le cnam

Optimisation de Bases de Données

Travaux Dirigés

CNAM Paris
nicolas.travers (at) cnam.fr

Chapitre

Table des matières

Chapitre 1

Stockage dans un SGBD

Le schéma de base de données suivant, ainsi que ses statistiques, seront utilisés dans l'ensemble des exercices du cours.

- **Schéma de la base de données :**
  Cours (Id_Cours, Id_Enseignant, Intitule)
  Personne (Id_Personne, Nom, Prenom, Annee_Naissance)
  Salle (Id_Salle, Localisation, Capacite)
  Reservation (Id_Salle, Id_Cours, Date, heure_debut, heure_fin)
- **Taille des différents types :**
  - Les identifiants (Id_...) et les nombres (ECTS, Capacite) sont codés sur 4 octets,
  - les dates, années et heures sur 4 octets,
  - Nom, Prenom, Intitule, Localisation sur 50 octets
- **Statistiques :**
  - 2 200 tuples dans la table `Cours`,
  - 124 000 tuples `Personne` (Contenant enseignants et auditeurs),
  - 500 tuples `Salle` (Avec une capacite de 20 à 600),
  - 514 000 tuples `Reservation`.
- **Configuration Base de données :**
  - Taille d'un ROWID : 10 octets
  - 11 pages en mémoire centrale,
  - une page fait 8 ko
  - un PCTFREE de 10% (utilisées pour les mises à jour)

1.1 Stockage

1.1.1 Calculer la taille d'une page utile (DataBlock), sachant que l'entête d'une page est de 192 octets

1.1.2 Calculer en nombre de pages, la taille de chaque table en nombre de pages (proposer le résultat sous forme de tableaux)

1.1.3 Aucun index n'est encore créé. Combien coûte la requête suivante :

```
SELECT Date, heure_debut
FROM Reservation
WHERE Id_Cours = 'NFE106' ;
```

1.1.4 Supposons que la table soit fragmentée sur le disque de manière aléatoire, et un temps de latence de 9 ms en moyenne par secteur (page). Quel temps faut-il pour évaluer la requête précédente ?

1.1.5 Le stockage et le typage des données n'est pas idéale, proposez des améliorations possibles et recalculer les pages [1]

1.1.6 Un index sur l'attribut `Localisation` aiderait à accéder uniquement à la salle concernée. Afin de retrouver le tuple, nous avons besoin d'un ROWID (pointeur logique). Détailler la composition d'un ROWID qui pourrait correspondre (après décompression) à ceci : **3.376.26.0**

1. Pour la suite des exercices, nous n'utiliserons que les résultats obtenus dans la question 2

1.2 B+Tree

1.2.1 Nous souhaitons créer un index B+Tree sur l'attribut `Identifiant` de la table `Salle`. Donner l'index B+Tree d'orbre 2 après insertion des valeurs suivantes. Vous ferez apparaître les étapes d'éclatement :
100, 25, 72, 48, 10, 33, 58, 110, 40, 52, 115, 80, 5, 28, 49, 75

1.2.2 Donner la requête SQL de création de cet index.

1.2.3 Nous créons maintenant un index d'orbre 3 sur l'attribut `Capacité` de la table `Salle`, avec les valeurs suivantes :
20, 30, 40, 20, 25, 200, 300, 150, 40, 20, 20, 50, 30

1.2.4 Lors de la création d'un index, on ne spécifie pas l'ordre de celui-ci. Il est calculé automatiquement en fonction de la taille de l'attribut indexé, du ROWID associé à chaque valeur, et de la taille d'une page (idem que pour les données). Calculer l'ordre de l'index sur l'attribut `Capacité`.

1.2.5 Au vu de l'ordre d'un BTree, on peut estimer la hauteur de cet index (permet d'estimer son coût de parcours). Donner la hauteur de l'index sur la `Capacité`.

