Présentation du langage SQL

SQL (Structured Query Language)

Un langage pour manipuler un schéma de BD relationnel.

- Initié par IBM pour donner un langage à son implémention du modèle relationnel: SEQUEL (Structured English as QUEry Language).
- Langage SQL normalisé:
  - SQL 2 (1992): définis différents niveaux de conformité,
  - SQL 3 (1999): ajoute les concepts "objets", entrepôts de données, réplication, etc.
  - etc.

SQL: 3 langages en 1

- LID (langage d'interrogation de données)
  - Select … From … Where …
  - Cf. cours SQL suivant pour l'interrogation.
- LMD (langage de manipulation de données)
  - Insertion de données,
  - Modification de données,
  - Suppression de données.
- LDD (Langage de définition de données)
  - Création de schéma,
  - Gestion des contraintes d'intégrité,
  - Évolution de schéma.

SQL pour l'interrogation

- Présentation
- SQL vs. algèbre
- Opérateurs spécifiques à SQL
- Notion de sous-requêtes
- Agrégation et regroupement
- Mises à jour.

Structure générale d'une requête : le BLOC

- Structure d'une requête formée de quatre clauses "importantes":

  SELECT [DISTINCT|UNIQUE|ALL] <liste-attributs>
  FROM <liste-tables>
  WHERE <liste-prédicats>
  GROUP BY <critère-regroupement>

  - SELECT: définit le format du résultat cherché ≈ projection
  - FROM: définit la source des données ≈ opérandes
  - WHERE: contraint les données du résultat ≈ sélection
  - GROUP BY: réalise un groupement des tuples résultats

- Exemple: Donner toutes les infos sur les personnes.

  SELECT * FROM Personne;

nP	**Nom**	**Adr**
1	Dupont	Grenoble
2	Aubry	Dijon
3	Duval	Grenoble

SQL/Algèbre

- Il est possible de trouver une équivalence SQL à chacun des opérateurs de l'algèbre: les plus simples sont l'identité, la projection et la sélection.
- L'identité en algèbre **Id(R)**

  SELECT * FROM R;
  - **rien de spécial: on veut tous les tuples d'une table R;**
  - **Important: on indique la table sur laquelle on travaille derrière la clause FROM.**
- La projection en algèbre $\pi_{a1,a2,\ldots an}$ **(R)**

  SELECT $a_1,a_2,\ldots a_n$ FROM R
- La sélection en algèbre $\sigma_{\text{prédicat}}$ **(R)**

  SELECT * FROM R WHERE prédicat
- Le renommage en algèbre : **R2 =** $\rho_{a1 \rightarrow b1, \ldots, an \rightarrow bn}$ **(R)**

  SELECT a1 AS b1, ..., an AS bn

  FROM R R2 (R2 est une copie logique de R)

La projection en SQL

- Syntaxe complète:

  SELECT [DISTINCT] listeColonnes
  FROM table [alias]
  [ordonnancement]

- Extraction de toutes les colonnes: utilisation du symbole *

SELECT * FROM Personne;

nP	Nom	Adr
1	Dupont	Grenoble
2	Aubry	Dijon
3	Duval	Grenoble

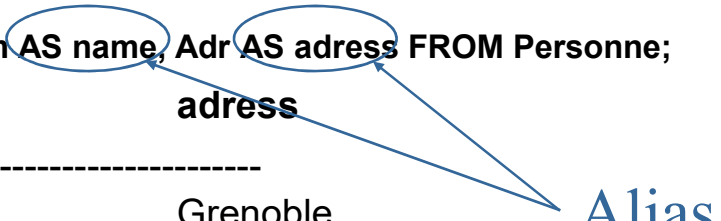
- Extractions de certaines colonnes: on précise leurs noms

SELECT Nom, Adr FROM Personne;

Nom	Adr
Dupont	Grenoble
Aubry	Dijon
Duval	Grenoble

SELECT Nom AS name, Adr AS adress FROM Personne;

name	adress
Dupont	Grenoble
Aubry	Dijon
Duval	Grenoble



La projection en SQL

- Gestion des alias
  - de colonne: utilisation du mot clé AS

  SELECT ancien_nom AS nouveau_nom
  - de table/renommage: (utilité, cf. la suite)

  FROM ancien_nom nouveau_nom
- Gestion des duplicatas
  - le résultat d'une requête SQL affiche tous les tuples correspondant aux critères de la requête, donc des tuples redondants peuvent apparaître.
  - le mot clé DISTINCT supprime les tuples redondants du résultat en utilisant certains attributs pour identifier les duplicatas.

SELECT Adr FROM Personne;

Adr
Grenoble
Dijon
Grenoble

SELECT DISTINCT(Adr) FROM Personne;

Adr
Grenoble
Dijon

La projection en SQL

- Ordonnancement des résultats: clause ORDER BY pour trier les tuples résultats

  ORDER BY {expression|position} [ASC|DESC] [NULL FIRST|NULL LAST]

  - expression: nom d'attribut ou résultat calculé,
  - position: position d'une expression dans SELECT,
  - ASC (défaut),DESC: trie ascendant ou descendant,
  - NULL FIRST, LAST: valeurs NULL au début ou à la fin.

SELECT * FROM Personne ORDER BY Nom ASC, nP DESC

nP	Nom	Adr
2	Aubry	Dijon
1	Dupont	Grenoble
3	Duval	Grenoble

La sélection en SQL

- La clause WHERE permet de restreindre l'ensemble des tuples résultats. On utilise
  - des opérateurs de comparaison (<,=,<,>=,<=,<>) pour comparer la valeur d'un attribut avec celle d'un autre, d'une constante, etc.
  - des opérateurs spéciaux: BETWEEN, IN, LIKE, IS [NOT] NULL
  - des opérateurs logiques servant à combiner le résultat des comparaisons (AND, OR) ou d'inverser un prédicat (NOT).

SELECT * FROM Personne WHERE Adr='Grenoble' and nP>1;

nP	**Nom**	**Adr**
3	Duval	Grenoble

Opérations ensemblistes

- **R ∪ S, R ∩ S et R - S**
  - En SQL BlocSQL {UNION | INTERSECT | MINUS} BlocSQL
  - Comment reconstruire l'ensemble des numéros de personne à partir de la table Etudiant et Enseignant ?

    SELECT nP from Etudiant UNION SELECT nP from Enseignant;
- Restrictions:
  - Les schémas des tables doivent être identiques, c'est-à-dire être composés des mêmes types (le nom des attributs est sans importance, seul le type des attributs est important): les opérations ensemblistes ne garantissent pas la sémantique.
  - Le nom des attributs est imposé par la première requête !
  - INTERSECT et UNION sont commutatifs,
  - MINUS n'est pas commutatif
- UNION: l'union de deux ensembles suppriment les doubles!
  - UNION ALL: idem UNION mais en conservant les doublons.

Le produit cartésien en SQL

- Le produit cartésien en algèbre : $R \times S$
  - C'est l'ensemble des couples (x,y) / $x \in R$ et $y \in S$,
  - En SQL, il suffit de travailler sur deux tables (sans plus):
    SQL1: SELECT * FROM R, S;
    SQL2: SELECT * FROM R CROSS JOIN S;
  - L'ordre des requêtes influe sur l'ordonnancement des attributs dans la table résultats.

