Résumé - Bases de données

1. Le modèle Entité/Association

Les entités
Définition 1 : on désigne par *entité* tout objet identifiable et pertinent pour l'application.
On parle d'*identifiant* ou de *clé* comme moyen technique pour distinguer les entités.

Les attributs
Les entités sont caractérisées par des propriétés appelées attributs. Un attribut est désigné par un nom et prend ses valeurs dans un domaine énumérable. Un attribut prend une valeur et une seule.

Types d'entités
Elle exprime un type, une classe, un ensemble dont les éléments sont les entités (ou occurrences).
Définition 2 : Le type d'une entité est composé de son nom, de ses attributs et de son identifiant.
Une entité est une instance du type E. Un ensemble d'entités $\{e_1, e_2, \ldots, e_n\}$ est une extension de E.
Définition 3 : soit E un type d'entité et A l'ensemble des attributs de E, une clé (ou identifiant) de E est un sous-ensemble minimal de A permettant d'identifier de manière unique une entité parmi n'importe quelle extension de E.
Il est possible d'avoir plusieurs clés pour un même ensemble d'entités. On en choisit alors une comme clé primaire et les autres comme clés secondaires. Un identifiant doit être

- *univalué* : à une occurrence correspond une seule valeur pour un identifiant donné,
- *discriminant* : à une valeur correspond une seule occurrence de l'individu
- *stable* : pour une occurrence donnée d'individu, une fois définie une valeur à son identifiant, cette valeur doit être conservée jusqu'à la destruction de l'occurrence
- *minimal* : s'agissant d'un identifiant composite, la suppression d'un de ses composants lui ferait perdre son caractère discriminant.

Association binaire
Elle modélise un ensemble d'associations de même nature entre 2 occurrences d'individus (de types différents ou de même type).
Définition 4 : une association binaire entre les types d'entités E_1 et E_2 est un ensemble de couples (e_1, e_2) avec $e_1 \in E_1$ et $e_2 \in E_2$.
Définition 5 : soit une association (E_1, E_2) entre 2 types d'entités, la cardinalité de l'association pour E_i, $i \in \{1,2\}$ est une paire {min,max} où min (resp. max) désigne le nombre minimal (resp. maximal) de fois où une entité e_i de E_i peut intervenir dans l'association.
On peut avoir des attributs qui ne peuvent être affectés qu'à l'association elle-même.
Définition 6 : la clé d'une association (binaire) entre un type d'entité E_1 et un type d'entité E_2 est le couple constitué de la clé c_1 de E_1 et de la clé c_2 de E_2.

Entité faible
Une entité ne peut exister qu'en étroite association avec une autre et est identifiée relativement à cette autre entité. On parle alors d'entité faible.

Associations généralisées
La définition d'une association n-aire est une généralisation de celle des associations binaires.
Définition 7 : une association n-aire entre n types d'entités $E_1, E_2, \ldots, E_n$ est un ensemble de n-uplets $(e_1, e_2, \ldots, e_n)$ où chaque e_i appartient à E_i.

Règle : soit A une association entre les types d'entité $(E_1, E_2, \ldots, E_n)$, pour sa transformation en type d'entité, il faut

1. attribuer un identifiant autonome à A,
2. créer une association A_i entre A et chacun des E_i. de cardinalité 1..1 (du côté de A) et 1..n de l'autre.

2. Le modèle relationnel

Une relation possède un nom, et se compose de colonnes désignées par un nom d'attribut avec des valeurs d'un certain domaine. Chaque ligne appelée tuple correspond à une entité. Le degré d'une relation est le nombre d'attributs (colonnes) et le cardinal d'une relation est le nombre de tuples (lignes). Un schéma relationnel est constitué d'un ensemble de schémas de relations.

Domaines
Un domaine de valeurs est un ensemble d'instances d'un type élémentaire (s'opposant au type structuré). Le système de types est figé et fourni par le système.

Attributs
Les attributs nomment les colonnes d'une relation. Le nom d'un attribut peut apparaître dans plusieurs schémas de relations.

Schéma de relation
Un schéma de relation est un nom suivi de la liste des attributs, chaque attribut n'étant présent qu'une seule fois et étant associé à son domaine. L'*arité* d'une relation est le nombre de ses attributs. $R(A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n)$

Instance d'une relation
Une instance d'une relation R se définit comme un sous-ensemble fini de produit cartésien des domaines des attributs de R (tous les tuples $(v_1,\ldots, v_n)$ où $v_i \in D_i$).