1.3 Index dense et non-dense

1.3.1 Quelle est la hauteur maximum d'un index dense sur l'attribut `id_Personne` de la table `Personne` ?

1.3.2 Quelle est la hauteur maximum d'un index non-dense sur l'attribut `id_Personne` de la table `Personne` ?

1.3.3 Donner la requête SQL (sous `Oracle`) pour la création de la table `Personne` en index non-dense

1.3.4 Supposons que la table Personne est non-dense sur *id_personne*. On souhaite maintenant ajouter un index sur l'attribut `Nom`, peut-il être un index non-dense ?

Chapitre 2

Interrogation des indexes

Informations sur la Base de Données

- Page disque : 8ko
- Taille d'un ROWID : 10o
- PCTFree : 10% (page utile 7 200o)
- $|\mathcal{M}| = 11$
- Clustering Factor, par défaut : dense ($\phi = \|R\|$), non-dense ($\phi = |R|$)
- Schéma de la base :

Table	Attributs	Taille	Nb tuples	Nb tuples/pages
Cours	(Id_Cours, Id_Enseignant, Intitule)	20 pages	2 200	112
Personne	(Id_Personne, Nom, Prenom, Annee_Naissance)	2 000 pages	124 000	62
Salle	(Id_Salle, Localisation, Capacite)	5 pages	500	112
Reservation	(Id_Salle, Id_Cours, Date, heure_debut, heure_fin)	2 000 pages	514 000	285

2.1 Requêtes mono-attributs

2.1.1 Pour la requête suivante, donner le coût d'évaluation en séquentiel, index dense, index non-dense :

```
SELECT Nom, Prenom FROM Personne WHERE Id_Personne = 125 ;
```

2.1.2 Idem pour la requête suivante, sachant la répartition des années de naissance dans l'histogramme 2.1.2. Index sur l'année de naissance :

```
SELECT Nom, Prenom FROM Personne WHERE Annee_Naissance = 1979 ;
```

Année	1978	1979	1980	1981	1982	1983	1984	...
Pourcentage	2,04%	3,226%	3,2%	2,12%	2,63%	2,48%	1,99%	...

Histogramme de répartition des années de naissance

2.1.3 Pour la requête précédente, on créer un index sur `Annee_Naissance` et couvrant sur `Nom, Prenom`. Calculer l'ordre de l'index (`Nom et Prenom sont stockés dans l'index`), sa hauteur max et le coût de la requête.

2.2 Requête multi-attribut

2.2.1 Donner le nombre de tuples résultats pour la requête suivante, sachant qu'il y a 5 000 prénoms différents et 3.226% des personnes sont nées en 1979 :

```
SELECT Nom, Prenom FROM Personne
WHERE Annee_Naissance = 1979 AND Prenom = 'Nicolas' ;
```

2.2.2 Donner le coût d'évaluation pour la requête précédente avec :

- En séquentiel,
- Index dense sur Annee_Naissance,
- Index non-dense sur Annee_Naissance,
- Index dense sur Annee_Naissance et couvrant sur Nom, puis Prénom (les 3 attributs sont dans l'index [1]),
- Index dense sur le Prénom, avec un *clustering factor*, $\phi = 100000$.

2.2.3 Peut-on combiner les deux indexes denses sur Annee_Naissance et Prenom, avant d'accéder aux données ? Calculer le coût d'exécution (deux parcours, un accès avec sélectivités combinées, avec $\phi = \|Personne\|$)

2.2.4 Donner le coût d'un index dense sur Annee_Naissance et couvrant sur Prénom, puis Nom.

1. La recherche se fait d'abord sur l'année, puis sur le nom, et enfin sur le prénom

2.3 Seuil de sélectivité

Pour aller plus loin, nous allons calculer dans quel cas un BTree est intéressant

2.3.1 Avec les formules du cours, un BTree est plus intéressant qu'un parcours séquentiel dans le cas suivant :

$$|I| + \Delta + \phi \times Sel < |R|$$

Développer le calcul suivi dans le cas d'un BTree dense dans le pire des cas.

