# 5. Le Langage SQL

C'est un langage fourni avec tout SGBD relationnel commercialisé. C'est un standard reconnu par l'IS0 depuis 87 (standard donc portabilité). On en est à la version 2 (SQL92) et la version 3 est annoncée pour bientôt. SQL est un LDD et un LMD. Il est aussi utilisé pour définir des vues, les droits d'accès, manipulation de schéma physique [6, 11].

*Remarque : presque tous les exemples portent sur le schéma suivant*

*MOVIE(TITRE, PAYS, ANNEE, REALISATEUR)*

*DISTRIBUTION(TITRE,ACTEUR)*

*CINEMA(NOMCINE, RUE, TEL)*

*CINESALLES(NOMCINE, SALLE,NBPLACES)*

*PROGRAMME(NOMCINE, SALLE, TITRE, SEMAINE,NBENTREES)*

## 5.1. Structure de base

Une requête SQL simple possède la forme suivante :

Select $A_1, \ldots, A_n$

From $r_1, \ldots, r_m$

Where $P$

Les Ai sont des attributs, les rj sont des noms de relations et P est un prédicat.

Cette requête est équivalente à $\pi\, A_1, \ldots, A_n\, (\sigma_P(r_1 \times \ldots \times r_m))$

### 5.1.2 La clause Select

La clause SELECT assure la projection de l'algèbre relationnel.

Pour lister tous les réalisateurs des films :

```
Select    realisateur
From      movie
```

L'utilisation de l'étoile (*) permet de sélectionner tous les attributs de la relation:

```
Select    *
From      movie
```

SQL retourne par défaut les doublons. Pour le forcer à les éliminer, on utilise la clause DISTINCT :

```
select distinct  realisateur
From                    movie
```

Avec SELECT, on peut utiliser des expressions arithmétiques ainsi que le renommage d'attributs :

```
Select    Prix_HT * 1.17 AS Prix TTC
From produit
```

### 5.1.3. La clause Where

Elle représente le prédicat de sélection dans l'algèbre relationnel. La condition concerne les attributs des relations qui figurent dans la clause FROM

```
Select distinct        realisateur
From                   movie
```

```
Where        realisateur = "Samir" and Acteur = "Ahmed"
```

SQL utilise les connecteurs and, or et not. Pour simplifier la clause WHERE, on peut utiliser la clause between.

```
Select    Num
From      compte
Where     Solde  between 0 and 10000
```

### 5.1.4. La clause From

La clause from représente le produit cartésien de l'algèbre relationnel.

Le titre et le réalisateur des films programmés à l'DZAIR d' Alger.

```
Select    Titre, Réalisateur
From movie, programme
Where    movie.titre = programme.titre and programme.NomCiné = "DZAIR"
```

### 5.1.5. Les variables n-uplets

Elles sont définies dans la clause FROM

```
Select      titre, réalisateur
From        movie as f, programme as p
Where       f.titre = p.titre and p.nomciné = "dzair"
```

Soit emp (id, nom, id_chef)

```
Select       el.nom, e2.nom as nom_chef
From         emp el, emp e2
Where        el.id_chef = e2.id
```

### 5.1.6. La clause Order by

Sql permet de trier les résultats de requête

```
Select      *
From        programme
Where       nomciné="dzair"
Order by    horaire asc, titre desc
```

### 5.1.7 opérateurs ensemblistes

```
Select ...
…
Union/ intersect/ except
Select ...
```

***Remarque*** : ces opérations éliminent les doublons, pour pouvoir les garder utiliser à la place intersect all... Si t apparaît m fois dans r et n fois dans s alors il apparaît :

- M + n fois dans r union all s
- Min(*m, n)* fois dans *r* intersect all s
- Max(0, *m - n)* fois dans *r* except all s

## 5.2. Les fonctions d'agrégats

Ce sont des fonctions qui agissent sur des ensembles (multi-ensembles) de valeurs :

Avg : la valeur moyenne de l'ensemble

Min : la valeur minimale

Max : la valeur maximale

Sum : le total des valeurs de l'ensemble

Count : le nombre de valeur dans l'ensemble

```
Select count(titre) from programme
```

Cette requête retourne le nombre de films programmés à alger.