Clé primaire d'une relation
C'est le plus petit sous-ensemble des attributs qui permet d'identifier chaque ligne de manière unique. Le choix de la clé est très important pour la qualité du schéma.

Clé étrangère d'une relation
Attributs d'une relation qui correspondent à la clé primaire d'une autre (ou de la même) table.

Base de données
C'est un ensemble fini de relations.

3. Passage d'un schéma E/A à un schéma relationnel

Règles

- Entité : pour chaque entité, on crée une relation de même nom que l'entité et chaque propriété de l'entité, y compris l'identifiant, devient un attribut de la relation et l'identifiant constitue la clé.
- Entité faible : on crée un mécanisme de clé étrangère pour référencer l'entité forte dans l'entité faible. La clé étrangère est une partie de l'identifiant de l'entité faible.
- Association 1:n entre A et B : on crée les relations R_A et R_B correspondant aux entités A et B, l'identifiant de B devient un attribut de R_A. Une occurrence de A référence l'occurrence de B qui lui est associée à l'aide d'une clé étrangère.
- Association binaire n:n entre A et B : on crée les relations R_A et R_B correspondant aux entités A et B, on crée une relation R_{AB} pour l'association, la clé de R_A et la clé de R_B deviennent des attributs

de R_{AB}, la clé de cette relation est la concaténation des clés de R_A et R_B et les propriétés de l'association deviennent des attributs de R_{AB}.

- Association ternaire : on atteint une des limites du modèle E/A en matière de spécification de contraintes. En première approche, on peut appliquer la règle énoncée pour les associations binaires généralisées.
  En cas d'association entre plus de 2 entités, la clé de la table représentant l'association est un sous-ensemble de la concaténation des clés. Il faut se poser soigneusement la question de la clé au moment de la création de la table car elle ne peut plus être déduite du schéma E/A.

Choix des identifiants

Il est préférable en général de choisir un identifiant « neutre » qui ne soit pas une propriété de l'entité.
En effet :

1. Chaque valeur de l'identifiant doit caractériser de manière unique une occurrence.
2. Si on utilise un ensemble de propriétés comme identifiant, la référence à une occurrence est très lourde.
3. L'identifiant sert de référence externe et ne doit donc jamais être modifiable.

Le problème de l'identifiant « neutre » est qu'il ne donne pas d'indication sur l'occurrence qu'il réfère.

Dénormalisation du modèle logique

Deux objectifs principaux :

1. Simplifier le schéma relationnel en réduisant le nombre d'éléments qui le composent.
2. Faciliter l'accès aux données en introduisant un certain degré de redondance.

Ces techniques reviennent à introduire des anomalies dans le schéma. Il faut donc systématiquement comparer le gain attendu avec les risques courus.

- Suppression de relations : on peut supprimer les entités qui portent peu d'attributs en les déplaçant vers une autre relation.
- Introduction de redondance : l'accès à une information peut être long ou compliqué et justifie l'introduction de redondances.

4. Algèbre relationnelle

La sélection σ

La sélection $\sigma_F(R)$ extrait de la relation R les tuples qui satisfont un critère de sélection F (=, <, >, ≥, ≤).

La projection π

La projection $\pi_{A1,\ldots, Ak}(R)$ ne garde que les attributs $A_1,\ldots, A_k$ de R. Contrairement à la sélection, on supprime des colonnes.

Le produit cartésien ×

R×S permet de créer une nouvelle relation où chaque tuple de R est associé à chaque tuple de S. Le nombre de ligne est $|R|*|S|$.

L'union ∪

R∪S crée une relation comprenant tous les tuples existants dans l'une ou l'autre des relations R et S. Les deux relations doivent avoir le même schéma.

La différence –

R-S a pour résultat tous les tuples de R qui ne sont pas dans S. Les deux relations doivent avoir le même schéma.

La jointure ∞

Elle consiste à rapprocher les lignes de deux relations pour lesquelles les valeurs d'un (ou plusieurs) attributs sont identiques.
$R \infty_F S = \sigma_F(R \times S)$, F étant une opération de comparaison liant un attribut de R à un attribut de S.

La division ÷

Elle permet de déterminer les occurrences de la première relation qui sont associées à toutes les occurrences de la seconde.