SELECT * FROM Personne, PersonnePrénoms;

nP	Nom	Adr	num	Prénom
1	Dupont	Grenoble	1	Jean
1	Dupont	Grenoble	1	Chantal
1	Dupont	Grenoble	2	André
1	Dupont	Grenoble	3	René
2	Aubry	Dijon	1	Jean
2	Aubry	Dijon	1	Chantal
2	Aubry	Dijon	2	André
2	Aubry	Dijon	3	René
3	Duval	Grenoble	1	Jean

etc…

- SI la clause WHERE est utilisé, on parle alors de produit cartésien restreint. Il est évident qu'il sera possible d'écrire une jointure en SQL à l'aide de la clause WHERE !

La jointure en SQL

- La θ-jointure en algèbre : $\mathbf{R} \bowtie_{\mathbf{A1=B1 \ldots \wedge An=Bn}} \mathbf{S}$

Personne

nP	Nom	Adr
1	Dupont	Grenoble
2	Aubry	Dijon
3	Duval	Grenoble

PersonnePrénoms

num	Prénom
1	Jean
1	Chantal
2	André
3	René

On désire fournir l'identité complète des personnes (nom, prénom et prénoms).
Pour cela il faut réaliser la jointure entre Personne et PersonnePrénoms.

SELECT * FROM Personne, PersonnePrénoms WHERE Np=num;

nP	Nom	Adr	num	Prénom
1	Dupont	Grenoble	1	Jean
1	Dupont	Grenoble	1	Chantal
2	Aubry	Dijon	2	André
3	Duval	Grenoble	3	René

La jointure en SQL

- La θ-jointure en algèbre : $R \bowtie_{A1=B1 \ldots \wedge An=Bn} S$
  - Notation SQL89 (declarative, liste de prédicats),

    SELECT * FROM R, S WHERE R.A1 = S.B1 … AND R.An = S.Bn;
    - Avec A1, …, An, B1, … Bn les attributs de R et S qui définissent le critère de jointure,
    - on parle d'inéqui-jointure si l'opérateur de comparaison dans le critère de jointure est différent de l'égalité (<>, >, <, etc.).
    - la clause WHERE peut contenir d'autres prédicats servant de sélection et non plus de critère de jointure.

  - Notation SQL92 (jointure explicite),

    SELECT * FROM R JOIN S ON R.A1 = S.B1 … AND R.An = S.Bn;
    - une clause WHERE peut être ajoutée pour définir des critères de sélection.

  - Remarquez la notation de chemin: **NomTable.NomAttribut**

La jointure en SQL

- Comment résoudre les ambiguités dans la référence des attributs:

  SELECT * FROM Personne, PersonnePrénoms WHERE Np=num;

  - On utilise le plus souvent possible une notation de chemin:

  SELECT * FROM Personne, PersonnePrénoms WHERE Personne.Np=PersonnePrénoms.num;

- On utilise les Alias pour éviter des noms de tables (trop) long et les ambiguïtés:
  - ✓ Un alias représente une instance de tuple en SQL

  SELECT * FROM Personne P, PersonnePrénoms PP WHERE P.Np=PP.num;

La jointure en SQL

- Application à l'autojointure:
  - On est dans le cas où les deux tables et leurs attributs ont même nom !
  - Exemple, Donner le numéro et nom des personnes homonymes (même nom),

    SELECT P1.nP,P1.nom FROM Personne P1, Personne P2
    WHERE P1.nom=P2.nom AND P1.nP<>P2.nP

    – on n'aurait pas pu écrire,

    SELECT nP,nom FROM Personne, Personne
    WHERE nom=nom AND nP<>nP

    Ne veut rien dire car ambigüe, il faut travailler sur deux instances de tuples afin de pouvoir comparer les tuples !
  - L'écriture au format SQL92 est du même genre:

    SELECT P1.nP,P1.nom FROM Personne P1 JOIN Personne P2 ON (P1.nom=P2.nom and P1.nP<>P2.nP)

- Il est possible de réaliser plus d'une jointure:
  - SELECT * FROM R1, R2,… Rn WHERE …
  - SELECT * FROM R1 JOIN R2 ON … JOIN Rn ON …

La jointure naturelle en SQL

- Les attributs mettant en relation deux tables ont le même nom (et le même type):
  - en SQL89, on retrouve la même forme d'expression du critère de jointure,

  SELECT * FROM R, S WHERE R.A1 = S.A1 … AND R.An = S.An;
    - **pour n'utiliser qu'un sous-ensemble des attributs communs des deux tables, il suffit d'écrire les égalités désirées dans la clause WHERE.**
  - en SQL92, la notation est plus explicite,

  SELECT * FROM R NATURAL JOIN S;
    - pour n'utiliser qu'un sous-ensemble des attributs communs des deux tables, la clause USING(A1,… An) permet d'expliciter la restriction.

  SELECT * FROM R NATURAL JOIN S USING (Ai);
- "Donner le nom et l'adresse des étudiants":

  SELECT P.nom, P.adr FROM Personne P NATURAL JOIN Etudiant;
  - Noter la notation de chemin dans la clause SELECT !

Requête imbriquée

- Sous-requête monoligne
  - la sous-requête renvoie un seul tuple (ou une seule valeur),
  - on peut lier cette sous requête à la requête principale en comparant les valeurs du tuble courant de la requête principale avec le tuple résultat de la requête interne (les opérateurs =,<>,>,<,>=,<=).
    La requête externe se comporte comme une boucle (parcourt tous les tuples), la requête interne comme un appel de fonction.
- Sous-requête multilignes
  - la sous-requête renvoie un ensemble de tuples (cas le plus fréquent),
  - on doit utiliser des opérateurs capable de tester un tuple/valeur de la requête par rapport à un ensemble de tuples/valeurs (requête interne): IN, ANY, ALL, EXISTS (utilisable avec NOT)
- Sous-requête synchronisée
  - la sous-requête référence les tuples de la requête externe, elle n'est pas indépendante. Ce type de requête très utilisée permet de traduire les questions les plus complexes.
  - Tous les opérateurs peuvent être utilisés.

Bloc emboîtée monoligne

- Dans le cas d'une sous-requête monoligne, on parle de bloc emboîtée monoligne:
  SELECT A1…,An FROM R WHERE Ai op (SELECT Bj FROM S WHERE p)
  Externe **Interne**
  qui peut s'écrire sous une forme non procédurale:
  SELECT R.A1…,R.An FROM R, S
  WHERE R.Ai op S.Bj AND p
  op={=,<>,>,<,>=,<=} et p un prédicat de filtrage quelconque.
- Exemple: "Donner le nom et l'adresse de l'étudiant N°500"
  SELECT nom, adr FROM Personne
  WHERE nP = (SELECT nP FROM Etudiant WHERE nE=500)
  ou bien
  SELECT P.nom, P.adr FROM Personne P, Etudiant E
  WHERE P.nP=E.nP AND E.nE=500
- Lire la requête: "Pour chaque tuple de la table personne, on teste si la valeur de nP est égale au numéro de personne de l'étudiant dont le numéro est 500."
- Attention à toujours vérifier que votre sous-requête ne puisse renvoyer qu'un seul tuple, sinon ERREUR !
  SELECT nom, adr FROM Personne
  WHERE nP = (SELECT nP FROM Etudiant) *Ne marche pas, pourquoi ?*
- *Vérifier aussi la sémantique du résultat de la sous-requête: Ne pas comparer l'âge du capitaine avec sa pointure de chaussure !!!*
- ***Cette écriture fait apparaître une autre manière pour réaliser une jointure !***

Bloc emboîtée multilignes: IN

- Dans le cas d'une sous-requête multiligne, on parle de bloc emboîtée multiligne. L'opérateur le plus utilisé est le test d'appartenance à un ensemble **IN**:

  SELECT A1…,An FROM R WHERE Ai IN (SELECT Bj FROM S WHERE p)

  qui peut s'écrire sous une forme non procédurale:

  SELECT R.A1…,R.An FROM R, S
  WHERE R.Ai = S.Bj AND p

  p un prédicat de sélection quelconque.

