Université IBM i

7 novembre 2023

IBM Innovation Studio Paris

S13 – Bonnes pratiques pour écrire un code SQL efficient

14:45 / 15:45

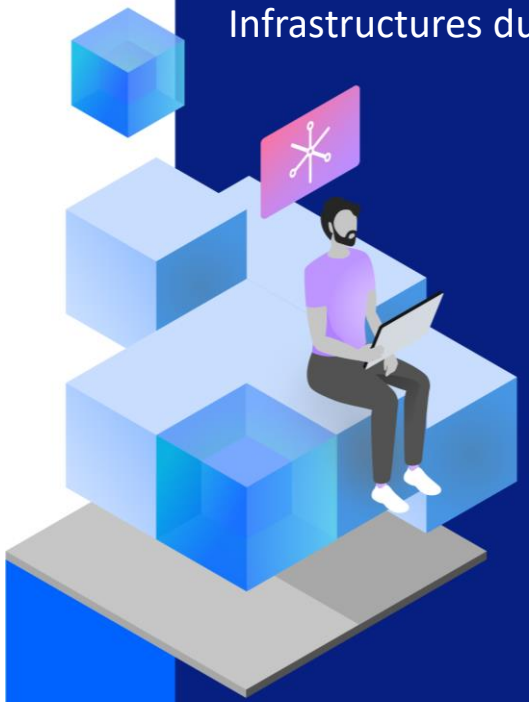
Nathanaël Bonnet
Gaia/Volubis
nathanael.bonnet@gaia.fr

infrasdufutur

#ibmi
#uii2023
#infrastructuredufuturIBM23

IBM

Infrastructures du futur



7 et 8 novembre 2023

Présentation

- Nathanaël BONNET
  - IBM i depuis 1999
  - Expert IBM i
  - Développement & intégration

IBM Champion 5 Year Milestone
Awarded through the program
IBM Champion

IBM Professional Certified Developer
IBM i 7.x

IBM i Community
Advocate Level 3
IBM Power Systems

GAIA
VOTRE EXPERT IBM i

VOLUBIS
DEVENEZ UN EXPERT

common FRANCE
MEMBER OF common EUROPE

Agenda

- 1. Introduction
  - Rappels
  - Périmètre
- 2. Quelques conseils d’écriture
  - Forme
  - Structure
  - Cas des dates
- 3. Quelques conseils de conception
  - Clés : primaires, uniques ou non, étrangères
  - Vues

IBM

Infrastructures du futur

7 et 8 novembre 2023

Université IBM i

7 novembre 2023

IBM i 35 YEARS

Let's Create

1. Introduction

Nous ne verrons pas

- Centre de Santé / Visual Explain / Index Advisor
  - ➔ Cf historique université

- Administration/Exploitation
  - ➔ Voir S07 - Contrôler l'exécution des requêtes SQL avec Query Supervisor et QAQQINI – 13h30 à 14h30

- La mise au point extrême des requêtes, au-delà de l'outillage
  - ➔ Voir S21 - L'hyper optimisation sous DB2 for i – 16h à 17h – Christian Grière

Nous verrons

- Suivant un parti-pris
  - Votre base existe, utilisons la au mieux !
    - Il est possible de casser toute la structure et faire mieux : ce n'est pas notre intention ici
  - Nous expliquons ici les « erreurs » les plus courantes
    - Et facilement réparables

Une « bonne » requête SQL

- Comment définir ?
  - Certains critères objectifs
    - Résultat attendu conforme
    - Performance acceptable
  - D'autres plus subjectifs
    - Lisibilité , maintenance
- En fonction
  - Des attentes & des compétences => adaptez à votre environnement

IBM

Infrastructures du futur

7 et 8 novembre 2023

Université IBM i

7 novembre 2023

IBM i 35 YEARS

Let's Create

2. Ecriture

2 principes fondamentaux

- SQL est un langage **ensembliste**
  - Ne pas reproduire une logique ligne à ligne
    - Type algorithme RPG/COBOL
  - Exprimer tout le traitement en 1 seule requête
    - Permet à SQL d'optimiser ses traitements

- SQL est un langage de **manipulation de données**
  - Distinguer toutes les sélections, jonctions, groupage, … sur la donnée brute
    - Permet à SQL d'optimiser ses traitements
  - Transformer la donnée ensuite
    - Exprimer une valeur date depuis 4 colonnes, formater les décimaux …