X : ensemble des attributs de R
Y : ensemble des attributs de S
Z : ensemble des attributs communs à X et à Y
R÷S renvoie les tuples r de R sur les attributs X-Z qui vérifient que pour tout tuple s de S sur les attributs de Z, le tuple rs (sur les attributs de X) appartient à R.
$R \div S = \pi_{X-Z}(R) - \pi_{X-Z}((\pi_{X-Z}(R) \times \pi_Z(S)) - R)$

Expression de requêtes avec l'algèbre

Station(**nomStation**, capacité, lieu, région, tarif)
Activité(*nomStation*, **libellé**, prix)
Client(**idClient**, nom, prénom, ville, région, solde)
Séjour(*idClient*, **nomStation, début**, nbPlaces)

Sélection généralisée

On peut généraliser les critères de sélection de σ. La composition de plusieurs sélections permet d'exprimer une *conjonction* de critères de recherche.
$\sigma_{capacité<200 \wedge région='Antilles'}(Station)$
De même, la composition de la sélection et de l'union permet de d'exprimer la *disjonction*.
$\sigma_{capacité<200 \vee région='Antilles'}(Station)$
Enfin, la différence permet d'exprimer la *négation* et d'éliminer des lignes.
$\sigma_{capacité<200 \wedge région\neq'Antilles'}(Station)$
Les opérateurs d'union et de différence permettent de définir une sélection σ_F où le critère F est une expression booléenne.

Requêtes conjonctives

Elles constituent l'essentiel des requêtes courantes. Ce sont celles avec des « et » (par opposition au « ou » et « non »). Elles s'écrivent avec des π, des σ et des × (et donc indirectement avec des ∞). On utilise la jointure dès que les attributs nécessaires pour évaluer une requête sont répartis dans plusieurs tables.
« Nom des clients qui sont partis en vacances dans leur région ainsi que le nom de cette région ? » :
$\pi_{nom,client.région}(Client \infty_{idClient=idClient, région=région}(Séjour \infty_{station=nomStation} Station))$

Requêtes avec -

Elles permettent d'exprimer « tous les O qui ne satisfont pas p ». On construit alors la requête A qui sélectionne tous les O, ensuite la requête B qui sélectionne tous les O qui satisfont p et finalement on fait A-B.
La différence peut être employée pour calculer le complément d'un ensemble.
« IdClient des clients et les stations où ils ne sont pas allés ? »
$(\pi_{idClient}(Client) \times \pi_{nomStation}(Station)) - \pi_{idClient,station}(Séjour)$

Quantification universelle

Une propriété est vraie pour tous les éléments d'un ensemble s'il n'existe pas un élément de cet ensemble pour lequel la propriété est fausse. On emploie alors la négation et la quantification existentielle.
« Quelles sont les stations dont toutes les activités ont un prix supérieur à 100 ? »
$\pi_{nomStation}(Station) - \pi_{nomStation}(\sigma_{prix<100}(Activité))$

Les requêtes les plus complexes sont celles qui utilisent la division.
« Quelles sont les activités qui sont dans toutes les stations ? »
Activité ÷ Station
« IdClient des clients qui sont allés dans toutes les stations ? » :
$\pi_{idClient}(Client) - \pi_{idClient}((\pi_{idClient}(Client) \times \pi_{nomStation}(Station)) - \pi_{idClient,station}(Séjour))$

5. SQL

Requêtes simples

Le résultat d'un ordre SQL est toujours une relation dont les attributs sont ceux spécifiés dans la clause SELECT. On peut donc considérer en première approche ce résultat comme un 'découpage' horizontal et vertical de la table indiquée dans le FROM, similaire à une utilisation combinée de la sélection et de la projection.
On peut renommer les attributs (en utilisant le mot-clé optionnel AS), appliquer des fonctions aux valeurs de chaque tuple et introduire des constantes.

```
SELECT libelle, prix/6,56 AS prixEnEuros
FROM Activite
WHERE nomStation = 'Santalba'
```

La spécification de clé permet d'éviter les doublons dans les relations stockées, mais ils peuvent apparaître dans le résultat d'une requête. Pour éviter d'obtenir deux tuples identiques, on utilise le mot-clé DISTINCT, mais cette opération peut être coûteuse.

Il est possible de trier le résultat d'une requête avec la clause ORDER BY

```
SELECT *    (* : tous les attributs)
FROM Station
ORDER BY tarif, nomStation
```

Pour trier en ordre descendant, on utilise le mot-clé DESC après la liste des attributs.
Dans la clause WHERE, on spécifie une condition booléenne (AND, OR, NOT, =, <, <=, >, >=, <>) portant sur les attributs des relations du FROM. Pour obtenir une recherche par intervalle, on utilise le mot-clé BETWEEN.

