Transport Aerien



1. Exprimez le fait que PIL_NUM de la table VOL est une clé étrangère qui fait référence au PIL de la table PILOTE :

$$\boldsymbol{\pi_{PIL_NUM}(VOL) \backslash \pi_{PIL}(PILOTE) = \emptyset}$$

Si le résultat est différent de l'ensemble vide alors il existe un PIL_NUM qui figure dans la table VOL mais pas dans la table PILOTE ce qui contredit la contrainte de clé étrangère.

2. Exprimez le fait que AV est une clé primaire dans Avion :

$$\boldsymbol{\sigma_{AVION1.AV=AVION2.AV \wedge AVION1.CAPACITE \neq AVION2.CAPACITE}(\rho_{AVION1}(AVION) \times \rho_{AVION2}(AVION)) = \emptyset}$$

C-à-d. $\boldsymbol{\rho_{AVION1}(AVION) \bowtie_{AVION1.AV=AVION2.AV \wedge AVION1.CAPACITE \neq AVION2.CAPACITE} \rho_{AVION2}(AVION) = \emptyset}$

Groupement et agrégation

1. Opération de groupement $\boldsymbol{\gamma_L(R)}$ représente l'ensemble de tuples construits comme suit :

Regrouper d'abord les tuples de $\boldsymbol{R}$ en un ensemble de groupes, chaque groupe ayant une seule combinaison de valeurs pour les attributs dans $\boldsymbol{L}$

Pour chaque élément de l'ensemble de groupes, produire un seul tuple dans le résultat.

Exemple :

$R =$

A	B	C
1	2	3
1	2	4
1	1	3
2	1	3
2	1	2

$\gamma_{A,B}(R) =$

A	B
1	2
1	1
2	1

Sur l'exemple ci-dessus, les groupes sont faits selon les attributs (A,B) : on a donc trois groupes :

(1,2)

(1,1)

(2,1)

Chaque groupe n'est représenté ***qu'une seule fois*** dans le résultat final.

On peut également effectuer des calculs en introduisant les opérations agrégées : SUM, AVG (moyenne), MIN, MAX, COUNT,…

2. Opération de groupement avec calcul $\boldsymbol{\gamma_{L,Op\rightarrow NouvAttr}(R)}$ se construit comme suit :

Regrouper d'abord les tuples de $\boldsymbol{R}$ en un ensemble de groupes, chaque groupe ayant une seule combinaison de valeurs pour les attributs dans $\boldsymbol{L}$

A l'intérieur de chaque groupe, appliquer l'opérateur d'agrégation $\boldsymbol{Op}$ et produire, avec les résultats, une nouvelle colonne $\boldsymbol{NouvAttr}$ contenant le résultat.

Enfin, produire un tuple pour chaque combinaison de valeurs présente pour les attributs dans la liste $\boldsymbol{L}$ plus $\boldsymbol{NouvAttr}$.

Remarque :

Il est possible d'omettre la liste $\boldsymbol{L}$: $\boldsymbol{\gamma_{Op\rightarrow NouvAttr}(R)}$ dans ce cas l'opération s'appliquera sans calcul de groupe. Par exemple pour calculer la somme d'une colonne de toute une table ou compter le nom d'éléments dans une table.

Exemples :

1. $\boldsymbol{\gamma_{A,SUM(B)\rightarrow SB}(R)}$ établie des regroupements sur $\boldsymbol{A}$ et somme $\boldsymbol{B}$ à l'intérieur de chacun d'eux.

$R =$

A	B	C
1	2	3
1	2	4
1	1	3
2	1	3
2	1	2

$\gamma_{A,SUM(B) \rightarrow SB}(R) =$

A	SB
1	5
2	2

2. $\gamma_{SUM(B) \rightarrow Total}(R)$ somme toute la colonne B (sans effectuer de groupes).

$\gamma_{SUM(B) \rightarrow total}(R) =$

total
7

3. $\gamma_{COUNT(A) \rightarrow nbElementA}(R)$ calcule le nombre de lignes dans R pour lesquelles l'attribut A a une valeur différente de $NULL$ (sans faire de groupes).

$\gamma_{COUNT(A) \rightarrow nbElementA}(R) =$

nbElementA
5

IV. Le langage S.Q.L.

1. Introduction

Historique

S.Q.L. est un langage structuré permettant d'interroger et de modifier les données contenues dans une base de données relationnelle.

S.Q.L. signifie Structured Query Language. Il est issu de SEQUEL : Structured English Query Language.

C'est le premier langage pour les S.G.B.D Relationnels. Il a été développé par IBM en 1970 pour système R, son 1er SGBDR.

S.Q.L. a été reconnu par l'ANSI (Association de Normalisation des Systèmes d'Information) puis imposé comme norme. Il n'existe pas de S.G.B.D.R sans S.Q.L.

Malheureusement, malgré la norme S.Q.L., il existe un ensemble de dialectes. Les différences entre ces différents dialectes sont souvent minimes et tous respectent un minimum commun : ce que nous allons étudier ici.

Définition

S.Q.L. est un langage relationnel qui permet d'effectuer les tâches suivantes :

- Définition et modification de la structure de la base de données
- Interrogation et modification non procédurale (c'est à dire interactive) de la base de données
- Contrôle de sécurité et d'intégrité de la base.

S.Q.L. est un langage interactif, mais il peut aussi être intégré dans un langage de programmation pour le développement d'applications.

S.Q.L. n'est pas le meilleur langage, en particulier pour la manipulation des données, mais c'est un standard.

