Plan du cours

- Généralités
  - Les principaux modèles
  - Le modèle relationnel
- L'utilisation des SGBD
  - Les langages
  - L'algèbre relationnelle
  - Le langage SQL

    Création de tables et ajout de données

    Interrogation de la base

    Vues et index
  - Les déclencheurs (triggers)

Définitions

Banque de données

Ensemble de données relatif à un domaine défini de connaissances, organisé pour être offert en consultation aux utilisateurs

≠

Base de données

Ensemble structuré de données enregistrées sur des supports accessibles par ordinateur pour satisfaire un ou plusieurs utilisateurs de façon sélective et en un temps opportun

Système de Gestion de Bases de Données

Logiciel gérant toutes les opérations sur les données de la base (création, interrogation, insertion, mises à jour, protection, sécurité, ...)

Approche traditionnelle

- On est passé directement des fichiers non informatisés aux fichiers informatisés
- Les données sont stockées dans un fichier maître
- Les différents utilisateurs accèdent aux données via des applications spécifiques indépendantes
- Les applications gèrent des fichiers dérivés
- Les données sont éventuellement dupliquées dans plusieurs fichiers dérivés

Problèmes de cette approche

- Complexité
- Redondance de l'information
- Coûts élevés de développement et de maintenance
- Manque de flexibilité
- Manque de sécurité

Historique

- Définition des modèles hiérarchiques et réseau au milieu des années 60
- IDS (premier système CODASYL) dès 1964
- Socrate (CII 1973) et Total (Cincom 1978) : modèle en réseau
- IMS (IBM) et System 2000 (MRI) construits sur le modèle hiérarchique (78)
- Depuis 1980, prépondérance des systèmes basés sur le modèle relationnel
- Origine : l'article E.F.CODD en 1970
- Premiers prototypes
- System-R (IBM San José en 75)
- Ingres (Stonebraker, Berkeley, 76)
- Premières versions commerciales
  - Oracle (76)
  - DB2 (IBM 82)
- Première norme SQL 1986

Les raisons de l'essor du domaine

- Vers 1960
- Émergence de gros volumes de données
- Évolution de la technologie liée aux supports de stockage
- Intégration de fichiers et d'applications gérés séparément
- Nécessité de représenter des liens de dépendance entre les enregistrements

L'approche bases de données

- Les données sont gérées dans une seule base
- Les bases de données sont gérées par un seul SGBD
- Les utilisateurs accèdent aux informations via le SGBD grâce à :
  - des langages d'interrogation
  - des programmes d'application
  - des outils divers

Les avantages des bases de données

- Diminution de la redondance
- Meilleure indépendance des données et des programmes d'application
- Accès aux données facilité
- Meilleure gestion de l'information

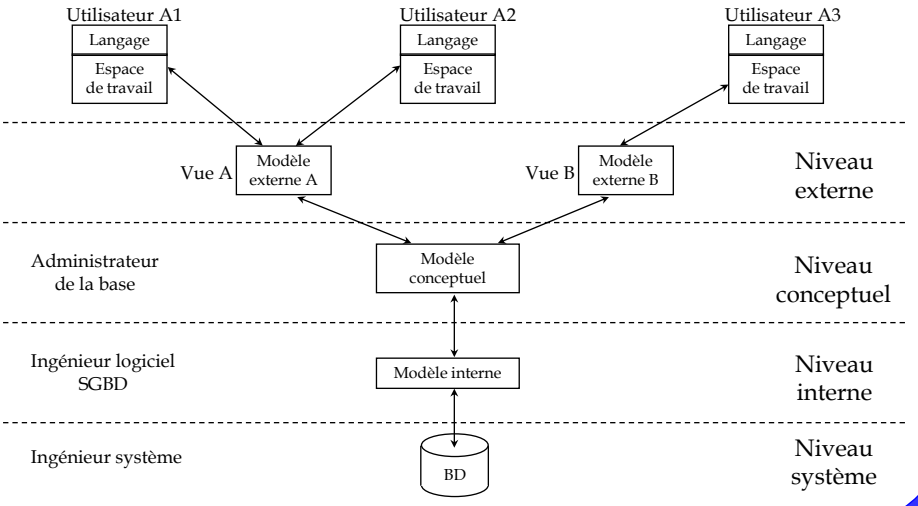
Fonctionnalités d'un SGBD

- Description des données
- Intégrité de la base
- Indépendance des données
- Souplesse d'accès aux données
- Partage des données
- Sécurité de fonctionnement
- Administration et contrôle

Structure fonctionnelle d'un SGBD

- Couche 1 : système de gestion de fichiers
  Gestion des récipients de données sur mémoire secondaire
- Couche 2 : SGBD interne
  Gestion des données stockées dans les fichiers
  - placement et assemblage
  - gestion des liens entre données et structure
  - recherche rapide (index)
- Couche 3 : SGBD externe
  présentation des données
  - aux programmes d'application
  - aux usagers ayant formulé leurs besoins en langage plus ou moins élaboré (requêtes, rapports, ...)

Architecture d'une base de données



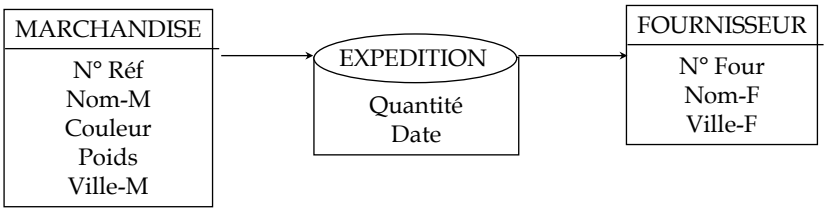
Les principaux modèles

- Modèle = tentative de représenter une certaine réalité
- Les modèles d'accès
  - le modèle hiérarchique
  - le modèle réseau
  - le modèle relationnel
- Définition d'un modèle de données :
  - une collection de structures de données
  - un ensemble d'opérateurs permettant de retrouver ou déduire des données
  - une collection de règles d'intégrité définissant l'ensemble des changements d'états de la BD

Exemple

Gestion d'expéditions de marchandises, dans le cas d'une entreprise nationale, possédant des usines dans des villes différentes, et passant des commandes de produits à divers fournisseurs

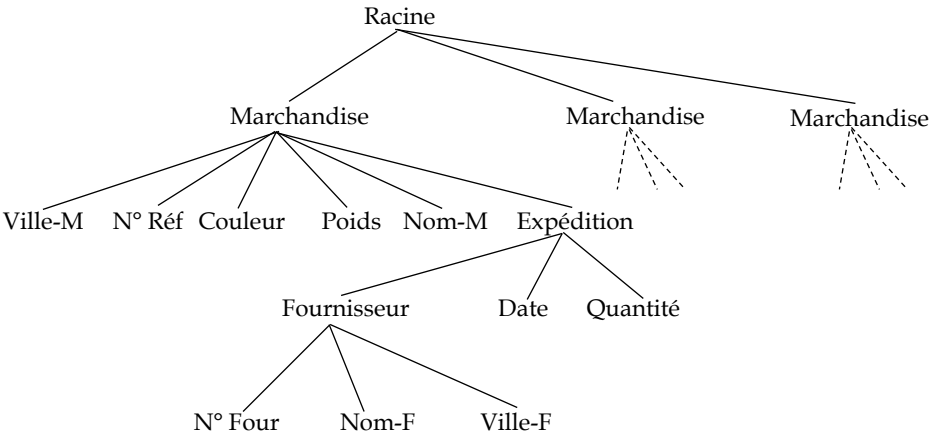
Deux entités : marchandise et fournisseur



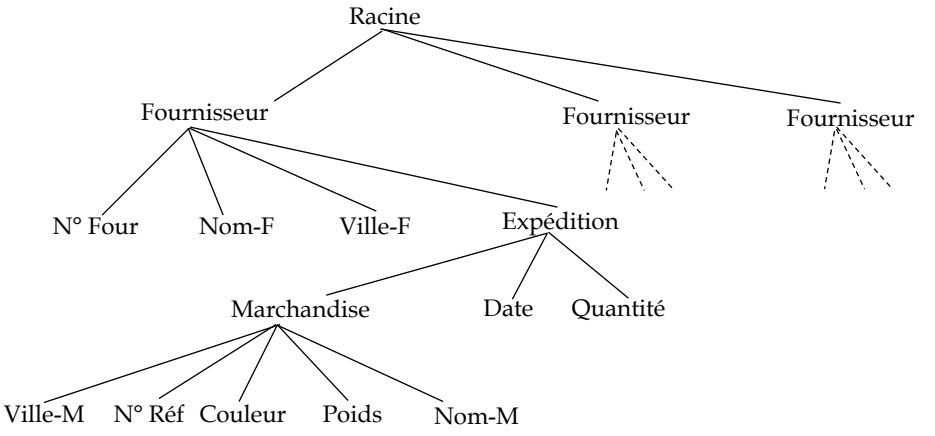
Le modèle hiérarchique

- Correspond plutôt à une approche système : comment sont stockées les données sur disque
- Représentent la majeure partie des systèmes commercialisés jusqu'en 80
- Le premier : IMS d'IBM (programme APOLLO) en 65
- Correspond à une structure très simple : arborescente
- Modèle basé sur la notion de hiérarchie dans laquelle tous les liens pointent dans la direction père - fils
  - L'entité = segment
  - Les liens = pointeurs