- Exemple: "Donner le nom et l'adresse des étudiants nés avant 1980"

  SELECT nom, adr FROM Personne
  WHERE nP IN (SELECT nP FROM Etudiant WHERE EXTRACT(YEAR FROM dateN)<1980)

  ou bien

  SELECT P.nom, P.adr FROM Personne P, Etudiant E
  WHERE P.nP=E.nP AND EXTRACT(YEAR FROM dateN)<1980

Lire la requête: "pour chaque tuple de la table personne, on teste si la valeur de nP est dans l'ensemble des numéros de personne des étudiants nés avant 1980."

- ***Cette écriture fait apparaître une autre manière générique pour réaliser une jointure !***

L'opérateur IN traduit une SEMI-JOINTURE en SQL!

Bloc emboîtée multilignes: IN

- Considérons maintenant la requête: "Donner le nom des étudiants dont le prénom n'est pas Simon"
  - On peut retranscrire directement cette requête: chercher les personnes dont le prénom n'est pas Simon

```
SELECT nom FROM Personne
WHERE nP IN (SELECT num FROM PersonnePrenoms WHERE prenom<>'Simon')
```

ou bien

```
SELECT P.nom FROM Personne P, PersonnePrenoms PP
WHERE P.nP=PP.num AND PP.prenom<>'Simon'
```

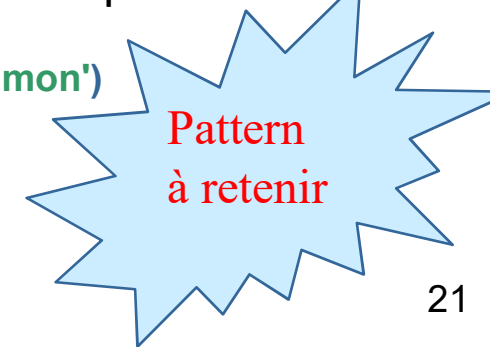
Ces requêtes SQL ne répondent pas à la question: il suffit qu'un étudiant avec le prénom Simon possède d'autres prénoms pour faire partie de l'ensemble résultat (dés qu'un tuple de la table prénom correspond au prédicat, il est retenu).

- "Donner le nom des étudiants dont le prénom n'est pas Simon"
  - On pense à l'envers: Quelles sont les personnes qui ne font pas partie de l'ensemble des personnes dont un prénom est Simon. C'est plus simple à écrire !

```
SELECT nom FROM Personne
WHERE nP NOT IN (SELECT num FROM PersonnePrenoms WHERE prenom='Simon')
```

Cette fois la requête est correcte.

  - Comment l'écrire avec l'opérateur ensemble MINUS ?
  - **<u>Avez-vous reconnu l'application de l'ANTI-JOINTURE</u>**



Bloc emboîtée multilignes: ALL et ANY

- ANY: renvoie vrai si la comparaison d'un élément à un ensemble est vérifiée pour au moins un élément de l'ensemble:
  - =ANY "au moins égal à un élément" ≡ IN
  - <ANY "inférieur à au moins un des élément"
  - >ANY "supérieur à au moins un des éléments"
  - Exemple: "Quels sont les étudiants qui ne sont pas les plus vieux"

  SELECT nE FROM Etudiant
  WHERE dateN >ANY (SELECT dateN FROM Etudiant)
  Pas d'ambiguïté ici car la résolution de nom est local à chaque requête.

- ALL: renvoie vrai si la comparaison d'un élément à un ensemble est vérifiée pour tous les éléments de l'ensemble:
  - >ALL "supérieur au maximum"
  - <ALL "inférieur au minimum"
  - Exemple: "Quels sont les étudiants les plus vieux"

  SELECT nE FROM Etudiant
  WHERE dateN <=ALL (SELECT dateN FROM Etudiant)

Bloc emboîtée non indépendant

- Considérons la requête déjà étudiée: "Donner le nom des étudiants dont le prénom n'est pas Simon"

```
SELECT nom FROM Personne
WHERE nP NOT IN (SELECT num FROM PersonnePrenoms WHERE prenom='Simon')
```

  - La requête interne est indépendante: elle peut s'exécuter sans la requête externe,
  - L'optimiseur du SGBD peut déjà calculer la requête interne, puis utiliser le résultat pour chaque tuple de la requête externe (chaque itération du test **nP IN**)
    - on peut voir la requête externe comme une boucle qui parcourt tous les éléments de la table Personne,
    - Pour chaque itération de la boucle on réalise le test **nP IN**…
    - La requête interne ressemble alors à une constante précalculée !

Bloc emboîtée non indépendant

- Considérons la même requête mais écrite de manière différente (synchronisée):
  - Pour chaque personne, on retient son nom si le prénom Simon n'est pas dans l'ensemble des prénoms de cette personne.

```
SELECT nom FROM Personne P
WHERE 'Simon' NOT IN (SELECT prenom FROM PersonnePrenoms PP WHERE PP.num=P.nP)
```

  - La requête externe parcourt les tuples de Personne les uns après les autres et le prédicat teste l'appartenance de 'Simon' à l'ensemble résultat de la requête interne,
  - La requête interne construit le liste des prénoms de la personne étudiée dans la boucle externe:
    - on a donc une sorte de pointeur sur un tuple de la requête externe,
    - grâce à ce pointeur, l'expression P.nP prend un sens dans la requête interne: c'est la valeur de l'attribut nP du tuple pointé dans la requête externe,
    - on obtient ainsi une construction "similaire" à une boucle imbriquée: for i.. for j.. T[i,j]...
  - Une requête synchronisée est parfois obligatoire, sinon on évitera cette construction (couteuse),

Bloc emboîtée utilisant EXISTS

- **EXISTS(E)** est un opérateur booléen qui renvoie VRAI si l'ensemble E n'est pas vide.
  - cet opérateur apparaît dans une clause WHERE,
  - l'ensemble E est construit par une sous-requête,
  - la requête interne est généralement synchronisée,
  - il est souvent utilisé en négation d'existence (**NOT EXISTS**).
- Exemple: "Donner le numéro des étudiants n'ayant obtenu aucun cours"

```
SELECT nE FROM Etudiant E
WHERE NOT EXISTS (SELECT * FROM Obtenu O WHERE E.nE=O.nE)
```

- Composition de requête: "Donner le nom des étudiants n'ayant obtenu aucun cours"

```
SELECT nom FROM Personne
WHERE nP IN (SELECT nP FROM Etudiant E
                    WHERE NOT EXISTS (SELECT * FROM Obtenu O WHERE E.nE=O.nE))
```