Dans tout ce qui suit les exemples seront donnés par rapport à la base de données TRANSPORT_AERIEN dont le schéma est la suivante :

AVION(<u>AV,</u> MARQUE, TYPE_AVION, CAPACITE, LOCALISATION)
PILOTE(<u>PIL</u>, NOM, ADRESSE)
VOL(<u>VOL_NUM</u>,PIL_NUM#,AV_NUM#,VILLE_DEPART,VILLE_ARRIVEE,HEURE_DEPART,HEURE_ARRIVEE)

AV : numéro d'avion MARQUE : marque de l'avion TYPE_AVION : type de l'avion CAPACITE : capacité en nb de passagers LOCALISATION : ville où est basée l'avion	PIL : numéro du pilote NOM : nom du pilote ADRESSE : adresse du pilote	VOL_NUM : numéro du vol VILLE_DEPART : ville départ VILLE_ARRIVEE : ville d'arrivée HEURE_DEPART: heure de départ HEURE_ARRIVEE : heure d'arrivée

Transport Aerien

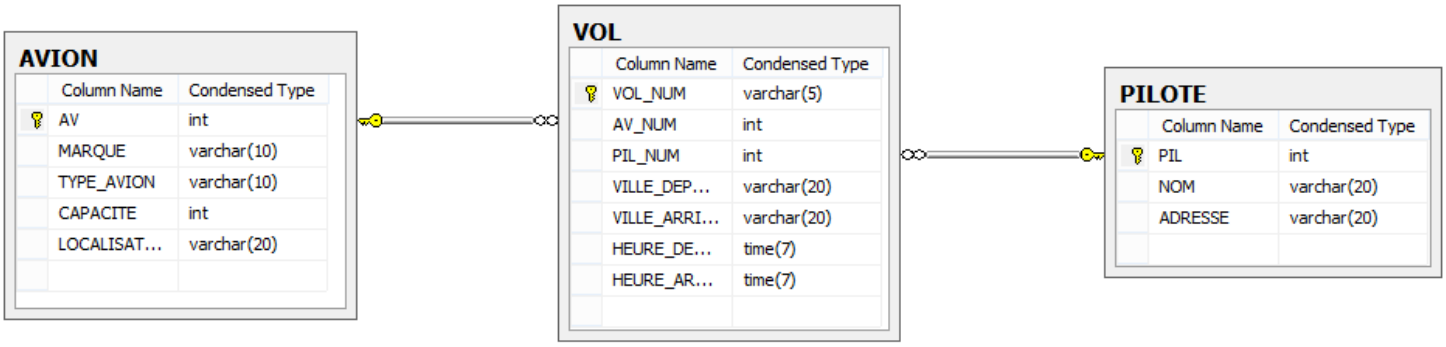


TABLE AVION

	AV	MARQUE	TYPE_AVION	CAPACITE	LOCALISATION
1	100	AIRBUS	A320	300	Rabat
2	101	BOIENG	B707	250	Laâyoune
3	102	AIRBUS	A320	300	Casablanca
4	103	CARAVELLE	CARAVELLE	200	Casablanca
5	104	BOEING	B747	400	Laâyoune
6	105	AIRBUS	A320	300	Agadir
7	106	ATR	ATR42	50	Laâyoune
8	107	BOEING	B727	300	Marrakech
9	108	BOEING	B727	300	Fès
10	109	AIRBUS	A340	350	Tanger
11	110	BOIENG	B707	300	Laâyoune

TABLE PILOTE

	PIL	NOM	ADRESSE
1	1	AHMED	Rabat
2	2	MOHAMED	Laâyoune
3	3	SAID	Agadir
4	4	MUSTAPHA	Fès
5	5	SALEH	Laâyoune
6	6	KHADIJA	Casablanca
7	7	MOHAMED SALEM	Marrakech
8	8	FATMA	Tanger
9	9	ABDELLAH	Laâyoune
10	10	BRAHIM	RABAT
11	11	SAMID	Laâyoune
12	12	imane	casablanca
13	13	ZINEB	RABAT

TABLE VOL

	VOL_NUM	AV_NUM	PIL_NUM	VILLE_DEPART	VILLE_ARRIVEE	HEURE_DEPART	HEURE_ARRIVEE
1	IT100	100	1	RABAT	LAÂYOUNE	07:30:00.0000000	09:00:00.0000000
2	IT101	100	2	LAÂYOUNE	CASABLANCA	17:30:00.0000000	19:00:00.0000000
3	IT102	101	1	LAÂYOUNE	RABAT	02:00:00.0000000	03:20:00.0000000
4	IT103	105	3	AGADIR	CASABLANCA	17:40:00.0000000	18:15:00.0000000
5	IT104	105	3	CASABLANCA	AGADIR	10:00:00.0000000	10:35:00.0000000
6	IT105	107	7	MARRAKECH	LAÂYOUNE	13:05:00.0000000	13:50:00.0000000
7	IT106	109	8	TANGER	LAÂYOUNE	15:30:00.0000000	17:00:00.0000000
8	IT107	106	9	LAÂYOUNE	DAKHLA	12:30:00.0000000	13:45:00.0000000
9	IT108	106	9	DAKHLA	LAÂYOUNE	07:30:00.0000000	09:00:00.0000000
10	IT109	107	7	LAÂYOUNE	MARRAKECH	00:30:00.0000000	01:45:00.0000000
11	IT111	101	4	RABAT	FÈS	08:30:00.0000000	09:10:00.0000000

2. Le langage SQL

Langage de Description de Données

CREATE

CREATE DATABASE

Création d'une base de données.

```
CREATE DATABASE TRANSPORT_AERIEN;
```

CREATE TABLE

Création de la description d'une table avec la liste de tous ses attributs et leur type.

```
CREATE TABLE AVION(
AV INT not null,
MARQUE VARCHAR(10) not null,
TYPE_AVION VARCHAR(10) not null,
CAPACITE INT not null,
LOCALISATION VARCHAR(20) not null,
PRIMARY KEY (AV)
);
CREATE TABLE PILOTE(
PIL INT IDENTITY(1,1) PRIMARY KEY,
NOM VARCHAR(20) not null,
ADRESSE VARCHAR(20) not null
);
CREATE TABLE VOL(
VOL_NUM VARCHAR(5) not null,
AV_NUM INT REFERENCES AVION(AV)  not null,
PIL_NUM INT REFERENCES PILOTE(PIL) not null,
VILLE_DEPART VARCHAR(20) not null,
VILLE_ARRIVEE VARCHAR(20) not null,
HEURE_DEPART TIME,
HEURE_ARRIVEE TIME,
PRIMARY KEY (VOL_NOM)
);
```

CREATE VIEW

```
CREATE VIEW VOL_RABAT    (VOL_NUM,PIL_NUM,AV_NUM,VD,VA,HD,HA)AS
```

```
SELECT
VOL_NUM,PIL_NUM,AV_NUM,VILLE_DEPART,VILLE_ARRIVEE,HEURE_DEPART,HEURE_AR
RIVEE
FROM VOL
WHERE VILLE_DEPART='RABAT'
```

Une vue est utilisée pour :

- Obtenir une table intermédiaire constituant un extrait d'une ou plusieurs tables
- Une restriction d'accès à la table pour l'utilisateur, c'est-à-dire une sécurité des données accrue.