2.3.2 Simplifier le calcul pour isoler la sélectivité (nous pourrons arrondir quelques valeurs)

2.3.3 Si l'on considère que : $|R| \sim \frac{\|R\|}{\frac{7200}{|tuple|}}$. Simplifier la formule.

2.3.4 Si on prend $Sel = \frac{1}{Card}$, simplifier.

Chapitre 3

Indexes Bitmap et Hachage

Informations sur la Base de Données

- Page disque : 8ko
- Taille d'un ROWID : 10o
- PCTFree : 10% (page utile 7 200o)
- $|\mathcal{M}| = 11$
- Clustering Factor, par défaut : dense ($\phi = \|R\|$), non-dense ($\phi = |R|$)
- Schéma de la base :

Table	Attributs	Taille	Nb tuples	Nb tuples/pages
Cours	(Id_Cours, Id_Enseignant, Intitule)	20 pages	2 200	112
Personne	(Id_Personne, Nom, Prenom, Annee_Naissance)	2 000 pages	124 000	62
Salle	(Id_Salle, Localisation, Capacite)	5 pages	500	112
Reservation	(Id_Salle, Id_Cours, Date, heure_debut, heure_fin)	2 000 pages	514 000	285

3.1 Bitmap

3.1.1 Création du bitmap

On souhaite créer un index bitmap sur les années, réparties de 1970 à 1992.

3.1.1.1 Quel est la taille en nombre de pages d'un vecteur Année ? (PCTFREE=0%, uniquement l'entête)

3.1.1.2 Quel est la taille de l'index ?

3.1.2 Interrogation du Bitmap

Pour chaque requête, donner le coût avec un index dense sur l'Année (3,226% en 1979) et avec le Bitmap sur l'année.

3.1.2.1

```
SELECT COUNT(*) From Personne Where Annee = 1979 ;
```

3.1.2.2 La sélectivité de < 1980 est égale à 43% des tuples

```
SELECT COUNT(*) From Personne Where Annee < 1980 ;
```

3.1.2.3

```
SELECT Annee, COUNT(*) From Personne Group By Annee ;
```

3.1.2.4 Donner la requête SQL pour créer un index Bitmap sur `Personne.Annee`.

3.2 Tables de hachage

Nous utiliserons deux fonctions de hachage différentes :

- $H_1 = x \mod n$
- $H_2 = \frac{x \times n}{N}$

x est la valeur que l'on souhaite insérer, compris entre 0 et N-1.
H est la valeur hachée de x, compris entre 0 et n-1 (cette valeur correspond à l'indice dans la table de hachage).

3.2.1 Création de la table de Hachage

On souhaite créer une table de hachage (ou table partitionnée) sur la table `Reservation` sur `Id_Cours` (permet de regrouper les séances prévues pour un même cours dans une même partition).

3.2.1.1 Créez un exemple de table de hachage de taille $n = 10$, et insérer les `id_cours` suivants en utilisant H_1 puis H_2 (avec N = 100) : 1, 12, 3, 15, 30, 42, 15, 60, 31, 12, 58, 78, 16, 15, 16

3.2.1.2 Au vu des statistiques de la base de données, et en supposant une répartition uniforme des cours :

(a) Combien de fois un identifiant de cours apparaît-il dans cette table ?

(b) Combien de fois un identifiant de salle apparaît-il ?

(c) Combien de d'identifiants *id_cours* différents par partition pour une table de hachage de taille 500 ?

3.2.1.3 Quelle est la taille d'une partition pour une table de hachage de taille 500 ?

3.2.1.4 Quelle est la taille de la table partitionnée ?

3.2.1.5 Donner la requête SQL de création de la table `Reservation` en table partitionnée par hachage (500 partitions).

3.2.1.6 Est-il possible de créer une index non-dense et une table de hachage sur la même table ?

3.2.2 Interrogation de la table de hachage

3.2.2.1 Quel est le coût d'évaluation pour la requête suivante (BTree dense / non-dense de hauteur 3, Hachage) :

```
SELECT Id_Salle, Date, heure_debut FROM Reservation WHERE Id_Cours = 300 ;
```

3.2.2.2 BTree non-dense et Hachage semblent similaires en temps de réponse. En serait-il de même si nous avions indexé/haché `Id_Salle` ?