Représentation 1



Représentation 2



Représentation en fichiers

M1	Ecrou	Rouge	12	Lyon

F1	Durand	Paris	300	5
F2	Dupond	Marseille	300	4

M2	Boulon	Vert	17	Paris

F1	Durand	Paris	200	4
F2	Dupond	Marseille	400	1
F3	Vidal	Lyon	200	2

M3	Vis	Bleu	17	Toulouse

F1	Durand	Paris	400	3
F2	Dupond	Marseille	300	4

M4	Vis	Rouge	14	Lyon

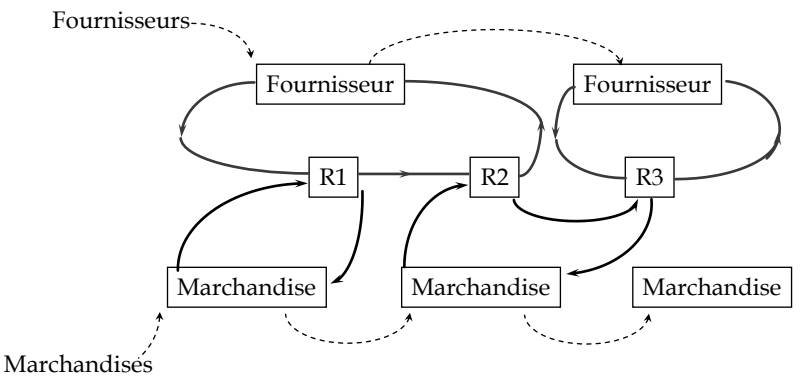
Analyse critique

- Deux défauts majeurs :
  - redondance => perte de place mémoire et risques d'incohérence en cas de modification d'information => opération coûteuse en temps d'exécution
  - dissymétrie : plusieurs implémentations possibles => favorise l'accès à certaines données
- Avantage :
  - représentation directe des liens et bonne rapidité lors de la recherche de données (dépendante de la hiérarchie)
- Remarques :
  - dépendance très forte entre le niveau conceptuel et le niveau physique
  - modèle figé
  - bonne optimisation des parcours nécessaire
  - temps de réponse dégradé lors de création car mise à jour de nbx pointeurs

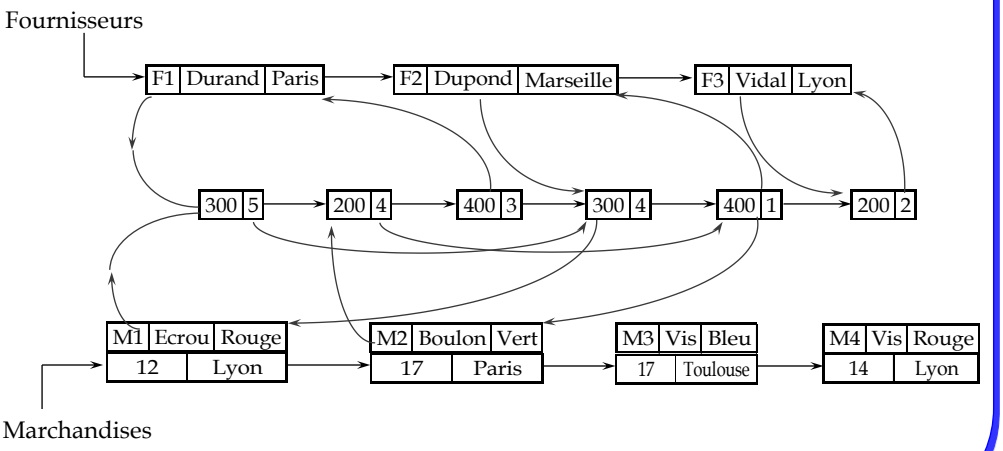
Le modèle réseau

- Proposé initialement par le groupe DBTG du comité américain CODASYL
- Les premières recommandations en 69 :
  - LDD
  - LMD : enrichissement du langage COBOL, navigationnel (travaille occurrence d'entité par occurrence d'entité pour se positionner)
- Le schéma est représenté sous forme d'un graphe, connectant les entités entre elles à l'aide de pointeurs
- Les liens N:1 et 1:N sont représentés directement tandis que les liens N:M sont transformés en deux relations N:1

Représentation simple



Représentation en fichiers



Analyse critique

- Ni redondance, ni symétrie
- La recherche peut être longue et dépend de la structure de la base
- Pas d'indépendance entre les niveaux logiques et physiques
- LMD navigationnel => nécessite du programmeur une bonne connaissance de la structure

Le modèle relationnel

- En 69 article de CODD (mathématicien chez IBM)
- Caractéristiques :
  - description simple des entités,
  - mise à jour sans anomalies de stockage,
  - manipulation non procédurale des données
- Ce modèle est le résultat d'une approche formelle mathématique qui regroupe dans une même théorie (l'algèbre relationnelle) la notion de données et de manipulation de données et ne fait aucune référence au niveau physique.
- Au modèle relationnel est associée la théorie de la normalisation qui permet d'éviter des incohérences dans la base lors des mises à jour.
- Principales références :

  Oracle 1979, Ingres 1980, Adabas 1972, Sybase, Informix, SQLBase

Les trois facettes du modèle

- Les définitions du modèle relationnel reposent sur une double base formelle :
  - la théorie des ensembles
  - la théorie des prédicats
- Structure

  Permet de décrire comment les données du schéma conceptuel sont structurées
- Manipulation

  Contient des opérateurs qui permettent de manipuler les données définies par les structures
- Contraintes d'intégrité

  Règles qui régissent la structure et qui traduisent des contraintes liées à la définition des données

Le vocabulaire

Relation MARCHANDISE

N° REF	Nom-M	Couleur	Poids	Ville-M
M1	Ecrou	Rouge	12	Paris
M2	Boulon	Vert	17	Brest
M3	Vis	Bleu	17	Lyon
M4	Vis	Rouge	14	Paris
M5	Ecrou	Vert	13	Marseille

Relation, table

Attribut, champ, rubrique

Tuple, enregistrement, fiche, ligne

Clé

Analyse critique

- Le schéma de données est simple et facile à manipuler (tables versus graphe)
- Indépendance physique : lors de la définition du schéma, aucune spécification physique n'est à préciser
- On dispose d'un LDD et d'un LMD de haut niveau

Les langages

- SQL (Strutured Query Langage)
- QBE (Query By Example)
- Autres langages ou éditeurs de recherche

Langage de description de données

- Utilisé pour spécifier le schéma conceptuel de la base. Lié au modèle de données utilisé par le SGBD
- Ne devrait pas contenir d'informations sur l'organisation physique des données ni sur les supports de stockage (≠ hiérarchique et réseau)
- Sert aussi à la spécification des vues (niveau externe)
- Grandes différences entre les LDD des systèmes existants => portage des applications coûteux en temps

Langage de description physique

- Permet de spécifier certains aspects de l'organisation physique des données
- Cette organisation consiste en une description des supports de stockage, du placement des données, des méthodes d'accès et de l'organisation des structures d'accès (hachage, B-arbres, ...)
- Ces choix sont destinés à l'administrateur afin d'améliorer les performances du système
- L'organisation physique est proposée mais elle n'est pas prise en charge par le SGBD

Langage de manipulation de données

- C'est un langage qui permet la recherche (consultation et extraction sous conditions) ou la mise à jour (écriture) des informations de la base
- Le LMD dépend aussi fortement du modèle utilisé
- Le mode d'utilisation diffère selon les modèles :
  - les commandes sont encapsulées dans des langages hôtes L3G pour les modèles hiérarchique et réseau => manipulation procédurale des données
  - dans le modèle relationnel, le LMD est déclaratif car fondé sur un langage logique. SQL est le langage le plus utilisé.

Les programmes d'application

- L'écriture d'une application se fait à partir du niveau externe => connaissance du modèle conceptuel
- Deux approches principales :
  - utilisation d'un L3G et communication au travers d'une API
    - les données sont transférées dans une zone de mémoire tampon
    - utilisation d'un protocole de communication entre deux systèmes autonomes
    - respect du principe d'indépendance des données
  - utilisation d'outils greffés sur le SGBD : les L4G

Les langages de 4ème génération

- Pas de définition précise ni de normalisation
- Véritable langage de programmation :
  variables, structures de contrôle, appel de fonctions diverses, ...
- Contient en général :
  - un générateur d'applications : enchaînement de menus et programmation de touches de fonctions
  - un générateur de rapports (mailing)
  - un générateur de masques d'écran
  - diverses API
- Conviennent en général à la création de petites applications personnalisées

Le langage SQL

- Issu des travaux réalisés autour des prototypes de systèmes relationnels
  - SEQUEL : System-R
  - QUEL : Ingres
- Première version : fin 70 - Oracle
- Propose des primitives couvrant la totalité des fonctionnalités que doit assurer un SGBD
- Normalisation :
  - 1ère norme : 86 : définit les mots du langage mais laisse l'implémentation aux constructeurs
  - 2ème norme : 89 : améliore la prise en compte des différents types d'intégrité
  - 3ème norme en 1999 : déclencheurs, nouveaux types de données