- Exemple
  - https://www.gaia.fr/3-bonnes-pratiques-decriture-de-requete-sql/

Outils

- STRSQL
  - Vraiment ?


```
                        Entrée d'instructions SQL

Saisissez l'instruction SQL, puis appuyez sur ENTREE.
===> with x as (
           Select Ordinal_Position as RowKey, Element as RowInfo
             from
     Table(SysTools.Split(Get_Clob_From_File('/home/NB/file.csv'),
     x'0D25')) a
             Where Trim(Element) > ''),
          y as (
           Select x.*, Ordinal_Position ColKey,
                  Trim(B '"' from Element) as ColInfo
             from x cross join Table(SysTools.Split(RowInfo, ';'))
     a)
     Select RowKey,
            Min(Case When ColKey = 1 Then ColInfo End) Code,
            Min(Case When ColKey = 2 Then ColInfo End) Nom,
            Min(Case When ColKey = 3 Then ColInfo End) Prenom
     From y
     Where RowKey > 1
                                                          A suivre...
F14=Supprimer ligne      F15=Scinder ligne      F16=Choisir bibliothèques
F17=Choisir fichiers     F18=Choisir zones      F24=Autres touches
```

Outils

- ACS – Exécution de script SQL
  - Gratuit (licences IBM i Access)

csv.sql

```
1  select * from table(split('code;nom;prenom', ';')) ;
2
3
4
5  -- Read *csv File from IFS
6  With x as (-- Split IFS File into Rows (at CRLF)
7            Select Ordinal_Position as RowKey, Element as RowInfo
8              from Table(SysTools.Split(Get_Clob_From_File('/home/NB/file.csv'), x'0D25')) a
9              Where Trim(Element) > ''),
10      y as (-- Split IFS File Rows into Columns (and remove leading/trailing double quotes ")
11           Select x.*, Ordinal_Position ColKey,
12                  Trim(B '"' from Element) as ColInfo
13             from x cross join Table(SysTools.Split(RowInfo, ';')) a)
14 -- Return the Result as Table
15 Select RowKey,
16        Min(Case When ColKey = 1 Then ColInfo End) Code,
17        Min(Case When ColKey = 2 Then ColInfo End) Nom,
18        Min(Case When ColKey = 3 Then ColInfo End) Prenom
19 From y
20 Where RowKey > 1 -- Remove header
21 Group By RowKey
22 with cs ;
```

Outils - ACS

- Coloration syntaxique
  - Lisibilité
  - Identification visuelle des commentaires, éléments de langages, noms de table/colonne, constantes …

Outils - ACS

- Formatage automatique (personnalisable)

csv.sql | *Untitled 2

```
select * from table(split('code;nom;prenom', ';')) ;

-- Read *csv File from IFS
With x as (-- Split IFS File into Rows (at CRLF)
  Select Ordinal_Position as RowKey, Element as RowInfo from Table(SysTool
Where Trim(Element) > ''),
   y as (-- Split IFS File Rows into Columns (and remove leading/trailing
Select x.*, Ordinal_Position ColKey,
 Trim(B '"' from Element) as ColInfo
from x cross join Table(SysTools.Split(RowInfo, ';')) a)
-- Return the Result as Table
Select RowKey,
Min(Case When ColKey = 1 Then ColInfo End) Code,Min(Case When ColKey = 2 T
From y Where RowKey > 1 -- Remove header
Group By RowKey with cs ;
```

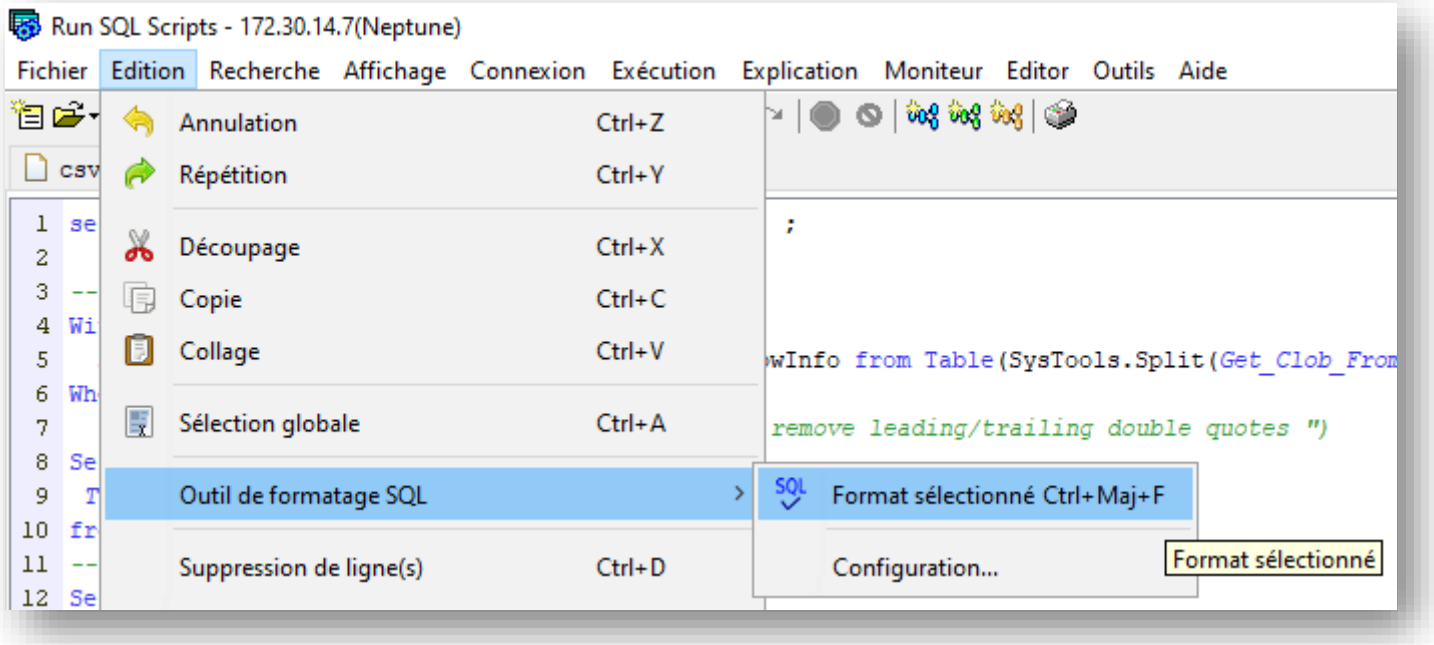
csv.sql | *Untitled 2

```
select * from table(split('code;nom;prenom', ';')) ;

-- Read *csv File from IFS
WITH x AS ( -- Split IFS File into Rows (at CRLF)
        SELECT Ordinal_Position AS RowKey,
               Element AS RowInfo
            FROM TABLE (
                   SysTools.Split(GET_CLOB_FROM_FILE('/home/NB/file.csv'), x'0D25')
                ) a
            WHERE TRIM(Element) > ''
    ),
    y AS ( -- Split IFS File Rows into Columns (and remove leading/trailing double quotes ")
        SELECT x.*,
               Ordinal_Position ColKey,
               TRIM(B '"' FROM Element) AS ColInfo
            FROM x
                CROSS JOIN TABLE (
                        SysTools.Split(RowInfo, ';')
                    ) a
    )
-- Return the Result as Table
    SELECT RowKey,
```