Chapitre 4

Jointures

Pour chacun des algorithmes suivants, donner son coût d'évaluation et les conditions d'utilisation (si nécessaires), en tenant compte de :

- Nombre de page $|R|$ et $|S|$;
- Nombre de tuples $\|R\|$ et $\|S\|$;
- Taille des index $|I_R|$ ou $|I_S|$ et sa sélectivité sel ;
- Nombre de pages en mémoire centrale $|M|$;
- Optimisation possible avec une structure ou un index.

4.1 Algorithmes de Jointures

4.1.1 Jointure par boucles imbriquées

4.1.2 Jointure par boucles imbriquées avec Index

4.1.3 Jointure par Tri-Fusion

4.1.4 Jointure par Hachage en mémoire

4.1.5 Jointure par Hachage sur le disque

4.2 Evaluation d'une requête

4.2.1 Donner le coût d'évaluation de la requête suivante avec les différents algorithmes de jointures. Vous devrez calculer les tailles de résultats intermédiaires (avec PCTFREE 0%).

```
SELECT NOM, Prenom FROM Personne P, Cours C WHERE P.Id_Personne = C.Id_Enseignant;
```

Chapitre 5

Optimisation et EXPLAIN

Informations sur la Base de Données

- Page disque : 8ko
- Taille d'un ROWID : 10o
- PCTFree : 10% (page utile 7 200o)
- $|\mathcal{M}| = 11$
- Clustering Factor, par défaut : dense ($\phi = \|R\|$), non-dense ($\phi = |R|$)
- Schéma de la base :

Table	Attributs	Taille	Nb tuples	Nb tuples/pages
Cours	(Id_Cours, Id_Enseignant, Intitule)	20 pages	2 200	112
Personne	(Id_Personne, Nom, Prenom, Annee_Naissance)	2 000 pages	124 000	62
Salle	(Id_Salle, Localisation, Capacite)	5 pages	500	112
Reservation	(Id_Salle, Id_Cours, Date, heure_debut, heure_fin)	2 000 pages	514 000	285

5.1 Optimisation

Pour les requêtes suivantes, proposer un algorithme de jointure optimisé (pas nécessairement optimal). Vous utiliserez les indexes et les organisations physiques ci-dessous. Donner le plan EXPLAIN correspondant.

- BTree :

Index	Hauteur	ordre	Type
Cours.Id_Cours	2	200	dense unique
Cours.Intitule	2	56	dense
Personne.Id_Personne	3	200	dense unique
Salle.Id_Salle	2	200	dense unique
Salle.Capacite	2	200	dense
Reservation.PK	2	94	non-dense unique

- Bitmap :

Index	Taille vecteur	Nb vecteurs
Personne.Annee	2	23

5.1.1 Seulement 10 cours commencent par "bases de données".

```
SELECT Nom, Prenom
FROM Personne P, Cours C
WHERE
        P.Id_Personne = C.Id_Enseignant AND
        Intitule like 'bases de Données%' ;
```

5.1.2

```
SELECT COUNT(*)
FROM Personne P, Cours C
WHERE
        P.Id_Personne = C.Id_Enseignant AND
        Intitule like 'bases de Données%' ;
```

5.1.3 3,226% sont nés en 1979.

```sql
SELECT COUNT(*)
FROM Personne P, Cours C
WHERE
        P.Id_Personne = C.Id_Enseignant AND
        Annee = 1979 AND
        intitule like 'bases de Données%';
```

5.1.4 Il n'y a qu'une seule salle '35.3.26' et 0,14% des réservations sont dans cette salle.

```sql
SELECT Date, heure_debut, heure_fin
FROM Reservation R, Salle S, Cours C
WHERE
        R.Id_Cours = C.Id_Cours AND
        R.Id_Salle = S.Id_Salle AND
        S.Localisation = '35.3.26' AND
        intitule like 'bases de Données%';
```