***Remarque :*** un même titre peut être compté plusieurs fois s'il est programmé à des heures différentes et dans des salles différentes.

```
Select count( distinct titre) from programme
```

### 5.2.1 agrégats et group by

Le nombre de films programmés dans chaque salle :

```
Select    nomciné, count (distinct titre)
From      programme
Group by nomciné
```

Les attributs qui apparaissant dans la clause select en dehors des agrégats *doivent* être associés à la clause group by

### 5.2.2. Agrégats et la clause having

Les salles où sont programmés plus de 3 films :

```
Select    nomciné, count(distinct titre)
From      programme
Group by nomciné
Having count (distinct titre) > 3
```

Le prédicat associé à la clause having est testé après la formation des groupes définis dans la clause group by.

### 5.2.3. Requêtes imbriquées

Sql fournit un mécanisme qui permet d'imbriquer les requêtes. Une sous requête est une requête sql (select-from-where) qui est incluse dans une autre requête. Elle apparaît au niveau de la clause where de la première requête.

Les films programmés à l'dzair non programmés à chelia

```
Select    titre
From      programme
Where     nomciné="dzair" and titre not in (
        Select    titre
        From      programme
        Where     nomciné ="chelia" )
```

Trouver les comptes dont les soldes sont supérieurs aux soldes des comptes de taha :

*Compte (num, solde, nomtit)*

```
Select    *
From      compte
Where     solde > all (
          Select    solde
          From      compte
          Where     nomtit = "taha" )
```

En remplaçant all par some, on obtient les comptes dont les soldes sont supérieurs au solde d'au moins un compte de taha.

Les cinémas qui passent tous les films programmés à l'dzair

```
Select    nomciné
From      programme pi
Where not exists (
          (select distinct titre
          From      programme
          Where     nomciné-"dzair")
          Except
          (select distinct titre
          From programme p2
          Where pl.nomciné = p2.nomciné))
```

***Test d'absence de doublons***

La clause unique permet de tester si une sous requête contient des doublons.

Les titres de films programmés dans une seule salle et un seul horaire :

```
Select    p.titre
From      programme p
Where unique (
          Select    pl.titre
          From      programme p1
          Where     p.titre = pl titre)
```

***Les relations dérivées***

Titulaire (nom, adresse)

Compte (num, solde, nomtit)

Donner le solde moyen des comptes de chaque personne ayant un solde moyen > à 1000

```
Select    nomtit, soldemoyen
From (
          Select    nomtit, avg(solde)
          From      compte
```

```
        Group by nomtit )
    As result (nomtit, soldemoyen)
Where soldemoyen > 1000
```

Noter qu'on aurait pu exprimer cette requête en utilisant la clause having

### 5.2.4. Les vues

Equivalent à une requête access. Une vue peut être considérée comme une relation quelconque lors de l'expression des requêtes. Une vue est une relation virtuelle dans le sens où elle ne contient pas effectivement des « tuples ». Elles permettent de définir des relations *virtuelles* dans le but de :

- Cacher certaines informations à des utilisateurs,
- Faciliter l'expression de certaines requêtes,
- Améliorer la présentation de certaines données.

Une vue est définie par une expression de la forme :

Create view v as requête

Requête est une expression quelconque de requête et v est le nom de la vue.

Emp (nume, salaire, dept, adresse)

```
Create view empgen as (
    Selec t nume, dept, adresse
    From emp )
```

Toutes les informations concernant les employés du département 5 :

```
Select   *
From     empgen
Where    dept = 5
```

### 5.2.5. Modification des relations

***Suppression :***

Supprimer tous les employés du département 5 : delete from emp

Where dept = 5

Supprimer du programme tous les films programmés à l'dzair où un des acteurs est ahmed :

```
Delete from programme
Where nomciné = "dzair' and exists (
    Select   titre
    From     movie
    Where    programme.tite = movie.titre and movie.acteur = "ahmed" )
```

Supprimer les comptes dont le solde est < à la moyenne des soldes de tous les comptes :

```
Delete from compte
Where solde < (select avg (solde) from compte)
```

***Problème*** : si les n-uplets sont supprimés un à un de la relation compte, alors à chaque suppression, on a une nouvelle valeur de avg (solde). La solution de sql est de d'abord, calculer avg(solde) et ensuite de supprimer les tuples satisfaisant le test sans recalculer à chaque fois la nouvelle valeur de avg (solde).