CREATE INDEX

```
CREATE INDEX VILLE_D ON VOL (VILLE_DEPART);
CREATE UNIQUE INDEX CLE ON PILOTE (PIL);
```

La création d'index sert à améliorer les performances lors de recherche dans la table sur cet attribut (`VILLE_DEPART` dans VOL ou PIL dans PILOTE)

DROP

DROP est utilisé pour supprimer une définition de table, de vue ou d'index.

```
DROP TABLE VOL;
DROP INDEX CLE;
```

ALTER

ALTER est utilisé pour modifier une définition de table, de vue ou d'index.

```
ALTER TABLE PILOTE ADD SALAIRE SMALLINT;
ALTER TABLE PILOTE DROP COLUMN SALAIRE;
ALTER TABLE PILOTE
  ALTER COLUMN NOM NVARCHAR(20) NOT NULL;
```

Langage de Manipulation des Données

Sélection

Le SELECT sert à interroger les données et à les présenter triées et/ou regroupées suivant certains critères.

Il s'agit de retrouver les enregistrements qui vérifient certains critères

Exemple : trouver les vols qui arrivent à 19 heures?

```
SELECT *
FROM VOL
WHERE HEURE_ARRIVEE>= '19:00'
```

Projection

Il s'agit de n'afficher que certains attributs dans une table

Exemple : lister les numéros de vols?

```
SELECT PIL
FROM PILOTE;
```

Opérateurs ensemblistes

On peut travailler en SQL avec la méthode ensembliste. Pour cela, il faut considérer que chaque table est un **ensemble** de tuples et que chaque SELECT produit un ensemble de tuples.

On utilise ensuite les opérateurs classiques sur les ensembles :

- UNION
- IN
- NOT IN

Exemples :

- Liste des avions AIRBUS allant à Laâyoune (INTERSECTION)

```
(SELECT AV FROM AVION
WHERE MARQUE = 'AIRBUS')
INTERSECT
(SELECT AV_NUM FROM VOL
WHERE VILLE_ARRIVEE = 'LAÂYOUNE')
```

Qui peut aussi s'écrire

```
SELECT * FROM AVION
WHERE MARQUE = 'AIRBUS'
                    AND AV IN
                         (SELECT AV_NUM FROM VOL
                              WHERE VILLE_ARRIVEE = 'Laâyoune')
```

C'est à dire trouver tous les avions dont le numéro appartient à l'ensemble des numéros d'avions des vols à destination de Laâyoune.

- Liste des avions AIRBUS n'allant pas à Laâyoune (DIFFERENCE) MINUS en ORACLE

```
(SELECT AV FROM AVION
WHERE MARQUE='AIRBUS')
EXCEPT
(SELECT AV_NUM FROM VOL
WHERE VILLE_ARRIVEE = 'Laâyoune')
```

Qui peut aussi s'écrire

```
SELECT AV FROM AVION
WHERE MARQUE='AIRBUS'
AND AV NOT IN
(SELECT AV_NUM FROM VOL
WHERE VILLE_ARRIVEE = 'Laâyoune')
```

C'est à dire trouver tous les avions dont le numéro n'appartient pas à l'ensemble des numéros d'avions des vols à destination de Laâyoune.

- Liste des avions de marque AIRBUS ou de plus de 200 places (UNION)

```
(SELECT AV FROM AVION
WHERE MARQUE='AIRBUS')
UNION
(SELECT AV FROM AVION
```

```
WHERE CAPACITE > 200)
```

Qui peut aussi s'écrire

```
SELECT * FROM AVION
WHERE MARQUE='AIRBUS' OR CAPACITE > 200
```

INSERT

Permet d'ajouter un enregistrement dans une table.

```
INSERT INTO AVION VALUES (101,'BOIENG','B707',250,'Laâyoune')
```

DELETE

Permet de supprimer un enregistrement d'une table.

```
DELETE FROM VOL
WHERE VILLE_DEPART='RABAT'
```

Et pour vider une table:

```
TRUNCATE TABLE VOL;
```

UPDATE

Permet de modifier les valeurs de certains attributs d'un ou plusieurs enregistrements dans une table.

Exemple : modifier la capacité de tous les avions basés à Rabat

```
UPDATE AVION
SET CAPACITE=CAPACITE*1.1
WHERE LOCALISATION='RABAT';
```

3. La sélection

DISTINCT

Le résultat d'un SELECT étant un ensemble, il peut y avoir des doublons. Le mot clé DISTINCT permet de préciser que l'on ne veut qu'un seul exemplaire de ces enregistrements.

Exemple : liste des types d'avions de plus de 250 places.

```
SELECT TYPE_AVION , CAPACITE
FROM AVION
WHERE CAPACITE > 250;
```

Les types A320 et B727 vont apparaître plusieurs fois:

	TYPE_AVION	CAPACITE
1	A320	300
2	A320	300
3	B747	400
4	A320	300
5	B727	300
6	B727	300
7	A340	350
8	B707	300

```
SELECT DISTINCT TYPE_AVION , CAPACITE
FROM AVION
WHERE CAPACITE > 250;
```

On obtient :

	TYPE_AVION	CAPACITE
1	A320	300
2	A340	350
3	B707	300
4	B727	300
5	B747	400

Fonctions intégrées

Des fonctions intégrées peuvent être combinées à la liste des attributs.