5.2 EXPLAIN

Donner la requête SQL correspondante et son coût d'exécution pour chacun des plans EXPLAIN suivants :

5.2.1 0,07% des réservations sont dans la salle 426, et 2% de celles-ci se passent le '2017-10-05'.

```
0_____SELECT STATEMENT
1_____,_____NESTED LOOPS
2*_____,_____,_____INDEX RANGE SCAN_____,_____,_____Reservation
3_____,_____,_____TABLE ACCESS BY ROWID_____,_____,_____Cours
4_____,_____,_____,_____AND EQUAL
5*_____,_____,_____,_____,_____INDEX UNIQUE SCAN_____PK_Cours
6*_____,_____,_____,_____,_____INDEX RANGE SCAN_____Cours_Intitule

(2) ID_Salle = 426 AND Date = '2017-10-05'
(5) R.ID_Cours = C.Id_Cours
(6) Intitule like 'bases de données%'
```

5.2.2 0,07% des réservations sont dans la salle 426, et 2% de celles-ci se passent le '2017-10-05'.

```
0_____SELECT STATEMENT
1*_____,_____SORT UNIQUE
2*_____,_____,_____MERGE JOIN
3_____,_____,_____,_____SORT JOIN
4*_____,_____,_____,_____,_____INDEX RANGE SCAN_____,_____,_____,_____Reservation
5_____,_____,_____,_____SORT JOIN
6_____,_____,_____,_____,_____TABLE ACCESS BY INDEX ROWID_____,_____Cours
7*_____,_____,_____,_____,_____,_____INDEX RANGE SCAN_____,_____,_____Cours_Intitule

(1) DISTINCT (Id_Cours)
(2) R.ID_Cours = C.Id_Cours
(4) ID_Salle = 426 AND Date = '2017-10-05'
(7) Intitule like 'bases de données%'
```

Informations sur la Base de Données

- Page disque : 8ko
- Taille d'un ROWID : 10o
- PCTFree : 10% (page utile 7 200o)
- $|\mathcal{M}| = 11$
- Clustering Factor, par défaut : dense ($\phi = \|R\|$), non-dense ($\phi = |R|$)
- Schéma de la base :

Table	Attributs	Taille	Nb tuples	Nb tuples/pages
Cours	(Id_Cours, Id_Enseignant, Intitule)	20 pages	2 200	112
Personne	(Id_Personne, Nom, Prenom, Annee_Naissance)	2 000 pages	124 000	62
Salle	(Id_Salle, Localisation, Capacite)	5 pages	500	112
Reservation	(Id_Salle, Id_Cours, Date, heure_debut, heure_fin)	2 000 pages	514 000	285

6.1 Choix de schéma

6.1.1 L'attribut `Localisation` n'est pas stocké de manière optimal. En effet, il est composé d'un numéro d'accès, d'un numéro d'étage et d'un numéro de porte. Proposer un meilleur stockage pour la table `Salle`.

6.1.2 Donner le nouveau schéma et la taille de la nouvelle table.

6.2 Oracle

Soit la requête de création de tablespace suivant :

```
CREATE TABLESPACE tabspace2
DATAFILE 'diskC:tabspacefile.dat' SIZE 2G
DEFAULT STORAGE (
        INITIAL 40K NEXT 800K NEXT 1600K
        MINEXTENTS 1 MAXEXTENTS 999
        PCTINCREASE 50
        PCTFREE 90 PCTUSED 70
)
```

6.2.1 Donner un tableau qui donne le nombre de pages contigües pour chaque extension (jusqu'à la 6°)

6.2.2 Donner le nombre d'extensions pour chacune des tables

6.2.3 Compte tenu du schéma de la table Réservation, comment pourrait-on améliorer son stockage ? Donner la requête de création de table, en tenant compte de l'index non-dense.