***Insertion***

Insérer un n-uplet dans la relation "compte" :

```
Insert into compte (num, solde, nomtit) values (511,1000, "taha")
```

Ou bien `insert into compte values (511,1000, «taha")`

Insère un n-uplet avec un solde *inconnu.*

```
Insert into compte values (511, null, "taha")
```

Ou bien `insert into compte(num, nomtit) values (511, "taha")`

Les 2 dernières maj sont équivalentes sauf si une valeur par défaut du solde a été spécifiée lors de la définition de la table compte.

Supposons qu'on a crée une relation titmoy (nomtit, moyenne) qui doit contenir le nom des clients de la banque ainsi que la moyenne des soldes de leurs comptes.

```
Insert into titmoy (nomtit, moyenne)
Select      nomtit, avg(solde)
From        compte
Group by nomtit
```

***Update***

Rajouter 1% à tous les comptes dont le solde est inférieur à 500 :

```
Update compte
Set solde = solde * 1.01
Where solde ≤ 500
```

La condition qui suit la clause where peut être une requête sql.

## 5.3. Sql en tant que ldd

- Le schéma des relations
- Les domaines des attributs
- Les contraintes d'intégrité
- La gestion des autorisations
- La gestion du stockage physique
- Les index associés à chaque relation

### 5.3.1. Domaines

- **Char(n)** : chaîne de caractères de taille fixe n
- **Varchar(n)** : chaîne de caractères de taille variable mais inférieure à n
- **Int** : entier (un sous ensemble fini des entiers, dépend de la machine)
- **Smallint** : entier. Sous ensemble de int
- **Numeric(p,d)** : réel codé sur *p* digits et max *d* digits pour partie à droite de la décimale.

- **Real** : un réel flottant.
- **Date** : yyyy-mm-dd (année, mois, jours)
- **Time** : hh :mm :ss (heure, minute, seconde)

Les valeurs nulles (null) sont possibles dans tous les domaines. Pour déclarer qu'un attribut ne doit pas être nul, il faut rajouter not null

- Create domain nom-client char(20)

### 5.3.2. Création des tables

- On utilise la clause create table

```
Create table compte (
        Num int not null,
        Solde int,
        Nomtit varchar( 20))
```

- Rajout de contraintes

```
Create table compte (
    Num int not null,
    Solde int default 0,
    Nomtit varchar(20),
    Primary key (num),
    Check (num > 1) )
```

- En sql92, si un attribut est clé alors il est différent de null.

### 5.3.3. Manipulation de schéma

La commande drop table permet de supprimer une table.

Ex: drop table compte.

Si une vue est définie sur la table compte alors il faut utiliser

Drop table compte cascade

La commande alter table permet de modifier le schéma d'une relation.

*Exemple* : alter table compte add date_ouverture

Date

Alter table compte drop solde cascade

### 5.3.4. Clé étrangère

Soient personne (nss, nom) et voiture (matricule, modèle, proprio).

« proprio » correspond au nss du propriétaire. C'est une *clé étrangère* dans le schéma voiture car c'est une clé dans un autre schéma.

```
Create table voiture (
    Matricule char(8),
    Modele  char(10),
    Proprio  char(3),
    Primary key(matricule),
```

Foreign key(proprio) references personne

On [delete | update] cascade |

Restrict |

Set null

)

Cascade: si une personne est supprimée, alors les voitures qu'elle possède sont supprimées.

Restrict : le système refuse la suppression d'une personne s'il y a des voitures qui lui sont rattachées. C'est l'option par défaut.

Set null: si une personne est supprimée, alors l'attribut proprio prend la valeur null.

L'insertion d'une voiture ne peut se faire que si le «proprio » existe dans personne (ou bien valeur nulle).