Langage de Définition de Données

- C'est la partie du SQL qui permet de créer des objets SQL
- Plusieurs catégories d'objets :
  - bases de données, tables, index, contraintes, synonymes, séquences, vues, ...
- Commandes de base :
  - CREATE, ALTER, DROP , RENAME, TRUNCATE, ...

  qui permettent de créer, modifier, supprimer un élément de la base

Dénomination des objets

- Ne pas choisir un mot-clé !
- Les mots-clés et les identifiants peuvent être mentionnés indifféremment en majuscules ou minuscules
- Commencent par une lettre ou un trait de soulignement
- Caractères suivants : combinaisons de caractères de soulignement, des lettres et des chiffres.
- Les caractères spéciaux sont exclus : ! @, #, $, %, ^, &, ou *
- Convention : si le nom est une combinaison de mots, chaque mot commencera en majuscules

Les types de données

- Pourquoi ?
  - validation lors de la saisie
  - optimisation du dimensionnement des tables
- Types alphanumériques
- Types numériques
- Types temporels
- Types " BLOBS "

Types de données textuels

- CHAR [(taille_maximale)]
  - de longueur fixe
  - si une taille maximale n'est pas précisée, par défaut 1
- NCHAR
  - contrepartie de CHAR pour l'encodage Unicode
- VARCHAR2(taille_maximale)
  - chaîne de caractères de longueur variable
  - la taille doit obligatoire être déclarée
- NVARCHAR2
  - contrepartie de VARCHAR2 pour l'encodage Unicode

Types numériques

- DECIMAL/ NUMERIC
  - commun pour tout usage
  - syntaxe : DECIMAL[([précision,] [échelle])]
- Précision
  - nombre total de chiffres (à la fois à gauche et à droite de la virgule), maximum = 38
- Entier : DECIMAL(précision)
- Nombre à virgule fixe
  - le nombre de chiffres après la virgule (échelle) est fixe
  - on précise à la fois la précision et l'échelle
- Nombre à virgule flottante
  - le nombre de chiffres après la virgule est variable
  - on ne précise rien

Types temporels

- On peut stocker la date et l'heure courante
  - DATE
  - TIMESTAMP
- DATE
  - du 1er janvier 4712 avant JC au 31 décembre 9999 après JC
  - format de date par défaut
    - DD-MON-YY
  - format d'heure par défaut
    - HH:MI:SS AM
- TIMESTAMP(précision_en_fraction_de_secondes)
  - la date et l'heure sont enregistrées avant une information supplémentaire : le nombre de secondes
  - il faut préciser le nombre de secondes à conserver

Types Large Object (LOB)

- On peut y stocker des sons ou des images numérisées
  - on peut aussi faire référence à des fichiers sur disque contenant les données

Large Object (LOB) Data Type	Description
BLOB	Binary LOB, storing up to 4 GB of binary data in the database
BFILE	Binary file, storing a reference to a binary file located outside the database in a file maintained by the operating system
CLOB	Character LOB, storing up to 4 GB of character data in the database
NCLOB	Character LOB that supports 2-byte character codes, stored in the database—up to a maximum of 4 GB

Définition de table

```
CREATE TABLE Etudiant   (
    Ide INTEGER,
    Nom CHAR(25),
    Sexe CHAR(1),
    Date_n DATE,
    Annee DECIMAL          )   ;

CREATE DOMAIN Numet INTEGER;
```

Termine toute requête !

Contraintes

- Plusieurs types de contraintes
  - contraintes d'intégrité
    - clé primaire
    - clé étrangère
  - contraintes sur les valeurs
    - non nullité
    - unicité
- Deux façons de définir une contrainte
  - au niveau attribut, elle ne s'applique qu'à un champ
  - au niveau table, elle s'applique à plusieurs attributs
    - on utilise souvent cette possibilité pour la définition d'une clé primaire multi-attribut
  - on peut donner un nom à la contrainte

Contrainte de non-nullité

- La valeur de l'attribut doit obligatoirement être renseignée
  - on ne peut pas définir cette contrainte au niveau table
- Syntaxes
  - nom varchar(30) not null contrainte anonyme
  - nom varchar (30) constraint etudiant_name_nn not null
  - les clés primaires ne peuvent pas être NULL (validation par défaut)
- Signification de la valeur NULL
  - elle n'est pas connue dans la table
  - elle n'existe pas dans la réalité
  - ne pas tricher en définissant des valeurs par défaut 0 ou ''
  - ne pas utiliser NOT NULL systématiquement car très contraignant

Contrainte d'unicité

- Une même valeur de l'attribut ne peut pas apparaître sur 2 enregistrements différents
  - utilisé lorsqu'il y a plus de 2 attributs candidats à devenir clé primaire
    - un seul devient clé primaire
    - les autres candidats doivent obligatoirement être contraints comme uniques
  - syntaxes
    - au niveau attribut
      - nom_department varchar(12) unique
    - au niveau table
      - constraint etudiant_nom_uk unique (nom, prenom)

Validité de valeur

- Pour vérifier des règles métier
- Contrainte générale que chaque enregistrement doit respecter
  - au niveau attribut
    - salaire decimal(5) check (salaire >= 0)
    - sexe char(1) check (sexe in ('M', 'F'))
  - au niveau table
    - check (sexe='M' or salaire< 1000)

Valeur par défaut

- Ce n'est pas une contrainte mais la syntaxe est la même
- Sera utilisée lorsqu'aucune valeur n'est fournie à l'insertion de l'enregistrement dans la table
  - pays char(2) default 'FR'
  - salaire decimal(5) default 0

Nommer les contraintes

- On donnera un nom à une contrainte pour pouvoir par la suite
  - modifier la contrainte
  - supprimer la contrainte
- Convention de nommage
  - nomdetable_nomdattribut_typedeconstrainte
  - PK, FK, UQ, NN, CK, RF
- Si vous ne les nommez pas, généralement le SGBD le fait pour vous, mais c'est plus difficile de les identifier
  - Il faudra interroger les tables du dictionnaire de données

Définition de l'intégrité

```
CREATE TABLE Etudiant          (
    Id_et DECIMAL(3) PRIMARY KEY,
    Nom CHAR(25) NOT NULL,
    Sexe CHAR
        CHECK (Sexe IN ('M','F') OR Sexe IS NULL),
    Date_naissance DATE,
    Annee DECIMAL(1) DEFAULT 3
    CHECK (Annee < 4) ) ;
```

Intégrité de référence

- Lorsque les attributs d'une table font références à des attributs dans une autre table
  - le SGBD vérifie si l'attribut référencé existe

```
CREATE TABLE ConventionStage    (

    Ide DECIMAL(3),

    Ids DECIMAL(4),

    FOREIGN KEY(Ide) REFERENCES Etudiant,

    FOREIGN KEY(Ids) REFERENCES Societe(Num_Soc),

    PRIMARY KEY (Ide, Ids)          ) ;
```


Maintenir l'intégrité de référence

- Que se passe-t-il si l'enregistrement référencé est modifié ou supprimé ?
- On utilise un déclencheur (trigger)
- On précise l'action au moment de la définition de l'intégrité de référence
- Plusieurs possibilités
  - On Delete Cascade
  - On Update (Delete) No Action
  - On Update (Delete) Set Null

Modification du schéma relationnel

```
ALTER TABLE Etudiant
        ALTER Nom CHAR(50) ;

ALTER TABLE Etudiant
        ADD Prenom CHAR(15);

ALTER TABLE Etudiant
        DROP Prenom;

DROP TABLE Etudiant;
```


Modifier les attributs

- Ajout
  ALTER TABLE employes ADD (job_id number);
- Modification de définition
  ALTER TABLE employes MODIFY (comm number(4,2) default 0.05);
- Suppression
  ALTER TABLE employes DROP COLUMN comm;
- Renommer
  ALTER TABLE employes RENAME COLUMN hiredate to dateembauche;
- Marquer l'attribut comme non utilisé
  ALTER TABLE employes SET UNUSED COLUMN job_id;
- Marquer la table en lecture seule
  ALTER TABLE employes READ ONLY;

Modifier les contraintes

- Ajout à une table existante
  ALTER TABLE nomtable
  ADD CONSTRAINT nomconstrainte definitionconstrainte;
- Supprimer une table existante
  ALTER TABLE table_name
  DROP CONSTRAINT constraint_name;
- Désactiver une contrainte
  ALTER TABLE table_name
  DISABLE CONSTRAINT constraint_name;
- Activer une contrainte
  ALTER TABLE table_name
  ENABLE CONSTRAINT constraint_name;

Mise à jour de données

```
INSERT INTO Etudiant
    VALUES (123,'Dupond', NULL, '1961-2-28', 2);

DELETE FROM Etudiant
    WHERE Annee = 3;

UPDATE Etudiant
    SET Annee = 3
    WHERE Annee = 2;
```

Insertion de données

- Syntaxe de base : on précise la valeur de chaque attribut

  INSERT into table_name
  VALUES (column1_value, column2_value, …);

- Syntaxe alternative : on ne précise que certaines attributs sélectionnés

  INSERT into table_name (columnname1, columnname2, …)
  VALUES (column1_value, column2_value, …);

- Attention : une clé étrangère doit référencer une donnée déjà insérée dans une autre table