- **Attention construction souvent couteuse: remplacer par un pattern de jointure externe !**

Opérateurs d'agrégation

- Attributs calculés simples:
  - le calcul est réalisé pour chaque tuple, le nombre de tuples résultats n'est pas modifié avec ou sans colonne calculée

SELECT np+1000, upper(Nom) FROM Personne;

- Opération d'agrégation: Impossible en algèbre !
  - le calcul porte sur les valeurs d'un ensemble de tuples
  - Opérateurs sur attributs numériques
    - SUM: somme des valeurs des tuples sélectionnés
    - AVG: moyenne
    - MIN: minimum
    - MAX: maximum
  - Opérateur de comptage, COUNT: nombre de tuples sélectionnés

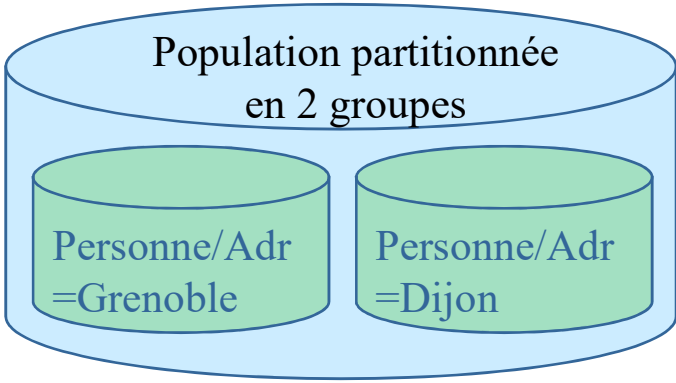
SELECT COUNT(*), COUNT(Adr), COUNT(DISTINCT Adr), MAX(Np) FROM Personne;

COUNT(*)	COUNT(Adr)	COUNT(DISTINCT Adr)	MAX(Np)
3	3	2	3

- Attention à l'usage de la clause DISTINCT !

Partition: GROUP BY

- Une partition est un sous-ensemble de tuples qui partage la même valeur pour un ou plusieurs attributs.
  - GROUP BY est une clause qui construit des groupes/partitions en fonction d'un critère de partitionnement.
  - GROUP BY est suivi du nom des colonnes dont les valeurs définissent les partitions.



"Pour chaque ville, donner le nombre de personne."

SELECT Adr, COUNT(*) AS NbP FROM Personne **GROUP BY Adr**;

Adr	NbP
Grenoble	2
Dijon	1

  - SELECT ne contient que des attributs apparaissant dans le GROUP BY et des expressions agrégées:
    - GROUP BY n'ordonnance pas mais agrège les tuples d'une partition, on a pas de GROUP BY sans agrégation.
    - Les attributs ne servant pas au partitionnement n'ont pas de sens dans l'ensemble résultat.

Restriction dans les partitions

- Filtrer les tuples d'une partition:
  - on utilise la clause WHERE dans son rôle de filtrage,

SELECT Adr, COUNT(*) AS NbP FROM Personne WHERE Np>1 GROUP BY Adr;

Adr	NbP
Grenoble	1
Dijon	1

- Filtrer les partitions:
  - on utilise la clause HAVING dans un rôle de filtrage des valeurs agrégés des partitions

SELECT Adr, COUNT(*) AS NbP FROM Personne **GROUP BY** Adr **HAVING** count(*)>1;

Adr	NbP
Grenoble	2

La division en SQL (1)

- Soit la requête: "Quels sont les étudiants qui sont inscrits à tous les cours de cycle 3 ?"
  - Pas d'opérateur de division en SQL,
  - Il faut réécrire la division **Inscrit** ÷ $\pi_{nomC}(\sigma_{cycle=3}$**(Cours))**
- Solution 1: Il faut que l'ensemble des cours en cycle 3 auquel on soustrait l'ensemble des cours de cycle 3 suivis par un étudiant donne un ensemble vide pour que cet étudiant fasse partie du résultat de la division !
  - on parcourt donc la table des inscrits et pour chacun,
  - on teste si l'ensemble défini précédemment existe ou non, s'il n'existe pas alors le prédicat doit être vrai et l'étudiant faire partie de l'ensemble réponse.

```
SELECT nE FROM Inscrit I1
WHERE NOT EXISTS (SELECT nomC FROM COURS WHERE cycle=3
                  MINUS
                  SELECT nomC FROM Inscrit I2,COURS C WHERE
I1.nE=I2.nE and I2.nomC=C.nomC and C.cycle=3)
```

  - On utilise deux requêtes internes dont une est synchronisée.

La division en SQL (2)

- Solution 2: Il faut que le nombre de cours en cycle 3 soit égal au nombre de cours de cycle 3 auquel est inscrit un étudiant !
  - on parcourt donc la table des inscrits en comptabilisant leur nombre de cours de cycle et on compare ce nombre... (requête externe)
  - … au nombre de cours de cycle 3 (requête interne).

```
SELECT nE FROM Inscrit I,cours C
Where I.nomC=C.nomC and C.cycle=3
GROUP BY nE
HAVING COUNT(I.nomC)=(    SELECT COUNT(nomC)
                          FROM COURS WHERE cycle=3)
```

  - On utilise un partitionnement afin de pouvoir comparer une opération d'agrégation avec le résultat d'une requête, cette requête compte simplement le nombre de cours de cycle 3.
  - Dans ce genre de requête, Faire très ATTENTION à l'importance des doublons et donc à l'utilisation (ou non) de **DISTINCT**.

Patterns de requêtes

- Citation multiple de la même relation
- Maximum, calculé & relatif
- Division
  - ✓ Division interne
  - ✓ Division externe

Citation multiple de la même relation (AutoJointure)

R

A	B	C
1	2	3
1	2	5
2	3	5
2	2	5
4	2	5

Quels couples <A1,A2> tels que R(A1, B1,C1) et R(A2,B2,C2) et B1<B2 ?

Select (distinct) R1.A, R2.A
From R R1 join R R2 on (R1.B < R2.B)

R1.A	R2.A
1	2
2	2
2	4

Citation multiple de la même relation

R

A	B	C
1	2	3
1	2	5
2	3	5
2	2	5
4	2	5

Quels couples <A1,A2> tels que R(A1, B1,C1) et R(A2,B2,C2) et B1≠B2 ?

Select (distinct) R1.A, R2.A
From R R1 join R R2 on (R1.B <> R2.B)

R1.A	R2.A
1	2
2	1
2	2
2	4
4	2

Pour éviter les doublons (x,y) (y,x):
Select (distinct) R1.A, R2.A
From R R1 join R R2 on (R1.B < R2.B)

Citation multiple de la même relation

Exemples : qui gagne plus que qui ? PERSONNE(nom, sal)

Select (distinct) R1.nom, R2.nom
From PERSONNE R1 join PERSONNE R2 on (R1.sal > R2.sal)

Et ça marche aussi avec l'algèbre:
R1:=Personne
R2:=Personne
$\pi_{R1.nom,\ R2.nom}(R1 \bowtie_{(R1.sal>R2.sal)} R2$

Mais souvent on aimerait trouver le plus âgé…

Pattern Maximum (ou minimum)

R

A	B	C
1	2	3
1	2	5
2	3	5
2	2	5
4	2	5

Quels A1 tels que R(A1, B1,C1)
et quelque soit R(A2,B2,C2)
avec A1≠A2 et C1 ≥ C2?