SUM : Somme des valeurs de l'attribut pour les enregistrements sélectionnés

MIN : Minimum des valeurs de l'attribut pour les enregistrements sélectionnés

MAX : Maximum des valeurs de l'attribut pour les enregistrements sélectionnés

AVG : Moyenne des valeurs de l'attribut pour les enregistrements sélectionnés

COUNT : Nombre d'enregistrements sélectionnés

- Nombre d'avions dans la table

```
SELECT COUNT(AV)
FROM AVION;
```

- Nombre d'avions en service

```
SELECT COUNT(DISTINCT AV_NUM)
FROM VOL;
```

- Avion de plus petite capacité

```
SELECT TYPE_AVION
FROM AVION
WHERE CAPACITE= (SELECT MIN(CAPACITE) FROM AVION);
```

- Capacités min et max des boeings

```
SELECT MIN(CAPACITE )AS [CAP MIN], MAX(CAPACITE )AS[CAP MAX]
FROM AVION
WHERE MARQUE='BOEING';
```

- Capacité moyenne des avions localisés à Laâyoune

```
SELECT AVG(CAPACITE)
FROM AVION
WHERE LOCALISATION='LAÂYOUNE';
```

- Capacité totale des avions

```
SELECT SUM(CAPACITE)
FROM AVION;
```

La Jointure

La **jointure** consiste à rechercher entre deux tables ayant un attribut commun (même type et même domaine de définition) tous les tuples pour lesquels ces attributs ont la même valeur.

Pour représenter la jointure il y a 2 méthodes :

- la méthode ensembliste qui réalise l'intersection de deux ensembles
- la méthode prédicative qui vérifie l'égalité de deux attributs

Méthode ensembliste

```
SELECT liste d'attributs
FROM table1
WHERE attribut de jointure
IN
    (SELECT attribut de jointure
       FROM table2
       WHERE condition)
```

Le SELECT qui suit le IN est celui qui est exécuté le premier. Le résultat est un ensemble de valeurs de l'attribut de jointure. On extrait ensuite de table1 tous les enregistrements dont la valeur de cet attribut appartient à l'ensemble.

Exemple :

- Nom des pilotes assurant un vol au départ de Laâyoune

```
SELECT NOM
FROM PILOTE
WHERE PIL IN   (SELECT PIL_NUM
               FROM VOL
               WHERE VILLE_DEPART='Laâyoune');
```

- Nom des pilotes conduisant un Airbus

```
SELECT NOM
FROM PILOTE
WHERE PIL IN   (SELECT PIL_NUM
                    FROM VOL
                    WHERE AV_NUM IN     (SELECT AV
                                        FROM AVION

                                        WHERE MARQUE='Airbus'));
```

Méthode prédicative

Il y a un seul SELECT pour toute la requête.

La liste de toutes les tables concernées apparaît dans le FROM

La traduction de la jointure se fait par l'équation de jointure (égalité entre 2 attributs)

Exemples :

- Type et capacité des avions en service

```
SELECT AVION.AV, MARQUE,CAPACITE
FROM VOL,AVION
WHERE VOL.AV_NUM=AVION.AV;
```

- Nom des pilotes en service

```
SELECT DISTINCT NOM
FROM VOL, PILOTE
WHERE VOL.PIL_NUM =PILOTE.PIL;
```

- Nom des pilotes assurant un vol au départ de Laâyoune

```
SELECT NOM
FROM VOL, PILOTE
WHERE     VOL.PIL_NUM= PILOTE.PIL
                    AND VILLE_DEPART='Laâyoune';
```

- Nom des pilotes conduisant un Airbus

```
SELECT NOM
FROM VOL, PILOTE, AVION
WHERE     VOL.PIL_NUM= PILOTE.PIL
                    AND VOL.AV_NUM=AVION.AV
                    AND MARQUE='Airbus';
```

Auto-jointure

L'auto-jointure est la jointure entre une table et elle-même, pour sélectionner des enregistrements correspondant à d'autres de la même table.

Exemple :

- Nom des avions ayant même capacité

```
SELECT AVION1.TYPE_AVION, AVION1.CAPACITE,
                     'Même capacité que:', AVION2.TYPE_AVION
FROM AVION AVION1, AVION AVION2
WHERE AVION1.AV<>AVION2.AV
                    AND AVION1.CAPACITE = AVION2.CAPACITE;
```

Opérateur de partitionnement

Group by

Ce mot clé permet d'effectuer des regroupements sur lesquels s'opèrent les fonctions intégrées.

Exemples :

- Nombre d'avions de chaque marque

```
SELECT MARQUE, COUNT(AV)
FROM AVION
GROUP BY MARQUE;
```

- Nombre de pilotes différents pour chaque avion en service

```
SELECT AV_NUM, COUNT(DISTINCT PIL_NUM)
FROM VOL
GROUP BY AV_NUM;
```

Having

Le mot clé **HAVING** associé au GROUP BY permet d'exprimer des conditions sur les regroupements :

- Numéros des pilotes assurant plus d'un vol

```
SELECT PIL_NUM
FROM VOL
GROUP BY PIL_NUM
HAVING COUNT(VOL_NUM)>1
ORDER BY PIL_NUM;
```

- Numéros des pilotes et Nombre de vols assurés au départ de Rabat

```
SELECT PIL_NUM , COUNT(VOL_NUM) AS NBR_VOL_RABAT
FROM VOL
WHERE VILLE_DEPART='RABAT'
GROUP BY PIL_NUM
HAVING COUNT(VOL)>=1;
```

- Numéros des pilotes et nombre de vols qui ont plusieurs vols dont un au moins au départ de Rabat

```
SELECT PIL_NUM , COUNT(VOL_NUM)AS NBR_VOL
FROM VOL
WHERE PIL_NUM IN (SELECT PIL_NUM
FROM VOL
WHERE VILLE_DEPART='RABAT')
GROUP BY PIL_NUM
HAVING COUNT(PIL_NUM)>1;
```

Opérateurs du WHERE

Pour exprimer les conditions dans la clause WHERE on dispose de certains opérateurs :

- >, <, =, <>, <=, >= pour les comparaisons
- BETWEEN
- IN et NOT IN : expriment l'appartenance (ou non) d'un tuple à l'ensemble résultat du SELECT imbriqué
- LIKE suivi d'une expression représentant un ensemble de valeurs. Dans ces expressions % désigne un ensemble de caractères, _ remplace une lettre
- EXISTS et NOT EXISTS : expriment l'appartenance (ou non) d'un tuple à l'ensemble résultat du SELECT imbriqué
- IS NULL et IS NOT NULL : testent si un attribut possède ou non une valeur