Duplication de table

CREATE TABLE dup_etud AS SELECT * FROM etudiants;

ne copie pas les contraintes d'intégrité

TRUNCATE TABLE dup_etud;

INSERT INTO dup_etud SELECT * FROM etudiants;

Insérer des dates

- On peut utiliser le format par défaut

  INSERT INTO table_name VALUES (99, '31-may-98');

- On peut utiliser la fonction TO_DATE pour convertir une chaîne de caractères en date

  TO_DATE('chainedate', 'modeledate')

  INSERT INTO table_name
  VALUES(99, TO_DATE('1998/05/31','YYYY/MM/DD'));

Mise à jour de données

- Syntaxe de la requête de mise à jour : UPDATE

  UPDATE table_name
  SET column1 = new_value1, column2 = new_value2, …
  WHERE condition;

- Syntaxe de la requête de suppression : DELETE

  DELETE FROM table_name WHERE condition;

  – on ne peut supprimer que dans une table à la fois

- Sans critère de filtre, tous les enregistrements sont mis à jour ou supprimés

Transactions

- Une transaction est un groupe atomique d'instructions
  - propriétés ACID (atomicité, cohérence, isolation et durabilité)
  - exemple : transfert d'argent entre comptes bancaires

```
update comptes
set solde = solde + retrait
where num_compte = 2012

update comptes
set solde = solde – retrait
where num_compte = 1202
```


Transaction Control Language

- C'est la partie du SQL chargée de contrôler la bonne exécution des transactions
- Elle permet de gérer les propriétés ACID des transactions
- Suivant les SGBD, les fonctionnements peuvent être différents
  - Oracle gère automatiquement des verrous en mode exclusif ou partagé suivant le type de requête
- On peut déférer la validation de contrainte au moment de la validation des requêtes : on précise ce fonctionnement lors de la définition de la contrainte
- On a parfois la possibilité de placer des points de retour
  - un SAVEPOINT est un marqueur dans une transaction qui permettra une annulation partielle des modifications

```
SAVEPOINT s1;
ROLLBACK TO s1;
```

Contrôle des transactions

- Transactions implicites
  - Create, Alter Table, Drop, Grant ...
- Transactions explicites
  - la mise à jour des données doit être validée pour être définitivement prise en compte
  - les ordres COMMIT et ROLLBACK permettent respectivement de valider ou d'annuler toutes les modifications intervenues sur la base de données depuis leur dernière évocation ou depuis le début de la session
  - COMMIT implicite avant et après toute requête DDL
  - ces ordres ne s'appliquent qu'au travail réalisé par l'utilisateur qui les invoque, et non pas à celui exécuté en parallèle par d'autres éventuels utilisateurs

Data Control Language

- C'est la partie du SQL qui s'occupe de gérer les droits d'accès aux tables
- Elle comporte les commandes de base suivantes :
  - GRANT, REVOKE

  qui permettent respectivement d'attribuer et de révoquer des droits
- On peut accorder 2 catégories de privilèges à un utilisateur : system ou objet
  - privilèges système : ce sont des privilèges généraux permettant de manipuler les objets
    - exemple : possibilité de créer ou modifier des tables, des indexes, des vues ...
  - privilèges d'objet : on donne des droits précis sur un objet identifié
    - exemple : possibilité de manipuler une table précise, ou d'exécuter une fonction identifiée

Administration

- L'ordre GRANT permet d'acquérir ou de donner à d'autres utilisateurs des droits sur la base
- Ce mécanisme est associé à la notion de propriété d'une relation : le propriétaire est celui qui l'a créée

```
GRANT SELECT, INSERT ON Etudiant TO sophie;
GRANT ALL ON Etudiant TO sophie
    WITH GRANT OPTION ;
GRANT EXECUTE ON procedure TO sophie;
```


Retirer un privilège

- On utilise la requête REVOKE

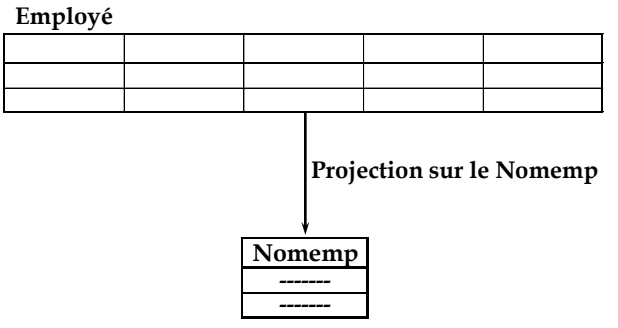
```
REVOKE privilege_name ON object_name
    FROM {user_name |PUBLIC |role_name}
```

L'algèbre relationnelle

- L'algèbre relationnelle est constituée d'un ensemble d'opérateurs qui permettent de spécifier les données qu'on veut extraire :
  - cinq sont dits primitifs, les autres peuvent en être dérivés
  - ils sont unaires ou binaires
- L'application d'un opérateur sur des relations génère une nouvelle relation
- Ce langage d'expression des requêtes nécessite une bonne connaissance du schéma conceptuel pour naviguer
- N'offre pas un très haut niveau de déclarativité car il faut indiquer l'ordre d'application des opérateurs

L'opération de projection

Consiste à enlever des champs à une relation



L'opération de sélection (restriction)

- Permet de caractériser un sous-ensemble de tuples dans une relation en utilisant un prédicat

Employé

Numemp	Nomemp	Préemp	Année	Codedép
1	Durand	Alain	1980	1
2	Dupond	Pierre	1983	3
3	Després	Marc	1990	3
4	Dupont	Jean	1983	2

Sélection des employés embauchés en 83

Numemp	Nomemp	Préemp	Année	Codedép
2	Dupond	Pierre	1983	3
4	Dupont	Jean	1983	2

Le produit cartésien

- Opérateur binaire
- Chaque enregistrement de la relation résultante est composé d'un enregistrement de la première et d'un de la seconde
- Peu utilisé explicitement car de faible intérêt

L'opération de jointure

- Permet de retrouver des informations organisées sur plusieurs tables à travers des relations dépendantes
- Si R et S sont deux relations ayant les attributs A1, ..., An en commun, alors la θ–jointure est la relation construite sur l'union des attributs de R et S contenant tous les tuples composés d'un tuple de R et d'un tuple de S qui coïncident au niveau de l'un des comparateurs usuels θ sur A1, ..., An
- Généralement on effectue des équi-jointures (l'opérateur de comparaison est =)

L'opération de division

- Permet généralement de retrouver les tuples d'une relation qui sont associés à tous les tuples d'une autre relation
- Si R et S sont deux relations construites sur les attributs A1, ... , An, B1, ..., Bm et B1, ..., Bm respectivement, alors la relation quotient est construite sur les attributs A1, ... , An et est composée des tuples *a1, ..., an* tels que, pour tout tuple *b1, ...,bn* de S, le tuple *a1, ..., an*, *b1, ...,bn* appartient à R

Les opérateurs ensemblistes

- Opérateurs binaires qui s'appliquent sur des relations unicompatibles, c'est-à-dire qui possèdent des domaines d'attributs qui sont les mêmes
- Union
- Intersection
- Différence

  Il s'agit des tuples de la première relation qui n'appartiennent pas à la seconde

Manipulation de données

Forme globale:

Dans cet ordre ↓

Select
From
Where
Group by
Having
Order by

Les requêtes de projection

On l'utilise pour obtenir tous les enregistrements d'une table

```
SELECT  Nom, Prenom  FROM Etudiant ;

SELECT  *  FROM Etudiant ;
```


Les requêtes de sélection

On utilise la clause WHERE pour préciser une condition que les enregistrements à afficher doivent satisfaire

```
SELECT     Nom, Prenom
FROM Etudiant
WHERE Annee = 2;

SELECT     *
FROM Etudiant
WHERE Annee = 3 AND Sexe = 'F' ;
```

Conditions

Ce sont des prédicats construits avec :

- des noms de champs
- des opérateurs de comparaison : =, <>, >,>=,<, <=
- des opérateurs booléens : and, or , not
- des constantes
  - les constantes alphabétiques sont entourées de simples quotes (apostrophes)
- d'autres opérateurs
  - IS NULL et IS NOT NULL
  - IN (liste de valeurs séparées par des virgules), NOT IN
  - BETWEEN value1 AND value2
  - LIKE chaîne_de_caractères
    - le symbole % remplace n'importe quelle chaîne, et _ remplace un caractère unique

Eliminer les résultats dupliqués

On utilise le mot clé distinct

SELECT DISTINCT salaire FROM employes;

Expressions

- Les chaînes de caractères sont entourées de quotes simples
- Valeurs numériques -1.26e+6
- Attributs de table : nom_table.nom_champ
- Les opérateurs de comparaison ne sont pas limités aux valeurs numériques et peuvent être utilisés avec des champs contenant du texte
- Opérateurs arithmétiques + - * / %
  - une expression est formée en combinant un nom de champ ou une constante avec un opérateur arithmétique