Select A (-> distinct A si A n'est pas la clé)
From R
Where C = (select max(C) from R)

Select A (-> distinct A si A n'est pas la clé)
From R join (select max(C) as C from R))

Maximum

R

A	B	C
1	2	3
1	2	5
2	3	5
2	2	5
4	2	5

Quels A1 tels que R(A1, B1,C1)
et quelque soit R(A2,B2,C2)
avec A1≠A2 et C1 ≥ C2?

Ça marche avec l'algèbre:
On raisonne sur la négation des ensembles
R1:=R
R2:=R
R3:= $\pi_{R1.A}$(R1 $\bowtie_{R1.C<R2.C}$ R2)
R[A] – R3[A]
OU R ▷ R3

Select A From R (On évite une agrégation)
MINUS
Select A From R R1 join R R2 on (R1.C<R2.C)

Exemples : qui gagne le plus ? PERSONNE(nom, sal)
qui est le plus âgé ? PERSONNE(nom, âge)

Qui gagne le plus ? (pattern max)

```
Select nom
From PERSONNE
Where sal = (select max(sal) from PERSONNE)
```

Qui gagne le plus ? (pattern max)

```
Select P1.age
From PERSONNE P1 join (select max(age) age from PERSONNE))
```

Maximum calculé

R

A	B	C
1	2	3
1	2	5
2	3	5
2	2	5
4	2	5

Quels A tels que la somme des B est la plus élevée?
-> il faut calculer la somme des B pour chaque A

```
Select A, sum(B) SB
From R
group by A
```

-> le problème est alors identique au maximum

Maximum calculé

R

A	B	C
1	2	3
1	2	5
2	3	5
2	2	5
4	2	5

```
Select A
From      (Select A, sum(B) SB
           From R group by A)
Where SB =          (select max(sum(B)
                    From R group by A))
```

```
Select A
From      (Select A, sum(B) SB
           From R group by A))
Natural join (select max(sum(B)) SB
                    from R
                    group by A)
```

Et si on veut éviter jointure et sous-requête,
Le pattern Group BY + Having est pratique et esthétique:

```
Select A
From R
group by A
Having sum(B) = (select max(sum(B))
                    from R
                    group by A)
```

Quelle mention a le plus de lauréat?

EXAM(NUME, TOTALNOTES, OPTION1, OPTION2)

On considère TotalPoint = (totalnotes+option1+option2)

On connaît la relation suivante pour repérer les mentions:

MENTION (valeur, minpoint, maxpoint)

Et on peut donc associer une mention à un étudiant:

```
Select nume, valeur
From exam join mention
On (totalnotes+option1+option2>=minpoint
And totalnotes+option1+option2 <= maxpoint)
```

Quelle mention a le plus de lauréat?

```
Select valeur
From exam join mention
On (totalnotes+option1+option2>=minpoint
And totalnotes+option1option2 <= maxpoint)
Group by valeur
Having count(nume) =                    ou count(*)
        (select max(count(nume))
        From exam join mention
        On (totalnotes+option1+option2>=minpoint
        And totalnotes+option1option2 <= maxpoint)
        Group by valeur
        )
```

Maximum relatif

R

A	B	C
1	2	3
1	**2**	5
2	**3**	5
2	**2**	5
4	**2**	5

Pour chaque A, quel est le B pour qui C est le maximum

-> il faut raisonner pour chaque A

Trouver le maximum de C

```
Select A, max(C)
From R
group by A
```

Retrouver le B associé …

```
Select A, B
From R
Natural join
    (Select A, max(C) as C
    From R
    group by A)
```

ou

```
Select A, B
From R
Where (A,C) in
            (select A, max(C)
            from R
            group by A)
```

Quel étudiant (A) a la meilleure note (C) par mention (B) ?

```
Select nume, valeur
From exam join mention
On (totalnotes+option1+option2>=minpoint
And totalnotes+option1+option2 <= maxpoint)
Where (nume, totalnotes+option1+option2 )
IN      (select nume, max(totalnotes+option1option2)
        From exam
        group by nume)
```

```
Select A, B
From R
Where (A,C) in
            (select A, max(C)
            from R
            group by A)
```

Division

R3

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

Quels sont les A que je trouve associés à tous les B ?

$R \div R[B]$

R3

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

Résultat attendu

R3/R3[B]

A
1
2

Résultat calculé

1	2
1	3
2	3
2	2
4	3

1

2

Division interne

On compte les valeurs distincts de B
Si une valeur de A est liée à autant de valeurs de B : c'est ok

```
Select A
From R3
Group by A
Having count(distinct B) =              (select count(distinct B)   from R3)
```

Ou

```
Select A
from
(Select A, count(distinct B) totB From R3 Group by A)
join
(select count(distinct B) MtotB from R3)
On (totB=MtotB)
```

Ou

```
Select A
from
(Select A, count(distinct B) totB From R3 Group by A)
Natural join
(select count(distinct B) totB from R3)
```

Quel film present dans tous les cinémas ?

Cinema(idcinema, idfilm, nbentrees, …)

$\Pi_{idfilm,idcinema}$ (cinema) ÷ $\pi_{idcinema}$(Cinema)

Rôle A

Rôle B

Select idfilm
From Cinema
Group by idfilm
Having count(distinct idcinema) =
(select count(distinct idcinema from Cinema)

Division externe

R

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

S

B
2
1

Quels sont les éléments de A qui apparaissent dans R avec tous les éléments de B dans S ?

Visuellement : c'est 1

Il faut écrire une requête sql dont le résultat est

A
1

En algèbre: $R \div \pi_B(S)$

S

B
3
1

La réponse : relation de schéma A sans n-uplet

Division externe
En SQL

```
Select A
From R natural join S
Group by A
Having count(distinct B) =      (select count(distinct B)
                                from S)
```

On évite la jointure entre R et S si $\pi_B(S) \subset \pi_B(R)$

```
Select A
From R
Group by A
Having count(distinct B) =      (select count(distinct B)
                                from S)
```

R

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

S

B
2
1

Résultat attendu

R ÷ S

A
1

R ⋈ S

Résultat calculé

A	B
1	2
1	1
2	2
4	2

1	2
1	1

2	2

4	2

→ 1

Quels acteurs tournent dans tous les films?