Exemples :

- Nom des pilotes dont la 2° lettre est un A

```
SELECT NOM
FROM PILOTE
WHERE NOM LIKE '_A%'
```

- Nom des pilotes en service

```
SELECT DISTINCT NOM
FROM VOL, PILOTE
WHERE PILOTE.PIL=VOL.PIL_NUM;
```

- Nom des pilotes ayant au moins un vol

```
SELECT PIL,NOM
FROM PILOTE
WHERE EXISTS (
SELECT VOL_NUM
FROM VOL
WHERE PIL_NUM=PIL
);
```

- Nom des pilotes n'ayant aucun vol

```
SELECT NOM
FROM PILOTE
WHERE NOT EXISTS (SELECT PIL
FROM VOL
WHERE PILOTE.PIL=VOL.PIL_NUM);
```

Syntaxe générale de la commande SELECT

Voici la syntaxe générale d'une commande SELECT :

```
SELECT[DISTINCT ] {* | expression [ AS nom_affiché ]} [, ...]
FROM nom_table [ [ AS ] alias ] [, ...]
[ WHERE prédicat ]
[ GROUP BY expression [, ...] ]
[HAVING condition [, ...] ]
[ {UNION | INTERSECT | EXCEPT } requête ]
[ ORDER BY expression [ ASC | DESC ] [, ...] ]
```

En fait l'ordre SQL SELECT est composé de 7 clauses dont 5 sont optionnelles :

`SELECT` **:** Cette clause permet de spécifier les attributs que l'on désire voir apparaître dans le résultat de la requête

`FROM` **:** Cette clause spécifie les tables sur lesquelles porte la requête.

`WHERE` **:** Cette clause permet de filtrer les n-uplets en imposant une condition à remplir pour qu'ils soient présents dans le résultat de la requête.

`GROUP BY` **:** Cette clause permet de définir des groupes (i.e. sous-ensemble).

`HAVING` **:** Cette clause permet de spécifier un filtre (condition de regroupement des n-uplets) portant sur les résultats.

`UNION`**,** `INTERSECT` **et** `EXCEPT` **:** Cette clause permet d'effectuer des opérations ensemblistes entre plusieurs résultats de requête (i.e. entre plusieurs SELECT).

`ORDER BY` : Cette clause permet de trier les n-uplets du résultat.

4. Requêtes pivots

Les requêtes pivots ont pris une importance considérable dans les outils d'aide à la décision, leur présentation agrégée permettant de condenser une information en une forme lisible, permettant l'analyse et la comparaison aisée de chiffres. Contraintes et leurs types

Exemple :

Le nombre des avions par marque et ses totaux par leurs villes de localisations :

```
SELECT
LOCALISATION,AIRBUS,ATR,BOEING,CARAVELLE,(AIRBUS+ATR+CARAVELLE+BOEING)
AS TOTAL  FROM
(SELECT LOCALISATION,MARQUE,AV
FROM AVION
)AS SRC

PIVOT

(
COUNT(AV)
FOR MARQUE IN (AIRBUS,ATR,BOEING,CARAVELLE)
) AS PVT
ORDER BY LOCALISATION DESC
```

	LOCALISATION	AIRBUS	ATR	BOEING	CARAVELLE	TOTAL
1	Tanger	1	0	0	0	1
2	Rabat	1	0	0	0	1
3	Marrakech	0	0	1	0	1
4	Laâyoune	0	1	3	0	4
5	Fès	0	0	1	0	1
6	Casablanca	1	0	0	1	2
7	Agadir	1	0	0	0	1

5. Introduction aux contraintes d'intégrité

Contrainte d'intégrité de domaine

Toute comparaison d'attributs n'est acceptée que si ces attributs sont définis sur le même domaine.

Le SGBD doit donc constamment s'assurer de la validité des valeurs d'un attribut. C'est pourquoi la commande de création de table doit préciser, en plus du nom, le type de chaque colonne.

Par exemple, pour la table AVION, on précisera que la VILLE_DEPART est une chaîne de caractères et l'HEURE_DEPART une heure. Lors de l'insertion de n-uplets dans cette table, le système s'assurera que les différents champs du n-uplet satisfont les contraintes d'intégrité de domaine des attributs précisées lors de

la création de la base. Si les contraintes ne sont pas satisfaites, le n-uplet n'est, tout simplement, pas inséré dans la table.

Contrainte d'intégrité de relation (ou d'entité)

Lors de l'insertion de n-uplets dans une table (i.e. une relation), il arrive qu'un attribut soit inconnu ou non défini. On introduit alors une valeur conventionnelle notée NULL et appelée valeur nulle.

Cependant, une clé primaire ne peut avoir une valeur nulle. De la même manière, une clé primaire doit toujours être unique dans une table. Cette contrainte forte qui porte sur la clé primaire est appelée contrainte d'intégrité de relation.

Tout SGBD relationnel (SGBDR) doit vérifier l'unicité et le caractère défini (NOT NULL) des valeurs de la clé primaire.

Contrainte d'intégrité de référence

Dans tout schéma relationnel, il existe deux types de relation :

- les relations qui représentent des entités de l'univers modélisé ; elles sont qualifiées de statiques, ou d'indépendantes ; les relations PILOTE et AVION en sont des exemples ;
- les relations dont l'existence des n-uplets dépendent des valeurs d'attributs situées dans d'autres relations ; il s'agit de relations dynamiques ou dépendantes ; la relation VOL en est un exemple.

Lors de l'insertion d'un n-uplet dans la relation VOL, le SGBD doit vérifier que les valeurs PIL_NUM et AV_NUM correspondent bien, respectivement, à une valeur de PIL existant dans la relation PILOTE et une valeur AV existant dans la relation AVION.