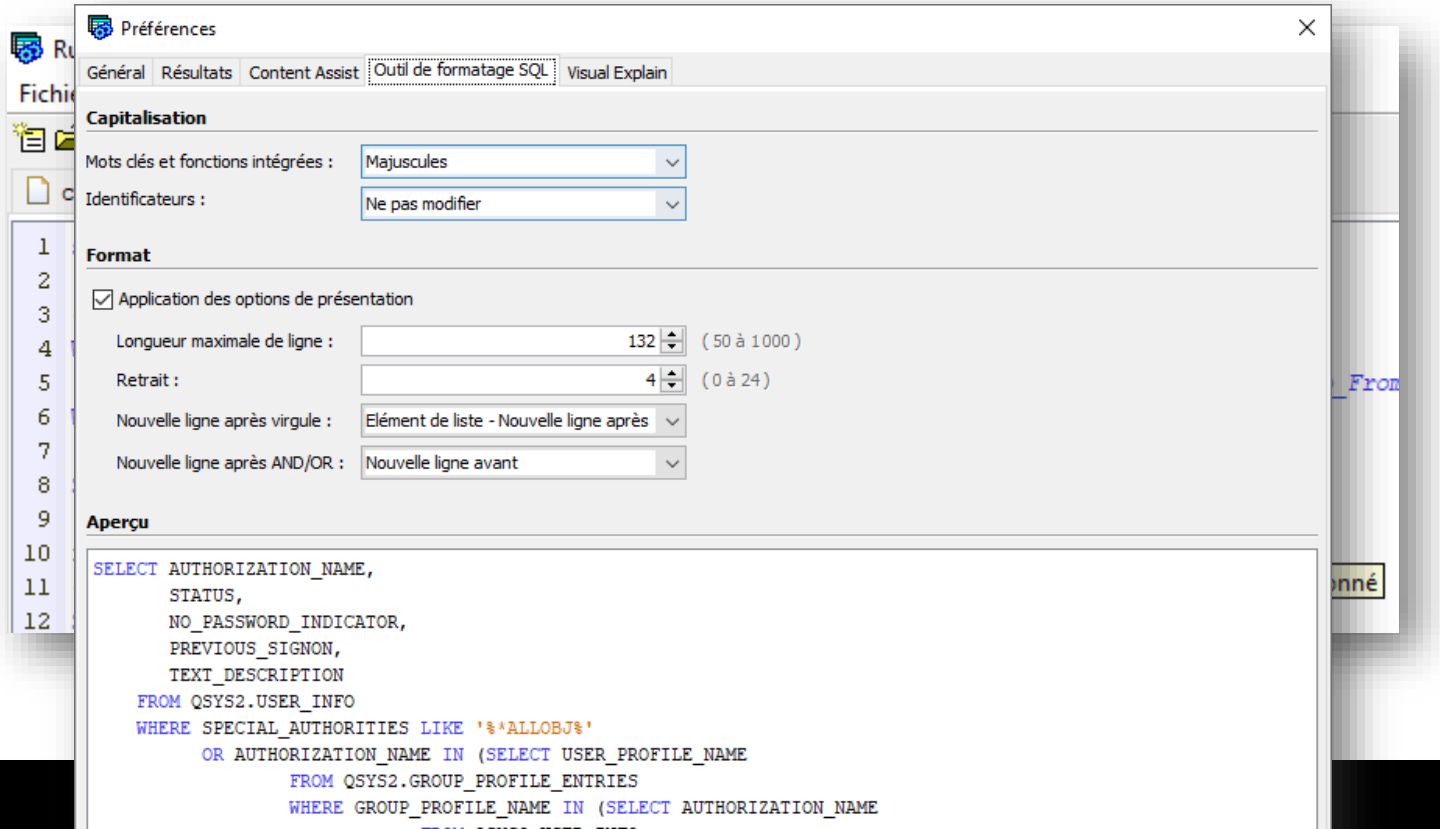
Outils - ACS

- Formatage automatique (personnalisable)



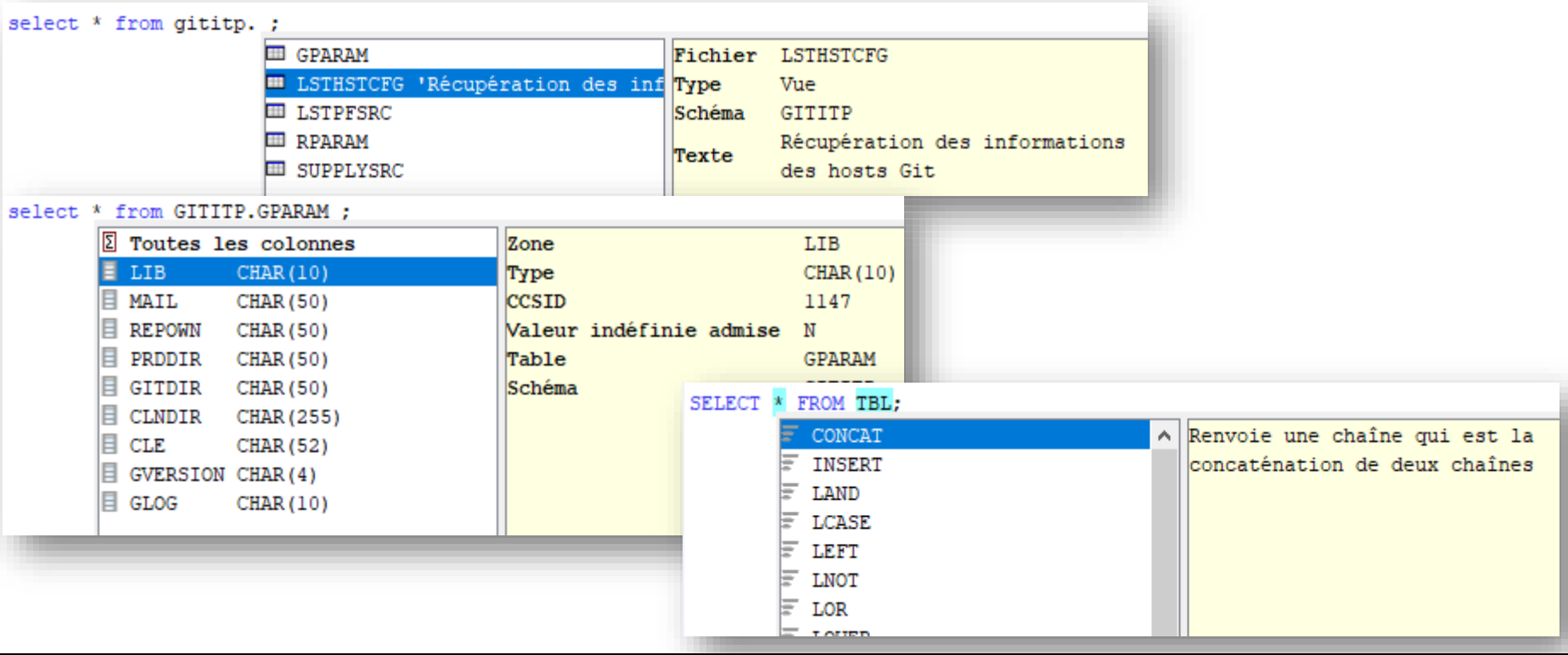
Outils - ACS

- Formatage automatique (personnalisable)



Outils - ACS

- Assistance & complétion (eq. F4 STRSQL)



Outils

- Gestion des scripts (et encodage)
  - Fichier source
  - IFS
  - PC (local ou lecteur réseau)
- Plusieurs scripts ouverts en simultané
- Scripts et result sets accessibles en simultané
- Basculement commentaire/code
- Ctl-Z (!)
- Recherche et remplacement
- Exemples (personnalisables)
- …