Film(idfilm, titre, réalisateur, …)

Tourne(idfilm, acteur)

```
Select acteur
From Tourne natural join Film
Group by acteur
Having count(distinct idFillm) =   (select count(distinct idFilm)
                                    from Film)
```

```
Select acteur
From Tourne (on peut éviter la jointure ici)
Group by acteur
Having count(distinct idFilm) =   (select count(distinct idFilm)
                                   from Film)
```

Les valeurs NULL
Et
La jointure externe
En SQL

Valeurs nulles

EXAM(NUME, TOTALNOTES, OPTION1, OPTION2)

Total sur 360 points (18 coefficients)

Les options se rajoutent (points au dessus de 10) :

dans la table sont mis les pts au dessus de 10

NumLyceen	TotalNotes	Option1	option2
10101010	210	1	10
12121212	250	3	null
13131313	158	null	6
14141414	177	null	4
15151515	280	8	null

Le domaine de tout attribut peut inclure une valeur NULL

La valeur NULL peut être insérée dans tout domaine, mais elle n'appartient à aucun domaine.

```
Select *
From EXAM
```

NUME	TOTALNOTES	OPTION1	OPTION2
10101010	210	1	10
12121212	250	3	∅
13131313	158	∅	6
14141414	177	∅	4
15151515	280	8	∅

```
select nume, totalnotes+option1+option2
 From exam
```

NUME	TOTALNOTES+OPTION1+OPTION2
10101010	221
12121212	∅
13131313	∅
14141414	∅
15151515	∅

∅ Représente la (non) valeur NULL

```
Select nume, totalnotes+option1+option2
From exam
Where totalnotes+option1+option2> 180
```

NUME	TOTALNOTES+OPTION1+OPTION2
10101010	221

Normalement on attendrait:

NUME	TOTALNOTES+NVL(OPTION1,0)+NVL(OPTION2,0)
10101010	221
12121212	253
14141414	181
15151515	288

Pour détecter les valeurs NULL

Is NULL

Is Not NULL

```
Select nume
From exam
Where option1 is NULL
```

```
NUME
----------
  13131313
  14141414
```

```
Select nume
From exam
Where option2 is not NULL
```

```
 NUME
----------
  10101010
  13131313
  14141414
```

Utilisation des valeurs nulles: NVL

Pour intégrer les valeurs NULL : cas de calculs arithmétiques
NVL avec valeur par défaut (élément neutre)

```
Select nume,
totalnotes+ option1+option2
From exam
```

NUME	TOTALNOTES+OPTION1+OPTION2
10101010	221
12121212	∅
13131313	∅
14141414	∅
15151515	∅

```
Select nume,
totalnotes+ nvl(option1,0)+Nvl(option2,0)
From exam
```

NUME	TOTALNOTES+NVL(OPTION1,0)+NVL(OPTION2,0)
10101010	221
12121212	253
13131313	164
14141414	181
15151515	288

```
Select nume
From exam
Where totalnotes+nvl(option1,0)+nvl(option2,0)> 180
```

```
NUMLYCEEN
----------
 10101010
 12121212
 14141414
 15151515
```

Résultats complets:

```
Select numlyceen,totalnotes+nvl(option1,0)+nvl(option2,0)
From bac
Where totalnotes+nvl(option1,0)+nvl(option2,0)>180
```

NUMLYCEEN	TOTALNOTES+NVL(OPTION1,0)+NVL(OPTION2,0)
10101010	221
12121212	253
14141414	181
15151515	288

Pour intégrer les valeurs NULL dans les jointures: jointure externe

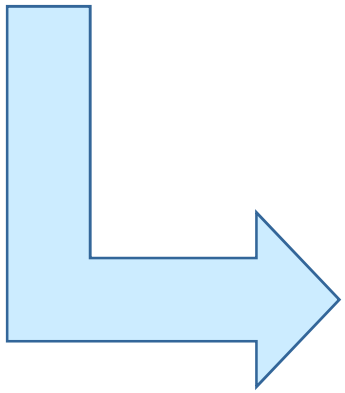
```
Select *
From R join S on (R.X1 = S.X1)
```

R

X1	Y
1	a
1	b
2	c

S

X1	Z
1	x
1	x
2	x
4	y



X1	Y	Z
1	a	x
1	b	x
2	c	x

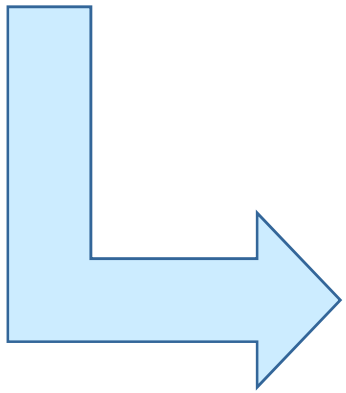
Select *
From R **right outer** join S
on R.X1 = S.X1

R

X1	Y
1	a
1	b
2	c

S

X1	Z
1	x
1	x
2	x
4	y



X1	Y	Z
1	a	x
1	b	x
2	c	x
4	∅	y

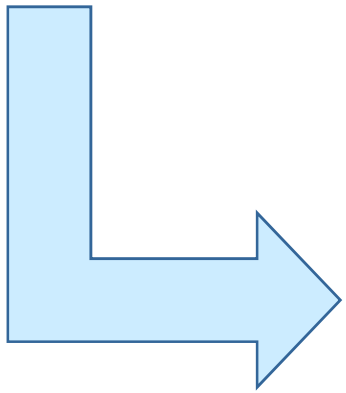
Select *
From R **left outer** join S
on R.X1 = S.X1

R

X1	Y
1	a
1	b
2	c

S

X1	Z
1	x
1	x
2	x
4	y



X1	Y	Z
1	a	x
1	b	x
2	c	x

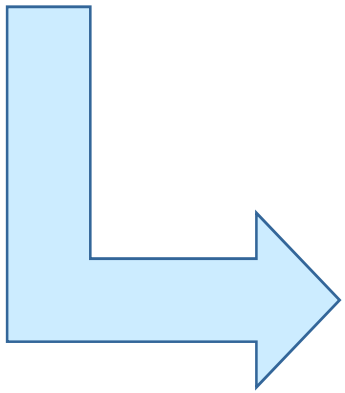
Select *
From R **left outer** join S
on R.X1 = S.X1

R

X1	Y
1	a
1	b
2	c
3	a

S

X1	Z
1	x
1	x
2	x



X1	Y	Z
1	a	x
1	b	x
2	c	x
3	a	∅

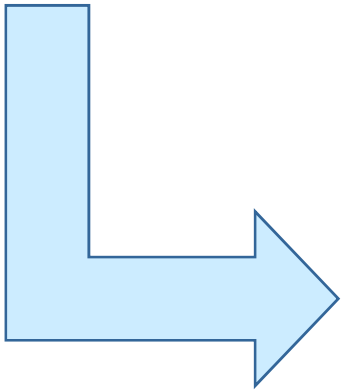
Select *
From R **full outer** join S
on R.X1 = S.X1

R

X1	Y
1	b
1	a
2	c
3	a

S

X1	Z
1	x
1	x
2	x
4	y



X1	Y	Z
1	a	x
1	b	x
2	c	x
3	a	∅
4	∅	y

Table mention

VALEUR	MINPOINT	MAXPOINT
passable	180	215
assezbien	216	251
bien	252	287
tresbien	288	380

nume	TotalNotes	Option1	option2
10101010	210	1	10
12121212	250	3	null
13131313	158	null	6
14141414	177	null	4
15151515	280	8	null

```
Select nume,valeur
From exam join mention
On (totalnotes+nvl(option1,0)+nvl(option2,0)>=minpoint
And totalnotes+nvl(option1,0)+nvl(option2,0) <= maxpoint) ;
```

NUME	VALEUR
14141414	passable
10101010	assezbien
12121212	bien
15151515	tresbien

Il manque ceux qui ont échoué (pas de mention) …

```
Select nume, nvl(valeur,'ajourne')
From exam left outer join mention
On totalnotes+nvl(option1,0)+nvl(option2,0)>=minpoint
And totalnotes+nvl(option1,0)+nvl(option2,0) <= maxpoint
Order by numlyceen;
```

NUME	NVL(VALEUR
10101010	assezbien
12121212	bien
13131313	ajourne
14141414	passable
15151515	tresbien

La modification de tuples en SQL

- La modification des valeurs d'un ou plusieurs tuple(s) est possible avec l'instruction UPDATE:

  UPDATE nomTable

  SET nomAttribut1=expression, ...