Lors de la suppression d'un n-uplet dans la relation PILOTE, le SGBD doit vérifier qu'aucun n-uplet de la relation VOL ne fait référence, par l'intermédiaire de l'attribut PIL au n-uplet que l'on cherche à supprimer. Le cas échéant, c'est-à-dire si une, ou plusieurs, valeur correspondante de PIL existe dans VOL, quatre possibilités sont envisageables :

- interdire la suppression ;
- supprimer également les n-uplets concernés dans VOL ;
- avertir l'utilisateur d'une incohérence ;
- mettre les valeurs des attributs concernés à une valeur nulle dans la table VOL, si l'opération est possible (ce qui n'est pas le cas si ces valeurs interviennent dans une clé primaire) ;

6. Créer une table : CREATE TABLE

Introduction

Une table est un ensemble de lignes et de colonnes. La création consiste à définir (en fonction de l'analyse) le nom de ces colonnes, leur format (type), la valeur par défaut à la création de la ligne (DEFAULT) et les règles de gestion s'appliquant à la colonne (CONSTRAINT).

Création simple

La commande de création de table la plus simple ne comportera que le nom et le type de chaque colonne de la table. A la création, la table sera vide, mais un certain espace lui sera alloué. La syntaxe est la suivante :

```
CREATE TABLE nom_table (nom_col1 TYPE1, nom_col2 TYPE2, ...)
```

Quand on crée une table, il faut définir les contraintes d'intégrité que devront respecter les données que l'on mettra dans la table.

Les types de données

Les types de données peuvent être :

BIT: Données de nombre entier avec une valeur de 1 ou 0.

BIGINT : Ce type permet de stocker des entiers signés codés sur 8 octets.

IDENTITY [(S, I)]: Ce type permet de stocker des entiers incrémentés(S (seed) = valeur de départ et I(increment) = valeur d'incrément). Une table ne peut contenir qu'une seule colonne d'identité. La colonne ne peut pas être mise à jour.

INTEGER : Ce type permet de stocker des entiers signés codés sur 4 octets.

SMALLINT: Ce type permet de stocker des entiers signés codés sur 2 octets.

TINYINT: Ce type permet de stocker des entiers signés codés sur 1 octet.

REAL : Ce type permet de stocker des réels comportant 6 chiffres significatifs codés sur 4 octets.

DOUBLE PRECISION : Ce type permet de stocker des réels comportant 15 chiffres significatifs codés sur 8 octets.

NUMERIC[(précision, [longueur])] : Ce type de données permet de stocker des données numériques à la fois entières et réelles avec une précision de 1000 chiffres significatifs. Longueur précise le nombre maximum de chiffres significatifs stockés et précision donne le nombre maximum de chiffres après la virgule.

CHAR(longueur) : Ce type de données permet de stocker des chaînes de caractères de longueur fixe. longueur doit être inférieur à 255, sa valeur par défaut est 1.

NATIONAL CHARACTER(longueur) Ou NCHAR(longueur): Données Unicode de longueur fixe dont la longueur maximale est égale à 4 000 caractères. Longueur par défaut = 1. La taille de stockage (en octets) correspond au double du nombre de caractères entrés.

CHARACTER VARYING(longueur) ou VARCHAR(longueur) : Ce type de données permet de stocker des chaînes de caractères de longueur variable. Longueur doit être inférieur à 2000, il n'y a pas de valeur par défaut.

NATIONAL CHARACTER VARYING(longueur) ou VARCHAR(longueur) : Données Unicode de longueur variable comptant entre 1 et 4 000 caractères. Longueur par défaut = 1. La taille de stockage (en octets) correspond au double du nombre de caractères entrés.

DATE : Ce type de données permet de stocker des données constituées d'une date.

TIMESTAMP : Ce type de données permet de stocker des données constituées d'une date et d'une heure.

BOOLEAN : Ce type de données permet de stocker des valeurs Booléenne.

MONEY : Ce type de données permet de stocker des valeurs monétaires.

TEXT : Ce type de données permet des stocker des chaînes de caractères de longueur variable.

Création avec Insertion de données

On peut insérer des données dans une table lors de sa création par la commande suivante :

```
CREATE TABLE nom_table [(nom_col1, nom_col2, ...)] AS SELECT ...
```

On peut ainsi, en un seul ordre SQL créer une table et la remplir avec des données provenant du résultat d'un SELECT. Si les types des colonnes ne sont pas spécifiés, ils correspondront à ceux du SELECT. Il en va de même pour les noms des colonnes. Le SELECT peut contenir des fonctions de groupes mais pas d'ORDER BY car les lignes d'une table ne peuvent pas être classées.

En SQL SERVER la commande est :

```
SELECT [(nom_col1, nom_col2, ...)]
INTO nom_table
FROM ...
```

Exemple :

Liste de tous les vols avec tous les informations des pilotes et avions.

```
SELECT * INTO VOL_NOM FROM
PILOTE,AVION,VOL
WHERE VOL.PIL_NUM=PILOTE.PIL
AND VOL.AV_NUM=AVION.AV;
```

7. Contraintes d'intégrité

Syntaxe

A la création d'une table, les contraintes d'intégrité se déclarent de la façon suivante :

```
CREATE TABLE table_name
(
nom_col_1 type_1 nom_contrainte_1,
nom_col_2 type_2 nom_contrainte_2,
nom_col_3 type_3 nom_contrainte_3,
....
);
```

Contraintes de colonne

Les différents contraintes de colonne que l'on peut déclarer sont les suivantes :

NOT NULL ou NULL : Interdit (NOT NULL) ou autorise (NULL) l'insertion de valeur NULL pour cet attribut.

UNIQUE : Désigne l'attribut comme clé secondaire de la table. Deux n-uplets ne peuvent recevoir des valeurs identiques pour cet attribut, mais l'insertion de valeur NULL est toutefois autorisée. Cette contrainte peut apparaître plusieurs fois dans l'instruction.

PRIMARY KEY : Désigne l'attribut comme clé primaire de la table. La clé primaire étant unique, cette contrainte ne peut apparaître qu'une seule fois dans l'instruction. La définition d'une clé primaire composée se fait par l'intermédiaire d'une contrainte de table. En fait, la contrainte PRIMARY KEY est totalement équivalente à la contrainte UNIQUE NOT NULL.