Fonctions agrégats

- Elles agrègent les résultats d'une requête plutôt que lister tous les enregistrements
  - SUM () calcule la somme des valeurs d'un champ sur tous les enregistrements correspondant éventuellement à une condition ; le champ doit correspondre à un type numérique
  - AVG calcule la moyenne des valeurs d'un champ
  - MAX () calcule la valeur maximale entre toutes les valeurs d'un champ
  - MIN () calcule la valeur minimale entre toutes les valeurs d'un champ
  - COUNT(*) calcule le nombre de valeurs d'un champ

```
select count(distinct group) as nombre_de_groupes from etudiant;
```

Fonctions mathématiques

- A utiliser dans la liste des champs d'un select ou dans la clause WHERE
- On peut utiliser les opérateurs arithmétiques (+, -, * and /) ou une des fonctions suivantes, appliquées sur des champs ou des constantes

ABS(X) : valeur absolue : convertit un nombre négatif en un positif
CEIL(X) : X est une valeur décimale qui sera arrondie à l'entier supérieur
FLOOR(X) : X est une valeur décimale qui sera arrondie à l'entier inférieur
GREATEST(X,Y) : retourne la plus grande des 2 valeurs
LEAST(X,Y) : retourne la plus petite des 2 valeurs
MOD(X,Y) : retourne le reste de la division
POWER(X,Y) : retourne X puissance Y
ROUND(X,Y) : arrondit X à Y décimales. Si Y est omis, X est arrondi à l'entier le plus proche
SIGN(X) : retourne un moins si X < 0, un plus sinon
SQRT(X) : retourne la racine carrée de X

Fonctions sur les chaînes de caractères

- lower(chaîne) : convertit la chaîne passée en paramètre en minuscules
- replace(chaîne,str1,str2) : remplace chaque occurrence de str1 dans chaîne char par str2
- substr(chaîne,m,n) : extrait n caractères de chaîne à partir de la position m
- length(chaîne) : retourne la longueur de la chaîne
- rpad(expr1,n,expr2) : complète à droite expr1 avec n fois expr2
  - souvent utilisé pour compléter par des espaces un champ de longueur fixe
- initcap(chaîne) : change le premier caractère de la chaîne en majuscule

Fonctions sur les dates

- On se souvient que SQL stocke des nombres correspondant à des dates à partir d'une date initiale (nombre 0)
- Quand on interroge un champ de type date, le nombre retourné est automatiquement mise en forme au format par défaut (DD-MON-YY)

  SELECT b FROM t; produit '01-APR-98'
- Pour afficher le nombre retourné à un autre format, on utilise la fonction TO_CHAR

  SELECT TO_CHAR(b, 'YYYY/MM/DD') FROM t; produit 1998/04/01
- De même, lorsqu'on veut insérer des valeurs de type date, il faut fournir une chaîne de caractères respectant le format par défaut (DD-MON-YY)

  INSERT INTO t VALUES ('01-APR-98');
- Si on souhaite insérer une valeur à un autre format, on utilise la fonction TO_DATE

  INSERT INTO t VALUES (TO_DATE('1998/04/01','yyyy/mm/dd');

Tri des résultats

- La clause ORDER BY suivi du nom d'un attribut trie les résultats d'une requête suivant les valeurs de cet attribut
- On utilise ASC (par défaut) ou DESC à la fin pour préciser l'ordre de tri

```
SELECT DISTINCT Nom
FROM Etudiant
WHERE Annee =2
ORDER BY Nom DESC;

SELECT Nom,Prenom
FROM Etudiant
WHERE Sexe = 'M'
ORDER BY Prenom DESC, Nom;
```

Opérations sur les ensembles

- On peut combiner les résultats de deux requêtes avec les mots clés **union** (pour des valeurs distinctes), **union all** (toutes les valeurs), **intersect** ou **minus** : les résultats sur lesquels on applique ces opérateurs doivent avoir le même nombre de champs et les mêmes types de données pour des champs de même position

requête1 **union** requête2

L'opération de jointure

- Permet de retrouver des informations organisées sur plusieurs tables à travers des relations dépendantes
- Si R et S sont deux relations sur de schémas de relation ayant les attributs A1, ..., An en commun, alors la θ–jointure est la relation construite sur l'union des attributs de R et S contenant tous les tuples composés d'un tuple de R et d'un tuple de S qui coïncident au niveau de l'un des comparateurs usuels θ sur A1, ..., An
- Généralement on effectue des équi-jointures (l'opérateur de comparaison est =)

Exemple

- Une table étudiant, une table société, une table convention
- Les étudiants sont identifiés par un numéro, clé primaire : ide
- Les sociétés sont identifiées par un numéro, clé primaire :ids

Jointure – Forme prédicative

- Dans la clause FROM on mentionne toutes les tables impliquées dans la jointure
- Dans la clause WHERE on mentionne une condition entre les champs comparables dans les 2 tables

```
SELECT nom, duree
FROM etudiant E, convention C
WHERE E.Ide = C.Ide AND Duree >=3 ;
```

- Quand les attributs dans les différentes tables ont les mêmes noms, on utilise une notation préfixée avec éventuellement un alias pour éviter toute ambigüité

Jointure sur plusieurs tables

- La jointure peut se faire sur un nombre variable de tables qui ont 2 à 2 des champs définis sur des domaines comparables
- Lorsqu'on joint n tables, on devra écrire n-1 conditions de jointure
  - sinon, on obtient généralement les résultats d'un produit cartésien

```
SELECT nom, raison_sociale
FROM etudiant E, convention C, societe S
WHERE E.ide = C.ide
    AND S.ids = C.ids;
```


Jointure – Forme ensembliste

- On peut imbriquer une requête dans une autre : elle sera écrite entre parenthèses
- En cas de jointure, la requête imbriquée apparaît dans la clause, dans une condition, avec n'importe lequel des opérateurs autorisés (IN, ALL, ANY)

```
SELECT nom, durée
FROM etudiant
WHERE ide IN (
        SELECT ide
        FROM convention
        WHERE duree >= 3 );
```

Opérateurs multilignes

- IN compare un élément à une valeur d'une liste résultat d'une sous-interrogation
- ANY compare un élément à chaque donnée d'une liste résultat d'une sous-interrogation
- ALL compare un élément à toutes les données d'une liste résultat d'une sous-interrogation

Jointure – Forme SQL2

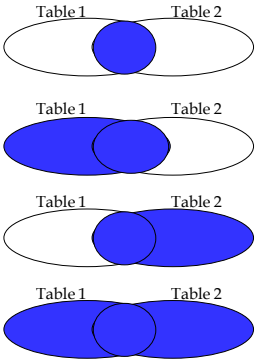
```
SELECT nom, duree
FROM etudiant E, convention C
WHERE E.ide = C.ide and E.ide=1;

SELECT nom, duree
FROM etudiant E INNER JOIN convention C
ON E.ide = C.ide
WHERE E.ide=1;

SELECT nom, duree
FROM etudiant NATURAL INNER JOIN convention
WHERE ide=1;
```

optionnel

pas de préfixe

Jointures externes

- Elles permettent d'extraire des enregistrements ne répondant pas aux critères de jointure interne
  - une ligne d'une table apparaîtra dans la table résultat de jointure même si elle n'a aucune correspondance dans la seconde table de jointure
  - l'ordre des tables devient important
    - Jointure interne
    - Jointure externe gauche
    - Jointure externe droite
    - Jointure externe totale



Ecriture d'une jointure externe

- On dit qu'une table est dominante et l'autre subordonnée
  - les enregistrements sans correspondance sont dans la table dominante
- Le sens de la directive de jointure LEFT ou RIGHT de la clause OUTER JOIN désigne la table dominante

  Select nom, duree from etudiant E left outer join convention C on E.ide=C.ide;

  Select nom, duree from convention C right outer join etudiant E on E.ide=C.ide;

  - le symbole + entre parenthèses (+) se place du côté de la table subordonnée : on peut le placer à gauche ou à droite de la condition de jointure

```
SELECT nom, duree
FROM  etudiant E, convention C
WHERE E.ide = C.ide (+);
```

Jointure externe bilatérale

- Les deux tables jouent une rôle symétrique, il n'y a pas de table dominante
- La directive FULL OUTER JOIN permet d'ignorer le sens de la jointure

```
SELECT nom, duree
FROM  etudiant E FULL OUTER JOIN convention C ON E.ide=C.ide;
```

- On peut faire l'union de 2 requêtes unilatérales si on veut utiliser le symbole (+)

```
SELECT nom, duree
FROM  etudiant E, convention C
WHERE E.ide = C.ide (+)
UNION
SELECT nom, duree
FROM  etudiant E, convention C
WHERE E.ide (+) = C.ide
```


Autojointure

- Lorsqu'il existe une relation entre champs d'une même table

```
SELECT e1.nom || ', ' || e1.prenom "Supérieur",
       e2.nom || ', ' || e2.prenom "Employé"
FROM employe e1, employe e2
WHERE e1. id_emp = e2.supe;
```

Sous-interrogation dans le FROM

- On peut construire dynamiquement une table dans la clause FROM d'une requête

```
SELECT  c.nb_co/e.nb_et
FROM(select count(ide) nb_co from convention) c,
        (select count(ide) nb_et from etudiant) e ;
```