  SET nomAttribut2=expression

  WHERE ...

  - nomTable est la table dont le contenu doit être mis à jour,
  - nomAttributi sont les attributs dont les valeurs doivent être modifiées. L'expression peut être de toute sorte et même le résultat d'une requête SQL !
  - la clause WHERE permet de filtrer les tuples qui seront modifiés.

- Exemple: "Modifier l'adresse des personnes dont la valeur est 'Genobl' par 'Grenoble' (Erreur de saisie !).

  UPDATE Personne SET adr='Grenoble' WHERE adr='Genobl';

  - Ce type de requête n'a pas de résultat, il faut une requête d'interrogation pour vérifier les modifications,

  SELECT * FROM Personne; (*par exemple*)

La suppression de tuples en SQL

- La suppression d'un ou plusieurs tuple(s) est possible avec l'instruction DELETE:

  DELETE FROM nomTable

  WHERE predicat

  - nomTable est la table dont le contenu doit être mis à jour,
  - la clause WHERE permet de filtrer les tuples qui seront supprimées.

- Exemple: "Supprimer les étudiants né avant le 1/01/1990" (Erreur de saisie !).

  DELETE FROM Etudiant WHERE dateN<TO_DATE('1/01/1990');

  - Ce type de requête n'a pas de résultat, il faut une requête d'interrogation pour vérifier les modifications,

  SELECT * FROM Etudiant; (*par exemple*)

SQL
pour la création de schéma

- Partie LDD (Langage définition de données)

Création de schéma: CREATE TABLE

- La commande CREATE TABLE permet de définir en 1 commande une relation avec ses attributs, leurs domaines, les identifiants primaires et externes, un certain nombre de contraintes simples.

```
CREATE TABLE  nomTable (
    colonne1 type1 [DEFAULT valeur1] [NOT NULL],
    colonne2 type2 [DEFAULT valeur2] [NOT NULL],
    [CONSTRAINT nomContrainte1 typeContrainte1 paramContrainte1] … );
```

- NomTable: le nom de la relation comportant des lettres, des chiffres et les symboles classiques (_,$,#,etc.). **Attention, Oracle est insensible à la casse (comme beaucoup de SGBD) !**
- Colonnei: le nom d'un attribut de la relation,
- Typei: un type (Oracle) qui définit le domaine d'un attribut,
- ***Le caractère ; termine toute commande Oracle.***

Types (Oracle) disponibles

- Type chaîne de caractères
  - de longueur fixe : **`CHAR(longueur)`**
    Par défaut la valeur est d'1 caractère et au maximum de 2000 caractères.
  - de longueur variable : **`VARCHAR2(longueur)`**
    Par défaut 1 caractère et au maximum 4000 caractères. Le stockage dépend de la taille réelle de la chaîne.
    La norme définit le type ***char(longueur)****.*
  - de flot de caractères: **`CLOB(longueur) jusqu'à 4Go.`**
  - de caractères Unicode: **`NCHAR(l), NVARCHAR2(l), NCLOB(l)`**

Types (Oracle) disponibles

- Type numérique décimale : **`NUMBER(précision, échelle)`**
  - `précision` : nombre entier de chiffres significatifs (1 à 38).
  - `échelle` : nombre de chiffre à droite de la virgule (-84 à 127). Une valeur négative arrondit le nombre.

  ex: VAL NUMBER(8,2) définit la donnée VAL avec 8 chiffres significatif dont deux après la virgule (-999999,99 à +999999,99)
  VAL NUMBER(8,-2) arrondit la donnée à la centaine.
  - *La norme définit les types **decimal(precision, echelle)** et **integer**.*

Types (Oracle) disponibles

- Type date/heure:
  - **`DATE`**: le format standard d'une date qui permet de stocker le siècle, l'année, le mois, le jour, l'heure, les minutes et les secondes. Le format par défaut dépend de la langue (le stockage est indépendant).
  - **`TIMESTAMP`**(précision): date et heure incluant des fractions de secondes (précision de 0 à 9).
  - **`INTERVAL YEAR`** (précision) **`TO MONTH`**: permet d'extraire une différence entre deux moments avec une précision mois/année (précision)
  - **`INTERVAL DAY`** (précision1) **`TO SECOND`** (précision2): différence plus précise entre deux moments (précision1 pour les jours et précision2 pour les fractions de secondes).
- Type binaires:
  - **`BLOB`**: données non structurées jusqu'à 4Go.
  - **`BFILE`**: données stockées dans un fichier externe à la base.

CREATE TABLE (exemple)

```
/* Description des Personnes */

CREATE TABLE Personne (nP CHAR(10), nom VACHAR2(15), adr VARCHAR2(50));

/* Description des prénoms
     d'une personne */

CREATE TABLE PersonnePrénoms(
     num CHAR(10),
     prénom VARCHAR2(15) );

-- Description des étudiants

-- qui sont par ailleurs des personnes.

CREATE TABLE Etudiant(nP CHAR(10), nE CHAR(10), dateN DATE);

/* Description des études réalisées par un étudiant */

CREATE TABLE EtudiantEtudes(nE CHAR(10), année NUMBER(1), diplôme
     VARCHAR2(30));

-- Fin du script SQL pour Oracle.
```

Gestion des contraintes

- Les contraintes sont déclarées de deux manières:
  - **mode *inline***: c'est-à-dire en même temps que l'attribut qu'elle contraint (contrainte monocolonne).
  - **mode *out-of-line***: la définition apparaît à la suite des définitions d'attributs, introduit avec le mot clé CONSTRAINT. Dans ce cas la contrainte peut être nommée (plus flexible) et peut porter sur plusieurs attributs (ou colonnes).
- On distingue 4 types de contraintes:
  - **UNIQUE** : impose une valeur distinct d'un attribut pour chacun des tuples.
  - **PRIMARY KEY** : déclare l'identifiant de la relation (ou clé de la table), la contrainte UNIQUE et l'option NOT NULL est sous-entendu pour les attributs concernés.
  - **FOREIGN KEY** : déclare un identifiant externe (ou clé étrangère) ainsi que la méthode employée pour assurer l'intégrité référentielle.
  - **CHECK** : permet de spécifier une condition simple que doit suivre le ou les attributs.

Contrainte PRIMARY KEY

```
/* Description des Personnes */

CREATE TABLE Personne (
    nP CHAR(10) PRIMARY KEY, -- contrainte inline
    nom VACHAR2(15) NOT NULL,
    adr VARCHAR2(50));

/* Description des prénoms
    d'une personne */

CREATE TABLE PersonnePrénoms(
    num CHAR(10),
    prénom VARCHAR2(15),
    -- contrainte out-of-line nommée
    CONSTRAINT pk_PersonnePrénoms PRIMARY KEY (num,prénom) );
```

- Dans la première table on peut utiliser une contrainte *inline* où *out-of-line*.
- Dans la seconde table, la contrainte *out-of-line* est obligatoire: elle porte sur deux colonnes, l'identifiant étant composé de deux attributs.