6.2.4 Donner la nouvelle taille de la table Reservation et le nombre d'extension à partir de votre requête de création.

6.3 DB2

6.3.1 On souhaite utiliser les possibilités de compression sous DB2 pour la table `Personne`. Donner la requête DB2 permettant d'estimer la place éconimisée

6.3.2 Avec un PCTFREE de 0% (192o pour les méta informations) et une moyenne de compression de 50% par tuple, donner la taille estimer en nombre de pages pour la table `Personne`.

6.3.3 Donner la requête de création de table compressée.

6.4 SQL Server

6.4.1 Après analyse des requêtes soumises au SGBD, on se rend compte que la requête type est :

```
SELECT Localisation
FROM Salle S, Reservation R
WHERE Date = '2017-10-05' AND
        S.Id_Salle = R.Id_Salle ;
```

Les index présents sur la table `Reservation` sont inutiles pour cette requête. Dire pourquoi.

6.4.2 Proposer la création d'un index pour améliorer cette requête

6.4.3 Donner la requête de création de table sous `SQL server` pour créer un index non-dense sur l'attribut `Date` de `Reservation`

Chapitre 7

Tuning de requêtes

Informations sur la Base de Données

- Page disque : 8ko
- Taille d'un ROWID : 10o
- PCTFree : 10% (page utile 7 200o)
- $|\mathcal{M}| = 11$
- Clustering Factor, par défaut : dense ($\phi = \|R\|$), non-dense ($\phi = |R|$)
- Schéma de la base :

Table	Attributs	Taille	Nb tuples	Nb tuples/pages
Cours	(Id_Cours, Id_Enseignant, Intitule)	20 pages	2 200	112
Personne	(Id_Personne, Nom, Prenom, Annee_Naissance)	2 000 pages	124 000	62
Salle	(Id_Salle, Localisation, Capacite)	5 pages	500	112
Reservation	(Id_Salle, Id_Cours, Date, heure_debut, heure_fin)	2 000 pages	514 000	285

7.1 Amélioration de requêtes

Pour chacunes des requêtes suivantes, trouver l'erreur et proposer une nouvelle requête plus optimisée :

7.1.1

```
SELECT NB
FROM (
        SELECT Id_Salle, COUNT(*) AS NB
        FROM Reservation
        GROUP BY Id_Salle) R, Salle S
WHERE R.Id_Salle = S.Id_Salle AND
        Localisation = '35.3.26'
```

7.1.2

```
SELECT Id_Salle, Localisation, Capacite, COUNT(*)
FROM Salle S, Reservation R
WHERE R.Id_Salle = S.Id_Salle
GROUP BY Id_Salle, Localisation, Capacite
```

7.1.3

```
SELECT * FROM Personne WHERE Prenom like '%las' ;
```

7.1.4

```
SELECT * FROM Reservation WHERE YEAR(Date) = 2017 AND WEEK(Date) = 8 ;
```

7.2 Indexation de fonction

7.2.1 Dans le chapitre précédent (6.1), nous avions changé le schéma de la table Salle (Id_Cours, Acces, Etage, Porte, Capacite). Toutefois, on souhaiterait continuer à faire des recherches de type Localisation = '35.3.26' (ou équivalent). Proposer une solution.

7.2.2 Il est également possible d'utiliser le *Pipeline* sans passer par un index de fonctions, pour générer cet accès en prenant la localisation en entrée de la procédure *pipeliné*. Proposer une implémentation PL/SQL avec *pipeline* pour intégrer la décomposition de la localisation.

7.3 Vue matérialisée

Le calcul de la requête suivante est trop couteux et effectué régulièrement sur la base de données :

```
SELECT Intitule, Acces, Etage, Porte, Date, Heure_debut, Heure_fin
FROM Cours C, Salle S, Reservation R
WHERE
        C.Id_Cours = R.Id_Cours AND
        R.Id_Salle = S.Id_Salle AND
        C.Id_Cours = XXX ;
```

Le paramètre XXX est changé à chaque fois en fonction du cours demandé.

7.3.1 Comme solution, nous proposons de tout sauvegarder dans une vue matérialisée. Donner la requête de création de la vue `Planning`.