***Valeurs nulles***

| Employé | Nom | Salaire |
|---|---|---|
| | Taha | 10 000 |
| | Redha | Null |

Select *

From employé

Where salaire > 12000 *ne retourne aucun tuple.*

*Pareil si la condition est* where salaire < 8000

Select sum (salaire) from employé; *retourne 10000*

Select count (salaire) from employé; *retourne 2*

Select avg (salaire) from employé; *retourne 10000*

Très différent de select sum (salaire) / count (salaire) from employé car count prend en compte la valeur null donc cela fera 10000/2=5000.

*En fait c'est équivalent à :*select sum (salaire) / count (salaire) from employé

where salaire is not null

Select count(*) from employé

Where salaire is not null; *retourne 1*

***Mise à jour des vues***

Personne (nom, salaire). Supposons que la table personne est vide.

Create view gros_salaire as

Select *

From personne

Where salaire > 10000

Insert into gros_salaire values("redha", 5000)

L'effet de cette commande est d'insérer dans la table *personne* le tuple < "redha", 5000>.

Noter que si l'on fait :

Select * from gros salaire; on n'obtient aucun tuple.

Si à la création de la vue on rajoute l'option :

With check option alors l'insertion est refusée.

Les mises à jour des vues sont traduites en des mises à jours des tables sous-jacente. *La traduction n'est pas toujours unique.* Quand nous avons plusieurs manières de traduire une mise à jour alors celle-ci est rejetée. ⇒ certaines vues ne permettent pas des mises à jour. Il faut une relation biunivoque entre la mise à jour de la vue et la mise à jour de la table.

### 5.3.5. Jointure externe

Si on fait : personne ixi voiture, on n'aura que les personnes qui ont une(des) voiture(s) qui sont dans le résultat.

```
Select *
From    personne p left outer join voiture v
        On p.nss = v.proprio
```

Cette requête retourne aussi les personnes n'ayant pas de voiture. Ces tuples auront des *valeurs nulles* pour les champs provenant de voiture. Si on met juste outer join alors on aura les personnes sans voitures et les voitures sans propriétaire.

La jointure est exprimée par : t1 inner join t2 on condition

Dans l'exemple, si l'on veut joindre personne et voiture alors

```
Select *
From    personne p inner join voiture v
        On p.nss = v.proprio
```

Si l'on mets right à la place de left dans la jointure, alors on aura les voitures sans les personnes. Si on ne mets ni left ni right, on aura les voitures et les personnes qui ne sont pas dans la jointure.

### 5.3.6. Mécanisme des droits

Soit la table pers (num, nom, adr, num_serv, salaire)

1. Taha ne peut pas accéder à pers.
2. Taha peut lire une partie de pers mais ne peut rien modifier.
3. Taha peut lire un seul tuple (celui le concernant) sans pouvoir le modifier.
4. Taha peut en plus modifier l'attribut adr.
5. Taha peut accéder à l'attribut salaire mais seulement entre 9h et 17h à partir du terminal 25.
6. Taha peut modifier salaire si celui-ci est inférieur à 8000.
7. Taha peut modifier la relation s'il est responsable du num_serv du tuple qu'il veut modifier.

Les droits dépendent du contenant, du contexte et/ou du contenu.

### 5.3.7. Les droits dans sql

Select: privilège qu'il faut posséder pour lire une table

Insert, delete, update: privilèges nécessaires pour mettre à jour une table.

Insert(x), update(x): privilège nécessaire pour insérer, mettre à jour l'attribut x.

***Octroi et retrait de privilèges***

Grant privilège on objet to utilisateur [with grant option]

Revoke [grant option for] privilège on objet from utilisateur

Restrict | cascade

***Exemples :***

Grant all on table résultat to directeur with grant option;

Grant insert on table résultat to sec_1;

Grant select, update(points) on table resultat to prof_1;

Remarque : un utilisateur peut recevoir le même privilège de plusieurs sources. Cela est utile quand l'une d'elles veut le lui retirer.

***Exemple*** : soit la séquence

A: grant select on table t tobwith grant option

B: grant select on table t tocwith grant option

C: grant select on table t todwith grant option

A: revoke select on table t from b

Ni b ni c ne pourront lire t

### 5.3.8. Utilisation des vues

Create view informations_perso

As select *

From employé

Where nom = user;

Grant select, update(adresse)

On informations_perso

To public