Les branchements

- La clause CASE … WHEN permet d'effectuer une mise en forme de données en fonction de valeurs initiales

```
SELECT nom, prix_unitaire, quantite,
        CASE quantite
                WHEN 0 THEN 'Erreur'
                WHEN 1 THEN 'Offre de -5% pour le prochain achat'
                WHEN 2 THEN 'Offre de -6% pour le prochain achat'
                WHEN 3 THEN 'Offre de -8% pour le prochain achat'
                ELSE 'Offre de -10% pour le prochain achat'
        END
FROM  commande ;
```

Opérateurs de comparaison

- Une condition utilisant ANY est vraie pour les enregistrements dont la valeur d'un champ est comparé vraie à au moins une valeur de la liste retournée par la sous-requête

```
SELECT nom, date_achat
FROM achat
WHERE date_achat > ANY ( SELECT date_achat
                         FROM achat
                         WHERE nom = 'Vélo');
```

- Une condition utilisant ALL est vraie pour les enregistrements dont la valeur d'une champ est comparé vraie pour toutes les valeurs de la liste retournée par la sous-requête

```
SELECT nom, date_achat
FROM achat
WHERE date_achat > ALL ( SELECT date_achat
                         FROM achat
                         WHERE nom = 'Vélo');
```

- L'opérateur *IN* est équivalent à l'opérateur '= *ANY*'

Les sous-tables

- On les utilise pour grouper des résultats d'une requête dont un attribut a la même valeur
- Tous les attributs mentionnés dans la clause GROUP BY doivent apparaître dans la section SELECT et inversement
- On peut utiliser les fonctions agrégats pour un calcul sur un groupe d'enregistrements

```
SELECT annee, COUNT(distinct num_et)
FROM etudiant
GROUP BY annee;
```

optionnel

Sélection de sous-tables

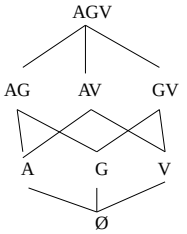
- On sélectionne les sous-tables qui correspondent à une condition en utilisant la clause HAVING
- On peut utiliser des requêtes imbriquées dans l'expression de la clause HAVING

```
SELECT annee
FROM etudiant
GROUP BY annee
HAVING COUNT(num_et) >10 ;
```

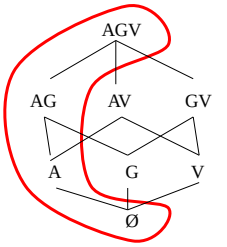

L'agrégation étendue

- L'opérateur group by construit une partition des résultats
- Les opérateurs associés au group by permettent de visualiser uniquement certaines parties de la partition

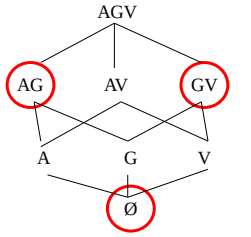
SELECT A, G, V, SUM(Q) FROM ...



GROUP BY CUBE(A, G, V)



GROUP BY ROLLUP(A, G, V)



GROUPING SETS((),(A, G),(G, V))

Exemples d'agrégations

```
select annee, groupe, ville_et, count(num_et) as nb_etud
from etudiant
group by

        grouping sets((annee, groupe),(annee, ville_et),());

        cube(annee, groupe, ville_et);

        rollup(annee, groupe, ville_et);
```


L'opération de division

- Permet généralement de retrouver les tuples d'une relation qui sont associés à tous les tuples d'une autre relation
- Si R et S sont deux relations construites sur les attributs A1, ... , An, B1, ..., Bm et B1, ..., Bm respectivement, alors la relation quotient est construite sur les attributs A1, ... , An et est composée des tuples *a1, ..., an* tels que, pour tout tuple *b1, ...,bn* de S, le tuple *a1, ..., an*, *b1, ...,bn* appartient à R

Les requêtes de division

- On les utilise pour obtenir les enregistrements qui sont en relation avec tous les enregistrements d'une autre requête/table
- On transforme «un enregistrement A est en relation avec tous les enregistrements» par «il n'existe pas un enregistrement qui n'est pas en relation avec l'enregistrement A»

Exemple : trouver une société

- qui a signé une convention avec tous les étudiants
- telle qu'il n'existe pas d'étudiant qui n'a pas signé de convention avec cette société

```
SELECT ids FROM societe S
WHERE NOT EXISTS (
    SELECT ide FROM etudiant E
    WHERE NOT EXISTS (
        SELECT * FROM convention C
        WHERE C. ids = S. ids and C. ide = E.ide )  );
```


Approche par cardinalités

- On calcule le nombre d'éléments dans chaque ensemble et on n'extrait que les éléments de même cardinalité

```
SELECT ids
FROM societe S, convention C
WHERE C.ids = S.ids
GROUP BY ids
HAVING count(disctinct ide) =
        ( SELECT ide FROM etudiant E  )
```

Sélection et division

Quelles sont les sociétés localisées à Marseille ayant signé une convention pour + de 2 mois avec tous les étudiants de 2nde année ?

```
SELECT ids FROM societe S
WHERE ville_soc='Marseille'
AND NOT EXISTS (
        SELECT Ide FROM Etudiant E
        WHERE annee=2
        AND NOT EXISTS (
                SELECT * FROM Convention C
                WHERE duree>2
                AND C.Ide = E.Ide AND C.Ids = S.Ids ) );
```


Fusion

- Pour combiner une séquence de requêtes INSERT, UPDATE, DELETE en une seule requête selon l'existence ou pas d'un enregistrement

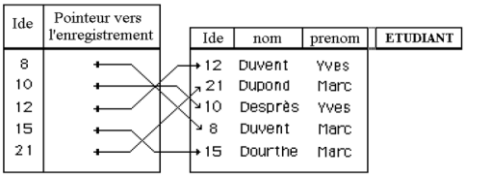
```
MERGE INTO depts destinat
USING (SELECT dno, dnom, loc FROM depts_enligne) origin
ON (destinat.dno = origin.dno)
WHEN MATCHED THEN
    UPDATE SET destinat.dnom = origin.dnom, destinat.loc = origin.loc
WHEN NOT MATCHED THEN
    INSERT (destinat.dno, destinat.dnom, destinat.loc)
            VALUES (origin .no, origin .dnom, origin .loc);
```

Organisation des données

- Les données dans une base sont mémorisées dans des fichiers dans l'ordre des insertions, et les accès réalisés via une lecture séquentielle de fichier (accès séquentiel)
- L'accès aux données dépend donc de l'organisation sur disque et de techniques permettant de les retrouver
- Lorsqu'on souhaite retrouver un sous-ensemble de données correspondant à un critère, on peut associer aux données des structures qui privilégieront les champs sur lesquels porte le critère par rapport aux autres

Index

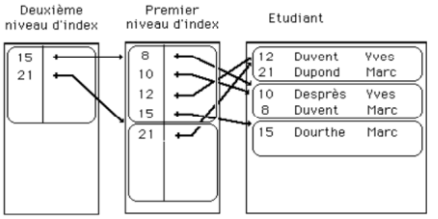
- Un index sur un ensemble de données est une structure à 2 champs
  - l'un représente des valeurs de l'ensemble choisies comme clé d'indexation
  - l'autre est un pointeur vers les enregistrements ayant même valeur d'indexation



- Les éléments dans un index sont triés
- Une recherche indexée est une recherche dichotomique sur la clé dans l'index

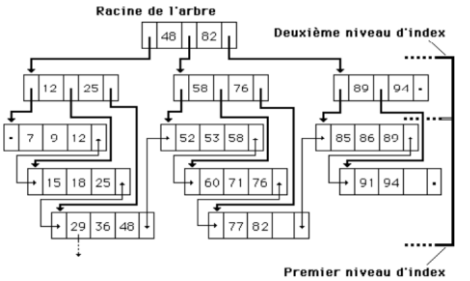
Index hiérarchiques

- Lorsque l'ensemble de données est volumineux, un second index peut être utilisé ; il est alors construit sur le premier, et les enregistrements sont regroupés par pages
  - la taille de l'index racine sera d'une page afin de limiter les accès
- Dans l'index de niveau supérieur, les valeurs stockées sont les dernières valeurs des pages de l'index de niveau inférieur



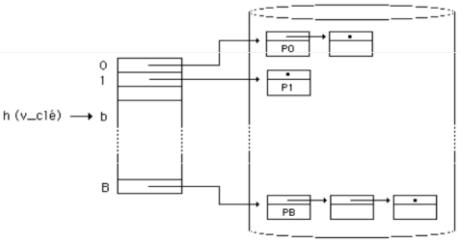
Les B-Trees

- Les arbres équilibrés (balanced-trees) sont des index hiérarchiques actualisés dynamiquement



Le hachage

- On peut aussi associer une valeur à des enregistrements
- On répartit alors par page des structures d'accès aux enregistrements correspondant à la même valeur
- Une fonction mathématique est appliquée aux valeurs de la clé, et calcule l'adresse de la première page des enregistrements correspondant à cette valeur



Les index

- Ce sont des structures optionnelles associées à des tables
- Il existe plusieurs types d'index : unique, non-unique, bitmap, basés sur des fonctions, pour renforcer les contraintes de clé primaire ...
- Sans index, le SGBD parcourt séquentiellement tous les enregistrements d'une table à la recherche de ceux qui correspondent à une condition
- Index de clé primaire
  - quand on définit une contrainte PRIMARY KEY, un index est généralement placé pour permettre de récupérer rapidement l'information