REFERENCES table [(colonne)] [ON DELETE CASCADE] : Contrainte d'intégrité référentielle pour l'attribut de la table en cours de définition. Les valeurs prises par cet attribut doivent exister dans l'attribut

colonne qui possède une contrainte PRIMARY KEY ou UNIQUE dans la table table. En l'absence de précision d'attribut colonne, l'attribut retenu est celui correspondant à la clé primaire de la table table spécifiée.

CHECK (condition) : Vérifie lors de l'insertion de n-uplets que l'attribut réalise la condition condition.

DEFAULT valeur : Permet de spécifier la valeur par défaut de l'attribut.

Contraintes de table

Les différentes contraintes de table que l'on peut déclarer sont les suivantes :

PRIMARY KEY (colonne, ...) : Désigne la concaténation des attributs cités comme clé primaire de la table. Cette contrainte ne peut apparaître qu'une seule fois dans l'instruction.

UNIQUE (colonne, ...) : Désigne la concaténation des attributs cités comme clé secondaire de la table. Dans ce cas, au moins une des colonnes participant à cette clé secondaire doit permettre de distinguer le n-uplet. Cette contrainte peut apparaître plusieurs fois dans l'instruction.

FOREIGN KEY (colonne, ...) REFERENCES table [(colonne, ...)] [ON DELETE CASCADE | SET NULL] : Contrainte d'intégrité référentielle pour un ensemble d'attributs de la table en cours de définition. Les valeurs prises par ces attributs doivent exister dans l'ensemble d'attributs spécifié et posséder une contrainte PRIMARY KEY ou UNIQUE dans la table table.

CHECK (condition) : Cette contrainte permet d'exprimer une condition qui doit exister entre plusieurs attributs de la ligne.

Les contraintes de tables portent sur plusieurs attributs de la table sur laquelle elles sont définis.

Il n'est pas possible de définir une contrainte d'intégrité utilisant des attributs provenant de deux ou plusieurs tables. Ce type de contrainte sera mis en œuvre par l'intermédiaire de déclencheurs de base de données (triggers).

Complément sur les contraintes

ON DELETE CASCADE : Demande la suppression des n-uplets dépendants, dans la table en cours de définition, quand le n-uplet contenant la clé primaire référencée est supprimé dans la table maître.

ON DELETE SET NULL : Demande la mise à NULL des attributs constituant la clé étrangère qui font référence au n-uplet supprimé dans la table maître.

La suppression d'un n-uplet dans la table maître pourra être impossible s'il existe des n-uplets dans d'autres tables référençant cette valeur de clé primaire et ne spécifiant pas l'une de ces deux options.

8. Supprimer une table : DROP TABLE

Supprimer une table revient à éliminer sa structure et toutes les données qu'elle contient. Les index associés sont également supprimés.

La syntaxe est la suivante :

```
DROP TABLE nom_table
```

9. Modifier une table : ALTER TABLE

Ajout ou modification de colonnes

```
ALTER TABLE nom_table {ADD/MODIFY} ([nom_colonne type [contrainte],
...])
```

Ajout d'une contrainte de table:

```
ALTER TABLE nom_table ADD [CONSTRAINT nom_contrainte] contrainte
```

La syntaxe de déclaration de contrainte est identique à celle vue lors de la création de table.

Si des données sont déjà présentes dans la table au moment où la contrainte d'intégrité est ajoutée, toutes les lignes doivent vérifier la contrainte. Dans le cas contraire, la contrainte n'est pas posée sur la table.

Renommer une colonne

```
ALTER TABLE nom_table RENAME COLUMN ancien_nom TO nouveau_nom
```

Et en SQL SERVER: `SP_RENAME 'nom_table.ancien_nom','nouveau_nom','COLUMN';`

Renommer une table

```
ALTER TABLE nom_table RENAME TO nouveau_nom
```

Et en SQL SERVER: `SP_RENAME 'nom_table','nouveau_nom';`

10. Exemples

Dans tout ce paragraphe, on basera les exemples sur le schéma GESTION_CINEMA qui est rappelé ci-dessous:

- CINEMA (**NUMCINEMA**, NOMCINEMA, NUMERO, RUE, VILLE)
- SALLE (**NUMCINEMA#, NUMSALLE**, CAPACITE, CLIMATISEE)
- HORAIRE (**IDHORAIRE**, HEUREDEBUT, HEUREFIN)
- SEANCE (**IDFILM#, NUMCINEMA#, NUMSALLE#, IDHORAIRE#**, TARIF)
- FILM (**IDFILM**, TITRE, ANNEE, GENRE, RESUME, IDMES)
- ARTISTE (**ID**, NOM, PRENOM, ANNEENAISSANCE)
- ROLE (**IDACTEUR#**, **IDFILM#**, NOMROLE)
- REALISATEUR (**IDMES**, NOMMES, PRENOMMES, ANNEENAISSANCEMES)

Exemple 1

Exemple simple : on restreint les valeurs possibles des attributs `capacité` et `climatisée` dans la table `SALLE`.

```
CREATE TABLE SALLE(
NUMCINEMA INTEGER NOT NULL FOREIGN KEY  REFERENCES CINEMA(NUMCINEMA),
NUMSALLE INTEGER,
CAPACITE INTEGER CHECK (CAPACITE <  300),
CLIMATISEE CHAR(1) CHECK (CLIMATISEE IN ('O','N')) DEFAULT 'N',
PRIMARY KEY (NUMCINEMA, NUMSALLE)
);
```

Il s'agit de contraintes portant sur des valeurs d'attributs. A chaque insertion d'un tuple, ou mise-à-jour de ce tuple affectant un des attributs contraints, le contrôle sera effectué. La règle est que la condition ne doit pas s'évaluer à FALSE.

Exemple 2

Voici un autre exemple illustrant l'utilisation d'une sous-requête : on souhaite remplacer la contrainte FOREIGN KEY par une clause CHECK.

```
CREATE TABLE SALLE(
NUMCINEMA INTEGER NOT NULLL,
NUMSALLE INTEGER,
CAPACITE INTEGER CHECK (CAPACITE <  300),
CLIMATISEE CHAR(1) CHECK (CLIMATISEE IN ('O','N')) DEFAULT 'N',
PRIMARY KEY (NUMCINEMA, NUMSALLE)
CHECK (NUMCINEMA IN (SELECT NUMCINEMA FROM CINEMA)))
```

Il s'agit d'une illustration simple d'une clause CHECK avec sous-requête, mais elle est incorrecte pour garantir l'intégrité référentielle. Pourquoi ? (penser aux événements qui déclenchent respectivement les contrôles des clauses CHECK et FOREIGN KEY).