Pour les chaînes de caractères, attention à la différence entre chaînes de longueur fixe et celles de longueur variable ; et attention à la différence majuscule/minuscule.
SQL fournit des options de recherche par motif à l'aide de la clause LIKE : '_' désigne n'importe quel caractère et '%' n'importe quelle chaîne.
Une date est spécifiée par le mot-clé DATE au format 'aaaa-mm-jj'

```
SELECT IdClient
FROM Sejour
WHERE debut BETWEEN DATE '1999-07-01' AND DATE '2006-07-01'
```

On admet que la valeur de certains attributs soit inconnue en utilisant le mot-clé NULL. On ne peut donc lui appliquer aucune des opérations ou comparaisons usuelles. Toutes les opérations appliquées à NULL retourne le résultat NULL. Toute comparaison avec NULL donne UNKNOWN.
TRUE = 1, FALSE = 0 et UNKNOWN = ½.
x AND y = min(x,y)
x OR y = max(x,y)
NOT x = 1-x
Les conditions exprimées dans une clause WHERE sont évaluées pour chaque tuple qui est conservé si cette évaluation donne TRUE. La présence d'une valeur nulle dans une comparaison a donc souvent le même effet que si cette comparaison échoue et renvoie FALSE. La présence de NULL peut avoir des effets surprenants.
NULL est un mot-clé pas une constante. Le prédicat pour tester l'absence de valeur dans une colonne est 'x IS NULL' (ou 'x IS NOT NULL'). Dans la mesure du possible, il faut éviter NULL en spécifiant la contrainte NOT NULL ou en donnant une valeur par défaut.

Requêtes sur plusieurs tables

- Jointure

On donne simplement la liste des tables concernées dans la clause FROM et on exprime les critères de rapprochement entre ces tables dans la clause WHERE.

On construit le produit cartésien des tables du FROM, en préfixant chaque attribut par le nom ou le synonyme de sa table pour éviter les ambiguïtés. Il n'y a alors plus qu'une seule table sur laquelle on interprète l'ordre SQL comme dans le cas d'une requête simple.
« Nom des clients habitant Paris, les stations où ils ont séjourné avec la date et le tarif hebdomadaire pour chaque station ? »

```
SELECT nom, nomStation, début, tarif
FROM Client, Sejour, Station
WHERE ville = 'Paris'
AND client.idClient = sejour.idClient
AND station.nomStation = sejour.nomStation
```

- Union, intersection et différence

UNION, INTERSECT ou EXCEPT.
« Tous les noms des régions dans la base ? »

```
SELECT region FROM Station
UNION
SELECT region FROM Client
```

« Régions où on trouve des stations mais pas des clients ? »

```
SELECT region FROM Station
EXCEPT
SELECT region FROM Client
```

L'union ne peut être exprimé autrement qu'avec UNION. Par contre, INTERSECT peut être exprimé avec une jointure et la différence s'obtient plus aisément à l'aide des requêtes imbriquées.

Requêtes imbriquées

« Nom des stations où ont séjourné des clients parisiens ? »

```
SELECT nomStation
FROM Client, Sejour
WHERE ville = 'Paris'
AND client.idClient = sejour.idClient
⇒ SELECT nomStation
  FROM Sejour
  WHERE idClient IN (SELECT idClient
                     FROM Client
                     WHERE ville = 'Paris')
```

- EXISTS R, renvoie True si R n'est pas vide, False sinon (ou NOT EXISTS).
- t IN R, renvoie True si le tuple t appartient à R, False sinon (ou NOT IN).
- v cmp ANY R, renvoie True si la comparaison de l'attribut v avec un moins un des tuples de R est vérifiée, False sinon, cmp ∈ {<, >, =, …}.
- v cmp ALL R, renvoie True si la comparaison de l'attribut v avec tous les tuples de R est vérifiée, False sinon, cmp ∈ {<, >, =, …}.

La différence s'exprime facilement avec NOT IN ou NOT EXISTS.

- Sous-requêtes corrélées

« Quels sont les clients (nom, prénom) qui ont séjourné à Santalba ? »

```
SELECT nom, prenom
FROM Client
WHERE EXISTS (SELECT *
              FROM Sejour
              WHERE nomStation = 'Santalba'
              AND client.idClient = sejour.idClient)
```

Fonctions d'agrégation

- COUNT qui compte le nombre de valeurs non nulles,
- MAX, MIN
- AVG qui calcule la moyenne des valeurs de la colonne,
- SUM qui effectue le cumul.

```
SELECT COUNT(nomStation), AVG(tarif), MIN(tarif), MAX(tarif)
FROM Station
```

On ne peut pas utiliser simultanément dans la clause SELECT des fonctions d'agrégation et des noms d'attributs, sauf dans le cas d'un GROUP BY.