Création d'un index

- Forme générale
  CREATE INDEX index_name ON table_name (column1, column2…);

- Forme générale d'un index UNIQUE
  CREATE UNIQUE INDEX index_name
  ON table_name (column1, column2…);

- Un index est automatiquement mis à jour lors d'une insertion ou d'une suppression
  - donc ne pas les utiliser sur des champs qui sont modifiés souvent
  - on les utilise sur les clés étrangères pour optimiser les jointures

Suppression d'index

- On ne conserve un index que s'il améliore les temps d'exécution des requêtes

- Les index sur une table doivent être mis à jour si les données référencées par un index sont mises à jour

- Les index inutiles alourdissent le système en occupant une place mémoire disque supplémentaire

- La syntaxe de suppression d'un index
  DROP INDEX index_name;

Exemples d'index

CREATE INDEX I_Et ON Etudiant (Nom);

CREATE UNIQUE INDEX I_ET
ON Convention (Ide, Ids DESC);

DROP INDEX I_Et ;

ALTER INDEX I_ET ... ;

Les vues

- Une vue est une table "logique" basée sur une requête, on peut la voir comme une requête stockée et on peut l'interroger comme une table
- La requête de création de vue incorpore une requête SELECT

  CREATE VIEW parking_employe (emplacement, nom, prenom) AS
  SELECT place_parking _emp, nom_emp, prenom_emp
  FROM employe
  ORDER BY place_parking _emp;
- Les champs peuvent être renommés dans une vue
- Quand on interroge la vue
  - les seuls champs utilisables sont celles définis dans la vue
  - dans cet exemple, les résultats seront triés, même si il n'y a pas de clause ORDER BY dans la requête SELECT utilisée pour accéder à la vue

Contraintes dans une vue

- Syntaxe de création d'un vue

  CREATE VIEW <nom de vue> [(champ, ….)]
  AS <requête select>
  [WITH [CHECK OPTION] [READ ONLY]
  [CONSTRAINT nom_const]];

- WITH READ ONLY : création d'une vue en lecture seule
- WITH CHECK OPTION : permet la spécification de contraintes sur les valeurs
  - toutes les instructions de modification de données exécutées sur la vue respectent le critère défini dans la requête select

Fonctions et vues

- On peut utiliser des calculs ou des fonctions sur les enregistrements pour ajouter des champs
- L'utilisateur a accès aux résultats sans connaître les fonctions sous-jacentes

```
CREATE VIEW salaire_dept
        (nom, salaire_moyen) AS
SELECT d.nom_dept, AVG(e.salaire_emp)
FROM employe e, departement d
WHERE e.num_dept_=d.num_dept
GROUP BY d.nom_dept;
```

Stabilité d'une vue

- Une vue ne stocke pas réellement de données
- Les données sont obtenues à partir des tables sous-jacentes et affichées dans une table de résultats (stockée temporairement) lorsqu'une vue est interrogée
- Lorsqu'on supprime la table sous-jacente, la vue n'est plus valide

Contraintes d'utilisation d'une vue

- Une vue est un moyen de gérer l'accès aux données en simplifiant l'accès ou en sécurisant l'accès à l'information en limitant aux seules données qu'un utilisateur a besoin de connaître
- Une vue peut afficher des données provenant d'une ou plusieurs tables
- Les vues peuvent changer l'apparence des données en renommant les champs
- Les vues peuvent être utilisées pour modifier les données de table sous-jacentes, ou pour limiter en lecture à une table
- On peut insérer des données à travers une vue, celles-ci seront ajoutées à la table sous-jacente, à condition que les contraintes d'intégrité sur la table soient satisfaites
  - s'il existe une contrainte NOT NULL sur un champ qui n'a pas été sélectionné dans la vue, l'insertion à travers la vue ne sera pas possible

Suppression d'une vue

- Le propriétaire d'une vue peut la supprimer avec la commande DROP VIEW

```
DROP VIEW salaire_dept;
```


Séquences

- On les utilise pour générer une série de nombres uniques
- Ce sont des objets de la base de données au même titre que les tables, vues, index, ...
- On peut les utiliser pour générer les valeurs d'une clé primaire sur une table
- On peut générer les valeurs en ordre croissant ou décroissant, à partir d'une valeur initiale, en boucle, en incrémentant d'un nombre spécifié, ...

Création d'une séquence

CREATE SEQUENCE <nom de sequence >
[START WITH <valeur de départ >]
[INCREMENT BY <nombre>]
[MAXVALUE < valeur maximale >]
[NOMAXVALUE]
[MINVALUE < valeur minimale >]
[CYCLE]
[NOCYCLE]
[CACHE <nombre>]
[NOCACHE];

Exemple

```
CREATE SEQUENCE IncrPar10
    START WITH 50
    INCREMENT BY 10 ;

INSERT INTO etudiant (num_et) VALUES (NEXT VALUE FOR IncrPar10) ;

INSERT INTO etudiant (num_et) VALUES (NEXT VALUE FOR IncrPar10) ;

SELECT num_et FROM etudiant  WHERE num_et >= 50;
    NUM_ET
    ----------
        50
        60
```

Les déclencheurs (triggers)

- Procédures stockées attachées à une table (max 12)
- Déclenchées automatiquement quand un changement INSERT, UPDATE, DELETE intervient sur la table
- Trois parties : un événement, une contrainte de trigger (optionnelle), et une action
  - lorsqu'un événement est produit, le trigger est déclenché, et un bloc d'instructions prédéfini est exécuté pour réaliser l'action souhaitée
- Lorsqu'on supprime une table (drop), tous les triggers associés sont aussi supprimés

Créer un trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER} triggering_event ON table_name
[FOR EACH ROW]
[WHEN condition]
DECLARE
    Déclarations
BEGIN
    Instructions exécutables
EXCEPTION
     Gestionnaires d'exceptions
END;
```

- BEFORE et AFTER précisent quand le trigger est déclenché (avant ou après l'événement déclenchant)
- L'événement déclenchant fait référence à une requête LMD sur la table associée au trigger
- La clause FOR EACH ROW mentionne un trigger de ligne qui est déclenché pour chaque ligne modifiée
- La clause WHEN précise une condition qui doit être vérifiée pour que le trigger soit déclenché

Déclenchement d'un trigger

- Déclenché uniquement si la mise à jour est réalisée avec succès
  - en cas de violation de contrainte d'intégrité, le trigger n'est pas déclenché
- Déclenché quelque soit le nombre d'enregistrements concerné (même 0)
- Row trigger : déclenché autant de fois qu'il y a d'enregistrements concernés par l'événement
- Statement trigger : déclenché une seule fois même si la mise à jour affecte plusieurs enregistrements

Statement trigger

```
CREATE OR REPLACE TRIGGER first_trigger
BEFORE DELETE OR INSERT OR UPDATE ON employees
BEGIN
   IF (TO_CHAR(SYSDATE, 'day') IN ('sat', 'sun'))
      OR (TO_CHAR(SYSDATE, 'hh:mi') NOT BETWEEN '08:30' AND '18:30')
   THEN
         RAISE_APPLICATION_ERROR(-20500, 'table is secured');
   END IF;
END;
```

Row trigger