- Nombreux outils non disponibles en 5250

Outils

- Mais aussi
  - RDi
  - VSCode
  - Data Studio
  - …

csv.sql

home > NB > sql > csv.sql

```
1  select * from table(split('code;nom;prenom', ';')) ;
2
3
4
5  -- Read *csv File from IFS
6  With x as (-- Split IFS File into Rows (at CRLF)
7              Select Ordinal_Position as RowKey, Element as RowInfo
8                from Table(SysTools.Split(Get_Clob_From_File('/home/NB/file.csv'), x'0D25')) a
9                Where Trim(Element) > ''),
10      y as (-- Split IFS File Rows into Columns (and remove leading/trailing double quotes ")
11             Select x.*, Ordinal_Position ColKey,
12                    Trim(B '"' from Element) as ColInfo
13               from x cross join Table(SysTools.Split(RowInfo, ';')) a)
14 -- Return the Result as Table
15 Select RowKey,
16        Min(Case When ColKey = 1 Then ColInfo End) Code,
17        Min(Case When ColKey = 2 Then ColInfo End) Nom,
18        Min(Case When ColKey = 3 Then ColInfo End) Prenom
19 From y
20 Where RowKey > 1 -- Remove header
21 Group By RowKey
22 with cs ;
```

*csv.sql

Line 19 Column 2

```
-- Read *csv File from
With x as (-- Split IF
            Select Ordi
              from Tabl
              Where Tri
     y as (-- Split IF
            Select x.*,
                   Trim
              from x cr
-- Return the Result a
Select RowKey,
       Min(Case When ColKey = 1 Then ColInfo End) Code,
       Min(Case When ColKey = 2 Then ColInfo End) Nom,
       Min(Case When ColKey = 3 Then ColInfo End) Prenom
From y
Where RowKey > 1 -- Remove header
Group By RowKey
with cs ;
```

Forme

- Important pour faciliter la compréhension !
  - Exemple
    - https://www.sqlstyle.guide/
    - Faites vos propres règles
  - Indentation (cf capacité ACS/RDi)
    - Repérer les débuts/fins
    - Repérer les imbrications
      - Sous-requêtes par exemple
  - Casse
    - Eléments du langage SQL en majuscule (instruction, fonction, mots réservés …)
    - Identificateurs en minuscule (noms de tables, colonnes, …)

Forme

- Nommage & commentaire
  - Choisir des noms significatifs (limite à 128 caractères)
  - Commenter les objectifs, particularités …

- Conseils
  - Ne pas utiliser les mots clés du langage ou registres
    - ACTION, DATE, USER …
    - Cf https://www.ibm.com/docs/en/i/7.5?topic=words-reserved ou https://www.ibm.com/docs/en/i/7.5?topic=elements-special-registers
  - Ne pas utiliser de caractères accentués ou d'espace
    - Possible mais plus complexe (exemple "Date facture")
    - Limite la portabilité
  - Séparer les mots par « _ »
    - « date_inscription » au lieu de « DateInscription »
  - Eviter les abréviations non usuelles
    - « Date_maj », « date_echeance »

SELECT *

- A éviter
  - Indiquer une sélection de colonnes
  - Permet
    - D’éviter un accès au catalogue pour chercher la liste des colonnes
    - De limiter la quantité de données à manipuler à ce qui est nécessaire

```
SELECT *
    FROM GITITP.GPARAM
    WHERE GVERSION = '2.0';

SELECT LIB,
       MAIL,
       CLE,
       GLOG
    FROM GITITP.GPARAM
    WHERE GVERSION = '2.0';
```

ORDER BY

- Avec SQL, le résultat d'une sélection n'a pas d'ordre prévisible
  - Indiquer ORDER BY si l'ordre a de l'importance pour vous
  - Ne rien indiquer sinon

```
SELECT empno,
       firstnme
    FROM employee
    ORDER BY hiredate;
```

IN / LIKE

- Remplacer IN par =

```
SELECT empno,
       firstnme
    FROM employee
    WHERE workdept in ('A00', 'B01')
    ORDER BY hiredate;
```

```
SELECT empno,
       firstnme
    FROM employee
    WHERE workdept = 'A00' OR
          workdept = 'B01'
    ORDER BY hiredate;
```

- Si LIKE est utilisé en commence par, remplacer par LEFT

```
SELECT empno,
       firstnme
    FROM employee
    WHERE FIRSTNME LIKE 'DA%'
    ORDER BY hiredate;
```

```
SELECT empno,
       firstnme
    FROM employee
    WHERE left(FIRSTNME , 2) = 'DA'
    ORDER BY hiredate;
```