Au lieu d'associer une contrainte à un attribut particulier, on peut la définir globalement. Dans ce cas la contrainte peut faire référence à n'importe quel attribut de la table et est testée tuple à tuple.

Exemple 3

Toute salle de plus de 300 places doit être climatisée :

```
CREATE TABLE SALLE(
NUMCINEMA INTEGER NOT NULLL,
NUMSALLE INTEGER,
CAPACITE INTEGER,
CLIMATISEE CHAR(1),
PRIMARY KEY (NUMCINEMA, NUMSALLE),
FOREIGN KEY NUMCINEMA REFERENCES CINEMA(NUMCINEMA),
CHECK (CAPACITE < 300 OR CLIMATISEE = 'O'))
```

L'utilisation des sous-requêtes n'est pas recommandée, à cause du problème souligné précédemment :

La contrainte peut être satisfaite au moment de l'insertion du tuple, et ne plus l'être après.

Exemple 4

La grande salle du Rex doit rester la plus grande !

```
CREATE TABLE SALLE(
NUMCINEMA INTEGER NOT NULLL,
NUMSALLE INTEGER,
CAPACITE INTEGER CHECK (CAPACITE <  300),
CLIMATISEE CHAR(1)
PRIMARY KEY (NUMCINEMA, NUMSALLE)
FOREIGN KEY NUMCINEMA REFERENCES CINEMA(NUMCINEMA),
CHECK (CAPACITE < (SELECT MAX (CAPACITE) FROM SALLE,CINEMA
                WHERE CINEMA.NUMCINEMA = SALLLE.NUMCINEMA
                AND CINEMA.NOMCINEMA ='Rex')))
```

Problème : si on diminue la taille de la salle du Rex, la contrainte peut ne plus être respectée. Il est donc préférable de ne pas utiliser la clause CHECK pour des contraintes impliquant d'autres tables.

Il est possible, et recommandé, de donner un nom aux contraintes avec la clause CONSTRAINT.

```
CONSTRAINT CLIM CHECK (CAPACITE < 300 OR CLIMATISEE = 'O')
```

Cela facilite la compréhension des messages, et permet de modifier ou de détruire une contrainte:

```
ALTER TABLE SALLE DROP CONSTRAINT CLIM
```

V. Les Jointures et leurs types

1. Principe

La Jointure est l'opération qui permet d'intégrer plusieurs Tables de Données ensembles, afin d'effectuer des requêtes prenant en compte des informations réparties entre ces tables.

2. Exemple

Considérons les Tables de Données Suivantes:

Client:

Nom	Adresse	Telephone
Mohamed	1, rue du bouleau	0123456789
Brahim	2, rue des écureuils	0132547698
Fatma	3, rue du chêne	0198765432
Khadija	4, rue des pinsons	0189674523

Facture:

Nom	Montant
Mohamed	25 000
Brahim	32 000
Khadija	18 000

3. Types de Jointures

Plusieurs types de jointures existent:

- Cross Jointure (= produit cartésien)
- Jointure Interne (= Inner Join, Equi-join, Natural Join)
- Jointure Externe (= Outer Join)
- Auto Jointure (= Self Join)

4. Définitions et exemples

Nous allons maintenant étudier les propriétés de chaque type de jointure.

Cross Jointure

Définition

Une Cross-Jointure (Jointure croisée) consiste en le produit cartésien des deux tables sur lesquelles la jointure est effectuée.

Syntaxe

```
SELECT Nom, Adresse, Montant

FROM Client CROSS JOIN Facture;
```

Jointure Interne

Définition

Une Jointure Interne permet d'intégrer plusieurs Tables de Données en mettant en correspondance les occurrences de plusieurs tables qui ont un attribut commun.

Syntaxe 1

```
SELECT Nom, Adresse, Montant
FROM Client NATURAL JOIN Facture;
```

Problème:

- Si des attributs de même nom et de signification différente apparaissent dans les tables jointes, ils seront utilisés pour réaliser la jointure naturelle. D'où un fort risque d'erreur non maîtrisées !
- Si un attribut possède deux noms différent dans les deux tables, la jointure ne peut pas avoir lieu

Solution:

Spécifier le nom du (des) attribut(s) qui servent de référence pour la jointure (INNER JOIN ... USING ou INNER JOIN ON)

Syntaxe 2

```
SELECT Nom, Adresse, Montant

FROM Client INNER JOIN Facture USING(Nom);

  Ou

SELECT Nom, Adresse, Montant
FROM Client INNER JOIN Facture

ON Client.Nom=Facture.Nom;
```

Jointure Externe

Définition

Problème:

Lors d'une jointure interne, seules les occurrences pour lesquelles des informations existent dans les 2 tables sont conservées.

Par exemple, lors d'une jointure entre la Table de Données 'Clients' et 'Commande', les clients pour lesquels aucune facture n'est en cours n'apparaitront.

Solution:

La jointure externe conserve l'ensemble des occurrences.

Exemple

```
SELECT Nom, Adresse, Montant
FROM Client C LEFT OUTER JOIN Facture F
ON C.Nom= F.Nom;
SELECT Nom, Adresse, Montant
FROM Client C RIGHT OUTER JOIN Facture F
ON C.nom= F.Nom;
```

- **LEFT OUTER JOIN:** prend comme référence la première table de 'FROM'
- **RIGHT OUTER JOIN:** prend comme référence la dernière table de 'FROM'
- **RIGHT FULL JOIN:** prend comme référence toutes les tables de 'FROM'

5. Performance

Problème

La jointure est l'opération la plus coûteuse sur une Base de Données en termes de performance.

Solution

Il est donc indispensable de réaliser les opérations de sélection/projection autant que possible avant la jointure.

Exemple

Recherche du téléphone d'une personne donnée à partir de deux tables, dans un annuaire qui contient 100 000 entrées.