```
CREATE OR REPLACE TRIGGER second_trigger
    AFTER DELETE OR INSERT OR UPDATE ON employees
    FOR EACH ROW
BEGIN
    IF DELETING THEN
        INSERT INTO xemployees(emp_ssn, emp_last_name, emp_first_name, deldate)
        VALUES (:old.emp_ssn, :old.emp_last_name, :old.emp_first_name, sysdate);
    ELSIF INSERTING THEN
        INSERT INTO nemployees(emp_ssn, emp_last_name, emp_first_name, adddate)
        VALUES (:new.emp_ssn, :new.emp_last_name, :new.emp_first_name, sysdate);
    ELSIF UPDATING('emp_salary') THEN
        INSERT INTO cemployees(emp_ssn, oldsalary, newsalary, up_date)
        VALUES (:old.emp_ssn, :old.emp_salary, :new.emp_salary, sysdate);
    ELSE
        INSERT INTO uemployees(emp_ssn, emp_address, up_date)
        VALUES (:old.emp_ssn, :new.emp_address, sysdate);
    END IF;
END;
```


Un peu de programmation

- Déclaration de variables : uneVar sonType;
  exemples : an NUMBER(1,0); vr_et etudiant%ROWTYPE;
- Instructions
  - affectation : uneVar:=2;
  - if ... then ... [else ...] end if;
  - while ... loop ... end loop;
  - ...

  ex : if an=1 then DBMS_OUTPUT.PUT_LINE('Premiere annee'); end if;
- Utilisation des résultats d'un select :
  - si la requête ne renvoie qu'une ligne
    select annee into an from etudiant where num_et=1;
    select * into vr_et from etudiant where num_et=1;
  - si la requête renvoie plusieurs lignes : utilisation d'un curseur

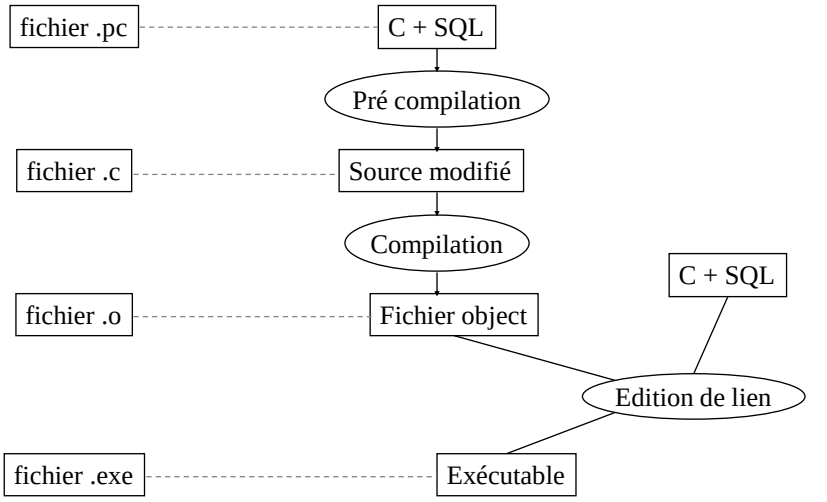
Ordre d'exécution

- Il peut exister plusieurs triggers sur une même table , y compris sur une même action (INSERT, UPDATE, DELETE)
  - il vaut mieux écrire des triggers simples et plusieurs triggers différents sur une même action
- Oracle déclenche dans un ordre aléatoire (non spécifié) les triggers de même type sur une même requête
- D'une façon générale :
  - BEFORE statement triggers
  - Boucle pour chaque enregistrement concerné par la requête SQL
    - BEFORE row triggers
    - verrouillage et modification de l'enregistrement
      - le verrou est conservé jusqu'à la fin de la transaction
    - AFTER row triggers
  - Vérification des contraintes d'intégrité
  - AFTER statement triggers

Gestion des triggers

- ALTER TRIGGER trigger_name DISABLE;
- ALTER TABLE table_name DISABLE ALL TRIGGERS;
- ALTER TABLE table_name ENABLE trigger_name;
- ALTER TABLE table_name ENABLE ALL TRIGGERS;
- DROP TRIGGER trigger_name;

Processus de génération de code



Eléments de base

- Les requêtes SQL sont insérées dans le fichier source de l'application
- Les requêtes SQL sont toujours précédées de EXEC SQL
- Toute requête interactive peut être utilisée dans une application
- Une requête SQL peut être insérée partout où une instruction C est autorisée
- A la différence du SQL interactif, on peut définir une section de déclarations

  BEGIN DECLARE SECTION ...
  END DECLARE SECTION
  DECLARE CURSOR ...
- Les requêtes SQL peuvent référencées les variables du C, et inversement ; une variable C peut être utilisée à tout endroit où SQL attend une constante
- SQL retourne une information complète sur la dernière requête exécutée
- On utilisera un curseur en C pour gérer chaque enregistrement d'une requête SQL qui retourne un ensemble d'enregistrements

Connexion

- Avant d'accéder aux données il faut se connecter au SGBD

```
EXEC SQL BEGIN DECLARE SECTION ;
        varchar user[10], password[15];
EXEC SQL END DECLARE SECTION ;
EXEC SQL CONNECT :user IDENTIFIED BY :password
        //; using :host_string;
EXEC SQL DISCONNECT ;
```


La section de déclarations

- Doit contenir une déclaration par variable C utilisée dans une requête SQL
- Dans la requête SQL, une variable C est précédée par :

  SELECT ... INTO *:uneVarC* FROM table WHERE ...= *:uneAutreVarC*
- Les types des variables et des attributs doivent correspondre

```
EXEC SQL BEGIN DECLARE SECTION;
        typedef struct {
                int num;
                varchar nom[31];
                varchar adres[31];
                int cp;
                char tel[13];
        } type_sClient;
        type_sClient unClient ;
EXEC SQL END DECLARE SECTION ;
```

L'instruction include SQLCA

- Cela correspond à la déclaration d'une structure contenant toute l'information sur l'exécution de la dernière requête SQL
- La SQL Communication Area (SQLCA) est une structure qui permet d'analyser les erreurs d'exécution du SQL imbriqué

  EXEC SQL include SQLCA;
- L'application vérifie les champs de la structure SQLCA pour détecter un échec (ou un succès) d'une requête SQL
- Le champ sqlcode de la e structure a pour valeur
  - null (0) si la requête a réussi
  - un nombre négatif si il y a eu erreur
  - un nombre positif si la requête a réussi sous certaines conditions
    - par ex. 1403 si il n'y a aucun enregistrement résultat de la requête

Utilisation d'un curseur

- Lorsqu'une requête SELECT renvoie plus d'un enregistrement
- Déclaration du curseur

  EXEC SQL DECLARE nom_curseur CURSOR FOR requete_SQL;
- Ouverture du curseur

  EXEC SQL OPEN nom_curseur;
- Obtenir un enregistrement

  EXEC SQL FETCH nom_curseur INTO :uneVar, :uneAutreVar ;
- Fermer le curseur

  EXEC SQL CLOSE nom_curseur ;

Exemple

```
EXEC SQL DECLARE cur_client CURSOR FOR
        select num, nom, adresse, codepostal, telephone from clients;
EXEC SQL OPEN cur_client ;
EXEC SQL FETCH cur_client INTO :unClient;
while (sqlca.slqcode== 0) {
         unClient.nom.arr[unClient.nom.len] = '\0';
         unClient.adres.arr[cunClient.adres.len] = '\0';
        printf("%6d %10s %20s %6d %15s\n",
         unClient.num, unClient.nom.arr,
         unClient.adres.arr, unClient.cp,
         unClient.tel);
        EXEC SQL fetch cur_client INTO :unClient;
}
EXEC SQL CLOSE cur_client ;
```


Gestion des erreurs

- Toute requête SQL imbriquée peut générer une erreur
- WHENEVER est une directive du pré compilateur qui génère le code de gestion des erreurs pour chaque requête SQL

  EXEC SQL WHENEVER <condition> <action>

  SQLERROR
  SQLWARNING
  NOT FOUND

  CONTINUE
  DO *name (args)*
  DO BREAK
  DO RETURN
  GOTO *label*
  STOP

- Exemples :

  EXEC SQL WHENEVER SQLERROR GOTO erreur1;

  EXEC SQL WHENEVER SQLWARNING STOP;

Messages d'erreur

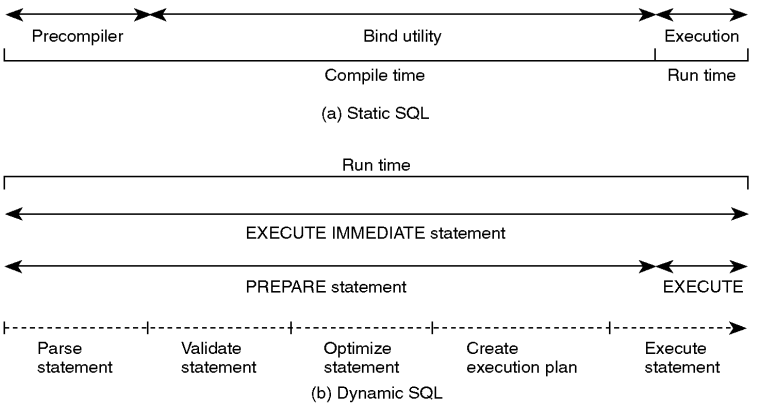
```
int  rapport_d_erreur ()
{   printf("Une erreur s'est produite\n");
    sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml]= '\0';
    printf("%s\n", sqlca.sqlerrm.sqlerrmc);
    return -1;
}

EXEC SQL WHENEVER SQLERROR DO rapport_d_erreur();
```


SQL dynamique

- Utile quand :
  - le format de la requête SQL n'est pas connu à la compilation
    - le SQL statique n'autorise pas l'utilisation des variables C pour des noms de tables ou de colonnes
  - le format est connu mais les objets n'existent pas lors de la compilation
- On va stocker la requête SQL dans une variable C, que l'on passera ensuite au SGBD pour interprétation
  - on utilise une chaîne de caractères pour enregistrer la requête SQL

```
strcpy(req_sql, "update employe set date_emb=sysdate where num_e = 1001");

EXEC SQL EXECUTE IMMEDIATE :req_sql;
```

SQL statique contre SQL dynamique



PREPARE et EXECUTE

- Le SGBD doit analyser, valider, et optimiser chaque requête EXECUTE IMMEDIATE, puis construire et exécuter le plan d'exécution
  - OK si la requête SQL n'est exécutée qu'une fois dans le programme
  - sinon inefficace
- Alternative : PREPARE et EXECUTE
- PREPARE demande au SGBD de lire une requête dynamique pour une exécution future : on donne un nom à la requête ainsi préparée

  EXEC SQL PREPARE nom_requete FROM :chaîne;
- Lorsqu'on fait exécuter la requête, il suffit de préciser son nom

  EXEC SQL EXECUTE nom_requete ;

Substitution de paramètres

- La requête EXECUTE USING permet compléter certaines portions non explicitées d'une requête préparée, en utilisant des identificateurs non déclarés

```
sprintf(req_sql, "update employe set salaire=:v1 where num= :v2");

EXEC SQL PREPARE req FROM :req_sql;

EXEC SQL EXECUTE req USING :nouv_salaire, :num_emp;
```