Valeurs calculées

- Eviter d'exprimer des critères de sélection / groupage sur des valeurs calculés

```
SELECT empno,
       firstnme,
       hiredate
    FROM employee
    WHERE YEAR(hiredate) BETWEEN 1970 AND 1975
    ORDER BY hiredate ;
```

```
SELECT empno,
       firstnme,
       hiredate
    FROM employee
    WHERE hiredate BETWEEN '1970-01-01' AND '1975-12-31'
    ORDER BY hiredate ;
```

```
SELECT empno,
       firstnme,
       hiredate
    FROM employee
    WHERE year(hiredate) = 1972 and month(hiredate)=02
    ORDER BY hiredate ;
```

```
SELECT empno,
       firstnme,
       hiredate
    FROM employee
    WHERE hiredate BETWEEN '1972-02-01' AND '1972-02-29'
    ORDER BY hiredate ;
```

Dates

- Utiliser les fonctions de calcul !

```
SELECT empno,
       firstnme,
       hiredate
    FROM employee
    WHERE hiredate BETWEEN '1972-02-01' AND '1972-02-29'
    ORDER BY hiredate;
```

```
SELECT empno,
       firstnme,
       hiredate
    FROM employee
    WHERE hiredate BETWEEN '1972-02-01' AND LAST_DAY('1972-02-01')
    ORDER BY hiredate;
```

```
SELECT empno,
       firstnme,
       hiredate
    FROM employee
    WHERE hiredate BETWEEN '1972-02-01' AND (date('1972-02-01') + 1 months - 1 days)
    ORDER BY hiredate;
```

- Permet d'éviter de gérer tous les cas de figure

```
SELECT empno, firstnme, hiredate FROM employee WHERE hiredate BETWEEN '1972-02-01' AND '1972-02-30' ORDER BY hiredate
Etat SQL : 22007
Code fournisseur : -181
Message : [SQL0181] Une valeur de la chaîne date, heure ou horodatage est incorrecte. Cause . . . . . : La représentation
caractères) incorrecte, soit la variable hôte ou la colonne qui contient cette chaîne. Si le nom est *N, la valeur a été
```

Dates / heures / horodatages

- De nombreuses fonctions sur les dates !
  - ADD_DAYS, ADD_HOURS, ADD_MINUTES, ADD_MONTHS, ADD_SECONDS, ADD_YEARS
    - En plus de l'opérateur naturel « + »
  - CURDATE, CURTIME, NOW
    - Horodatage système
  - DATE, DAY, DAYNAME, DAYOFMONTH, DAYOFWEEK, DAYOFWEEK_ISO, DAYOFYEAR, DAYS, EXTRACT, HOUR, LAST_DAY, MINUTE, MONTH, MONTHNAME, MONTHS_BETWEEN,SECOND NEXT_DAY, QUARTER, TIME, TIMESTAMP, TIMESTAMP_FORMAT, TIMESTAMP_ISO, TIMESTAMPDIFF, TIMESTAMPDIFF_BIG, TO_CHAR, TO_DATE, TO_TIMESTAMP, VARCHAR_FORMAT, WEEK, WEEK_ISO, YEAR
    - Découpage, calcul, formatage …

Dates / heures / horodatages

- Simple !

```
100 values varchar_format(current timestamp, 'Day dd month YYYY') ;
```

00001
Dimanche 05 novembre 2023

```
102 -- jour et 30 jours fin de mois
103 values current date, LAST_DAY(current date + 30 days);
```

00001
2023-11-05
2023-12-31

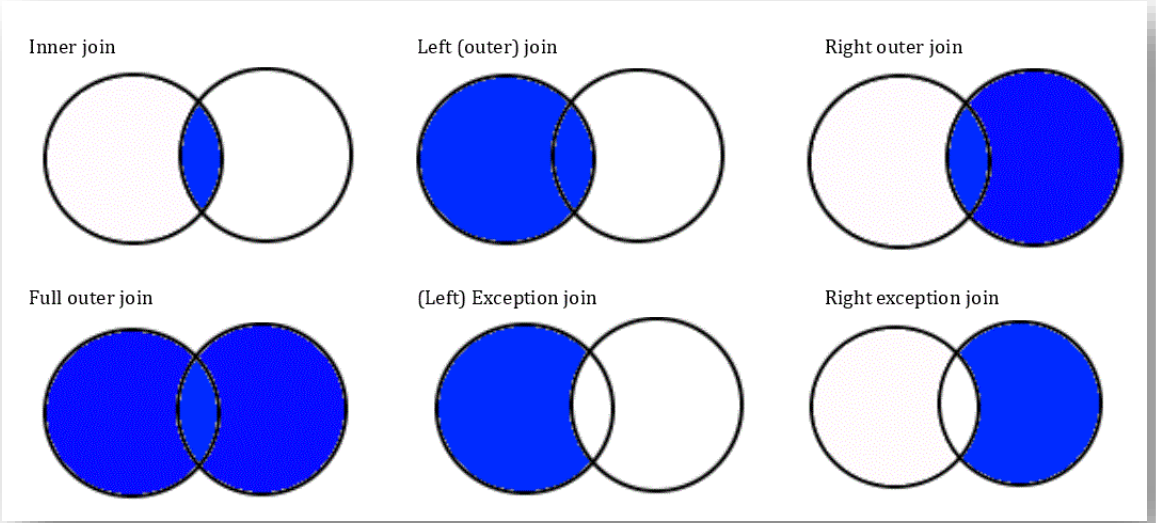
```
105 -- quel trimestre ?
106 -- la fin de mois prochaine est-elle dans le même trimestre ?
107 values quarter(current date), quarter(LAST_DAY(current date + 1 months));
```