7.3.2 Quel est le ou les attributs qui devront être indexés ?

7.3.3 Donner la taille prise par les données de cette vue. (PCTFREE=0, métainfo 192o)

7.3.4 Quel est le coût d'exécution de la requête suivante :

```
SELECT Acces||'.'||Etage||'.'||Porte as Localisation, Date, Heure_debut, Heure_fin
FROM Planning
WHERE Id_Cours = 25 ;
```

7.4 HINT

Les requêtes suivantes ne produisent pas le plan d'exécution attendu. Utiliser des HINTs qui permettraient de leur donner la bonne orientation.

7.4.1 L'index `BTree_Capacite` n'est pas utilisé car les statistiques ne sont pas à jour :

```
SELECT Date, Heure_debut, Heure_fin
FROM Salle S, Reservation R
WHERE S.Id_Salle = R.Id_Salle AND
        Capacite > 200
```

7.4.2 L'index `bitmap_annee` est utilisée alors qu'il génère un trop grand nombre de pointeurs :

```
SELECT Intitule
FROM Cours C, Personne P
WHERE P.Id_Personne = C.Id_Enseignant AND
        ANNEE_Naissance = 1960 ;
```

7.4.3 Sur la requête précédente, on souhaiterait en plus forcer l'utilisation d'une jointure par boucle imbriquée avec index et également qu'elle soit dirigée par la table `Cours` (en effet, un JoinHash est généré mais ne tient pas en mémoire à cause d'une mauvaise répartition)

Chapitre 8

Dénormalisation

Informations sur la Base de Données

- Page disque : 8ko
- Taille d'un ROWID : 10o
- PCTFree : 10% (page utile 7 200o)
- $|\mathcal{M}| = 11$
- Clustering Factor, par défaut : dense ($\phi = \|R\|$), non-dense ($\phi = |R|$)
- Schéma de la base :

Table	Attributs	Taille	Nb tuples	Nb tuples/pages
Cours	(Id_Cours, Id_Enseignant, Intitule)	20 pages	2 200	112
Personne	(Id_Personne, Nom, Prenom, Annee_Naissance)	2 000 pages	124 000	62
Salle	(Id_Salle, Localisation, Capacite)	5 pages	500	112
Reservation	(Id_Salle, Id_Cours, Date, heure_debut, heure_fin)	2 000 pages	514 000	285

8.1 Normalisation

8.1.1 Soit les dépendances fonctionnelles suivantes :

- Nom → Ville
- (NomClient, PrenomClient) → Nb_places
- date_creation → ID_groupe

Vérifier, étant données ces dépendances, que les tables de ce schéma sont en BCNF. Justifier votre réponse pour chaque forme normale.

Si ces tables ne sont pas en BCNF effectuer les transformations nécessaires pour les normaliser.

8.1.2 couverture minimale

8.2 Dénormalisation

8.2.1 Soit la requête suivante qui compte le nombre de cours par responsable :

```
SELECT Nom, COUNT(*)
FROM Personne P, Cours C
WHERE C.id_enseignant = P.id_Personne
GROUP BY Nom;
```

- Sur le schéma original (avant dénormalisation), calculer le coût de la jointure par boucle imbriquée (sans projections) : Cours ⋈ Personne.

8.2.2 Proposer une restructuration d'*ajout de colonne redondante* et donner le coût.

8.2.3 Soit la requête suivante :

```
SELECT id_Salle, Localisation, Capacite
FROM Reservation R, Salle S
WHERE R.id_Salle = S.id_Salle
AND R.Date = '2016-10-05'
```

La sélectivité de '2016-10-05' est de 0,5836%. Calculer le coût pour une jointure entre Reservation ⋈ Salle avec boucle imbriquée. Attention, il n'y a pas d'index sur Reservation.Date !!

Index Id_salle (non dense) : Hauteur : 1 ordre : 200 Selectivité : 1/500

8.2.4 Proposer une restructuration de *composition* entre salle et Reservation