Contrainte Foreign Key

- Cette contrainte est le noyau de la gestion de cohérence d'une base de données relationnelle (qui reposent justement sur la mise en relation de données): l'intégrité référentielle.
  - basée sur une relation entre identifiant (clé primaire/candidate) et identifiant externe (clé étrangère): "Un étudiant est une personne qui doit exister dans la base".
  - on parle souvent de tables:
    - **père**: cette table possède la clé primaire référencée (ex: Personne).
    - **fils**: cette table possède une clé étrangère qui référence la clé primaire/candidate d'une table père (ex: Etudiant).
- Les problèmes résolus/évités par un SGBD en activant cette cohérence:
  - **fils vers père**: impossible d'ajouter un tuple fils qui ne puisse pas se rattacher à un tuple père (ex: pas d'étudiant sans être une personne),
  - **père vers fils**: impossible de supprimer un tuple père ayant encore des tuples fils qui lui sont rattachés (ex: plus d'étudiants si plus de personnes).

Contrainte Foreign Key (suite)

- Il faut donc gérer cette contrainte qui assure l'intégrité référentielle du coté père et du coté fils:
  - du coté père: on s'assure d'avoir une clé primaire (PRIMARY KEY) pour pouvoir référencer les tuples de la relation père.
  - du coté fils: on définit la clé étrangère à l'aide d'une contrainte FOREIGN KEY (qui précise le ou les attributs du fils qui référencent la clé primaire/candidate du père).
- CONSTRAINT contrainte FOREIGN KEY(f_attr1, f_attr2) REFERENCES p_table(p_attr1,p_attr2)
  - contrainte: nom de la contrainte pour faciliter sa gestion,
  - f_attr_i: nom des attributs de la table fils composant la clé étrangère,
  - p_table: nom de la table père référencée,
  - p_attr_i: nom des attributs de la table père p_table composant sa clé primaire/candidate qui sont référencés par f_attr_i.

Contrainte Foreign Key (exemple)

```
/* Description des Personnes: table père */

CREATE TABLE Personne (
    nP CHAR(10) PRIMARY KEY, -- définition de l'identifiant chez le père
    nom VACHAR2(15) NOT NULL,
    adr VARCHAR2(50));

/* Description des Etudiants: table fils */

CREATE TABLE Etudiant(nE CHAR(10) PRIMARY KEY,
    nP CHAR(10),
    dateN DATE,
    -- définition de l'identifiant externe
    CONSTRAINT fk_Etudiant_Personne FOREIGN KEY(nP)
        REFERENCES Personne(nP) );
```

Contrainte Foreign Key (gestion de la cohérence père -> fils)

- La suppression d'un tuple du père donnent trois alternatives:
  - on interdit la suppression de ce tuple s'il est encore référencé par des tuples des fils: il faut supprimer tous les tuples fils avant. On ne doit pas survivre à ces enfants!
  - on accepte la suppression de ce tuple mais cette suppression est automatiquement propager aux tuples des fils. Ainsi la cohérence est respectée. La clause optionnelle ON DELETE CASCADE doit être ajoutée.
  - on accepte cette suppression de tuple mais sans propagation aux tuples des fils. Les composants de la clé étrangère des tuples des fils prennent la valeur NULL. La clause optionnelle ON DELETE SET NULL doit être ajoutée (et ne pas contredire les contraintes NOT NULL).
- Exemple:

CREATE TABLE Etudiant(nE CHAR(10) PRIMARY KEY,
nP CHAR(10),DateN DATE,
-- définition de l'identifiant externe
CONSTRAINT fk_Etudiant_Personne FOREIGN KEY(nP)
REFERENCES Personne(nP) ***ON DELETE CASCADE****);*

Contrainte Check(condition)

- Cette contrainte contrôle l'insertion ou la modification d'un tuple en fonction d'une condition explicite:
  - impose un domaine de valeurs (restriction plus précise qu'un typage),
  - Les contraintes plus complexes se programment via des "déclencheurs" .
- Exemple:

CREATE TABLE Etudiant(nE CHAR(10) PRIMARY KEY, nP CHAR(10), DateN DATE, CONSTRAINT ck_DateN CHECK ((SYSDATE-DateN)/365>18));

- La condition accepte divers opérateurs:
  - de comparaison (>,=,<,>=,<=,<>)
  - logiques (NOT, AND, OR)
  - mathématiques (+,-,*,/)
  - des fonctions (cf. fin de ce cours)
  - des opérateurs intégrés (BETWEEN, IN, LIKE, IS NULL)

Contrainte CHECK (Suite)

- Les opérateurs intégrés (comme le format de condition) sont très utilisés en SQL (en mode interrogation).

CREATE TABLE Personne (
nP CHAR(10) ***PRIMARY KEY****,*
nom VACHAR2(15),
adr VARCHAR2(50)

- Opérateur BETWEEN … AND … teste l'appartenance à un intervalle de valeurs, renvoie un booléen (VRAI/FAUX).

CONSTRAINT ck_nP1 CHECK (nP BETWEEN 1 and 10),

- Opérateur IN (…) teste l'appartenance à un ensemble de valeurs.

CONSTRAINT ck_nP2 CHECK (nP IN (1,2,3,4,5,6,7,8,9,10)),

- Opérateur LIKE … compare une expression à une chaîne de caractère type (% pour un ou plusieurs caractères, _ pour un seul caractère).

CONSTRAINT ck_adr CHECK (adr LIKE '% Grenoble %'),

- Opérateur IS NULL teste si une expression correspond à la valeur NULL.

CONSTRAINT ck_nom CHECK (nom NOT IS NULL));

Insertion de tuples en SQL

- La commande INSERT INTO permet d'insérer ligne par ligne de nouveaux tuples:

  INSERT INTO table VALUES …

  - table: le nom de la table où insérer un tuple.
  - on peut renseigner ensuite toutes les valeurs d'un tuple:

    INSERT INTO Personne VALUES (1,'Dupont', 'Grenoble');
  - ou bien ne renseigner que certaines valeurs:

    INSERT INTO Personne(nP, nom) VALUES (2,'Aubry');

    La valeur de l'attribut Adr est alors implicitement NULL.

- Problème avec la date:
  - les formats '01/09/04', '01-09-04' et '01-09-2004' sont équivalents
  - sinon il faut utiliser la fonction TO_DATE(texte,format) qui transforme une chaîne de caractères en une date en fonction d'une expression de formatage:

    INSERT INTO Etudiant VALUES (1,1001,'22-01-71');

    ou INSERT INTO Etudiant VALUES(1,1001,TO_DATE('22 Janvier 71','DD MONTH YY');

Supprimer des tables

- La commande DROP (pour supprimer une table)
  - DROP TABLE nom_table [CASCADE CONSTRAINTS];
  - CASCADE CONSTRAINTS
    - Supprime toutes les contraintes référençant une clé primaire (primary key) ou une clé unique (UNIQUE) (permet de supprimer une table sans être gêné par ses fils)
    - Si on cherche à détruire une table dont certains attributs sont référencés sans spécifier CASCADE CONSTRAINT, Oracle retourne une erreur.
  - Avant de créer un schéma il est toujours préférable d'avoir un script qui le détruit (pour éviter les erreurs de cohérence):
    - La destruction suit un chemin précis: d'abord les fils, ensuite les relations pères (inverse de la création).

DROP TABLE EtudiantEtudes;

DROP TABLE Etudiant;

DROP TABLE PersonnePrénoms

DROP TABLE Personne;