00001
4
4

Jointures

- Rappel
  - L'ordre n'a plus d'importance !
  - Le moteur SQL (SQE) réécrit vos requêtes

  - Plusieurs types de jointures
    - INNER JOIN
    - **CROSS JOIN**
    - LEFT/RIGHT OUTER JOIN
    - LEFT/RIGHT EXCEPTION
    - FULL OUTER
    - **LATERAL**



Jointures – CROSS JOIN

- Ce n’est pas une vraie jointure
  - Un produit cartésien (toutes les lignes de la table A par toutes les lignes de la table B)
  - Souvent n’est pas écrit syntaxiquement

```
-- CROSS JOIN
SELECT *
    FROM employee
         CROSS JOIN department
    WHERE WORKDEPT = DEPTNO;
```

```
-- Souvent
SELECT *
    FROM employee,
         department
    WHERE WORKDEPT = DEPTNO;
```

  - A remplacer par [INNER] JOIN

```
SELECT *
    FROM employee
         JOIN department
             ON WORKDEPT = DEPTNO;
```

Jointures – CROSS JOIN

- Intérêt ?
  - Meilleure utilisation des clés par l'optimiseur
  - Meilleure lisibilité : on distingue
    - Les critères de jointure = sur quels critères on rapproche des lignes de plusieurs tables
    - Les critères de sélection= quelles sont les lignes qui nous intéressent à la fin

Utiliser des alias de table

- Facilite la lecture
  - On sait directement de quelle table provient quelle colonne

```
-- alias de table
SELECT e.EMPNO,
       e.FIRSTNME,
       e.LASTNAME,
       e.WORKDEPT,
       e.HIREDATE,
       e.JOB,
       d.DEPTNAME
    FROM employee e
         JOIN department d
              ON e.WORKDEPT = d.DEPTNO
    WHERE e.hiredate > '1975-01-01';
```

Sous-requête -> jointure

- Eviter les sous-requêtes IN/NOT IN/EXISTS
  - La sous-requête peut être répétée à chaque ligne de la requête globale

```
-- Les employés embauchés à partir de 1975 dans les départements ayant un manager
SELECT e.EMPNO,
       e.FIRSTNME,
       e.LASTNAME,
       e.WORKDEPT,
       e.HIREDATE,
       e.JOB
    FROM employee e
    WHERE e.hiredate > '1975-01-01' AND
          e.workdept IN (SELECT d.deptno
                  FROM department d
                  WHERE d.mgrno IS NOT NULL) ;
```

```
-- Les employés embauchés à partir de 1975 dans les départements ayant un manager
SELECT e.EMPNO,
       e.FIRSTNME,
       e.LASTNAME,
       e.WORKDEPT,
       e.HIREDATE,
       e.JOB
    FROM employee e join department d ON e.WORKDEPT = d.DEPTNO
    WHERE e.hiredate > '1975-01-01' AND mgrno IS NOT NULL ;
```

Penser ensemble !

- Et non ligne à ligne
  1. Faire une seule requête
  2. Eviter les fichiers temporaires (cf 1)
  3. Eviter les curseurs (cf 1 + 2)

- Pour cela : CTE
  - Common Table Expression
  - Capacité à utiliser des requête temporaires dans la requête
  - 1 CTE peut se baser sur une autre CTE définie précédemment
  - La requête finale utilise les CTE
  - Intérêt : le moteur SQL sait ce que voulez faire à la fin !

Penser ensemble !

- Principe

```
-- Les employés embauchés à partir de 1975 dans les départements ayant un manager
WITH dept_mgr AS (
        SELECT deptno,
               deptname
            FROM department
            WHERE mgrno IS NOT NULL
    )
    SELECT e.EMPNO,
           e.FIRSTNME,
           e.LASTNAME,
           e.WORKDEPT,
           e.HIREDATE,
           e.JOB,
           d.DEPTNAME
        FROM employee e
             JOIN dept_mgr d
                 ON e.WORKDEPT = d.DEPTNO
        WHERE e.hiredate > '1975-01-01';
```

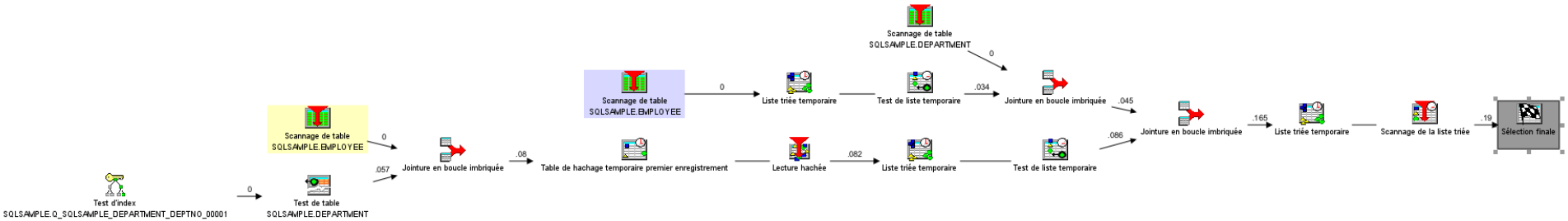
Penser ensemble !