La clause GROUP BY

« Afficher les régions avec le nombre de stations »

```
SELECT region, COUNT(nomStation)
FROM Station
GROUP BY region
```

La clause HAVING

« Nombre de places réservées par client, pour les clients ayant réservé plus de 10 places ? »

```
SELECT nom, SUM(nbPlaces)
FROM Client, Sejour
WHERE client.idClient=sejour.idClient
GROUP BY nom
HAVING SUM(nbPlaces) >= 10
```

Mises à jour

Insertion

INSERT INTO R(A_1, A_2, …, A_n) VALUES (v_1, v_2, …,v_n)

R est le nom de la relation, A_1, A_2, …, A_n sont les noms des attributs dans lesquels on souhaite placer une valeur (les autres attributs seront à NULL ou à la valeur par défaut) et v_1, v_2, …,v_n sont les valeurs.

Il est également possible d'insérer dans une table le résultat d'une requête. La partie VALUES est remplacée par la requête.

Destruction

```
DELETE FROM R
WHERE condition
```

Condition est une condition valide pour toute clause WHERE.

Modification

```
UPDATE R SET A1=v1, A2=v2, …, An=vn
WHERE condition
```

« Augmenter les prix des activités de la station Passac de 10% »

```
UPDATE Activite
SET prix=prix*1.1
WHERE nomStation='Passac'
```

Toutes les mises à jour ne deviennent définitives qu'à l'issue d'une validation par COMMIT. Entre-temps, elles peuvent être annulées par ROLLBACK.

Utilisateurs

L'accès à une base de données est restreint à des utilisateurs connus du SGBD et identifiés par un nom et un mot de passe. Chaque utilisateur se voit attribuer certains droits sur les schémas et les tables.

```
CONNECT utilisateur
```

Le propriétaire (concepteur) d'un schéma a tous les droits sur les éléments de ce schéma. Il définit les droits des autres utilisateurs sur les éléments de ce schéma.

```
GRANT <privilège>
ON <élément du schéma>
```



```
TO <utilisateur>
[WITH GRANT OPTION]
```

privilège = INSERT (insertion), UPDATE (modification), SELECT (recherche), DELETE (destruction), REFERENCE (pour permettre de faire référence à une table dans une contrainte d'intégrité), USAGE (pour permettre l'utilisation une définition ≠ table ou insertion).

Pour accorder un privilège, il faut en avoir le droit soit parce qu'on est propriétaire de l'élément du schéma, soit parce que le droit est accordé par la commande WITH GRANT OPTION.

On peut désigner tous les utilisateurs avec le mot-clé PUBLIC, et tous les privilèges avec l'expression ALL PRIVILEGES.

On supprime un droit avec la commande

```
REVOKE <privilège>
ON <élément du schéma>
FROM <utilisateur>
```

Définition d'un schéma

Un schéma est l'ensemble des déclarations décrivant une base de données au niveau logique. Outre les tables, un schéma peut comprendre des vues, des contraintes de différents types, des triggers (procédures déclenchées par certains événements), etc.

On crée un schéma en lui donnant un nom puis en donnant la liste des commandes créant les éléments.

```
CREATE DATABASE officiel_des_spectacles
CREATE TABLE Film …
CREATE VIEW …
CREATE ASSERTION …
CREATE TRIGGER …
```

Pour choisir le schéma courant (déjà créé), on utilise la commande

```
USE officiel_des_spectacles
```

Contraintes et assertions

Ces contraintes portent le plus souvent sur la restriction des attributs à un ensemble de valeurs, mais on peut trouver des contraintes plus complexes faisant référence à d'autres relations. Une contrainte s'exprime par la commande CHECK (condition).

```
CREATE TABLE Salle
(NomCine    VARCHAR (30) NOT NULL,
No          INTEGER,
Capacite    INTEGER CHECK (Capacite < 300),
Climatise   CHAR(1) CHECK (Climatise IN ('O', 'N')),
PRIMARY KEY (NomCine, No),
FOREIGN KEY NomCine REFERENCES Cinema)
```

Au lieu d'associer une contrainte à un attribut particulier, on peut la définir globalement avec CONSTRAINT en donnant un nom à chaque contrainte.

« Toute salle de plus de 300 places doit être climatisée ».

```
CONSTRAINT clim CHECK (Capacite < 300 OR Climatise = 'O')
```

On peut modifier ou détruire une contrainte : ALTER TABLE … DROP CONSTRAINT ...

```
ALTER TABLE Salle DROP CONSTRAINT clim
```