- Exemple

```
-- Les employés embauchés à partir de 1970 dans les départements ayant un manager
-- Qui ne sont pas eux-mêmes manager
-- Et qui gagnent plus que la moyenne des employés (non manager) dans leur département
WITH dept_mgr AS (
        SELECT cte_d.deptno,
               cte_d.deptname,
               cte_d.mgrno
            FROM department cte_d
            WHERE cte_d.mgrno IS NOT NULL
    ),
    dept_avg_sal AS (
        SELECT cte_e.WORKDEPT,
               AVG(cte_e.salary) AS avgsal
            FROM employee cte_e
                 LEFT EXCEPTION JOIN department ct2_d
                     ON cte_e.WORKDEPT = ct2_d.DEPTNO AND
                         cte_e.empno <> ct2_d.mgrno
            GROUP BY cte_e.WORKDEPT
    )
    SELECT e.EMPNO,
           e.FIRSTNME,
           e.LASTNAME,
           e.WORKDEPT,
           e.HIREDATE,
           e.JOB,
           e.salary,
           d.DEPTNAME,
           d.mgrno,
           INT(s.avgsal) AS salaire_moyen_dept
       FROM employee e
            JOIN dept_mgr d
                ON e.WORKDEPT = d.DEPTNO AND
                    e.EMPNO <> d.MGRNO
            LEFT JOIN dept_avg_sal s
                ON e.workdept = s.WORKDEPT AND
                    e.salary < s.avgsal
       WHERE e.hiredate > '1970-01-01'
       ORDER BY (s.avgsal - e.salary) DESC;
```

Penser ensemble !

- Exemple
  - Le moteur réécrit tout cela et exécute au mieux !



Penser ensemble !

- Intérêt
  - Pas de fichier dans QTEMP
  - Pas de programme refaisant une lecture ligne à ligne de fichiers intermédiaires
  - Capacité du moteur SQL de réécrire / optimiser la requête dans sa globalité
  - Plus simple à écrire

ALIAS de colonne

- Pour donner un nom
  - A vos colonnes calculées
  - A des colonnes non explicites

```
-- Les employés embauchés à partir de 1970 dans les départements ayant un manager
-- Qui ne sont pas eux-mêmes manager
-- Et qui gagnent plus que la moyenne des employés (non manager) dans leur departemen
WITH dept_mgr AS (
        SELECT cte_d.deptno,
               cte_d.deptname,
               cte_d.mgrno
            FROM department cte_d
            WHERE cte_d.mgrno IS NOT NULL
    ),
    dept_avg_sal AS (
        SELECT cte_e.WORKDEPT,
               AVG(cte_e.salary) AS avgsal
            FROM employee cte_e
                 LEFT EXCEPTION JOIN department ct2_d
                     ON cte_e.WORKDEPT = ct2_d.DEPTNO AND
                         cte_e.empno <> ct2_d.mgrno
            GROUP BY cte_e.WORKDEPT
    )
```

```
    SELECT e.EMPNO,
           e.FIRSTNME,
           e.LASTNAME,
           e.WORKDEPT,
           e.HIREDATE,
           e.JOB,
           e.salary,
           d.DEPTNAME,
           d.mgrno,
           INT(s.avgsal) AS salaire_moyen_dept
        FROM employee e
             JOIN dept_mgr d
                 ON e.WORKDEPT = d.DEPTNO AND
                     e.EMPNO <> d.MGRNO
             LEFT JOIN dept_avg_sal s
                 ON e.workdept = s.WORKDEPT AND
                     e.salary < s.avgsal
        WHERE e.hiredate > '1970-01-01'
        ORDER BY (s.avgsal - e.salary) DESC;
```

Transaction

- Si vous utilisez le contrôle transactionnel
  - Eviter les transactions longues
  - En durée
    - 1 transaction BD est une action la plus courte possible
    - Pas d'attente d'événement extérieur
    - Ne pas confondre avec une transaction métier
  - En nombre de lignes
    - A priori limité
    - Quel usage de transaction à 1.000.000 de lignes ?

IBM

Infrastructures du futur

7 et 8 novembre 2023

Université IBM i

7 novembre 2023

IBM i 35 YEARS

Let's Create

3. Conception

Relations & Clés

- DB2 est une base de données relationnelle
  - Relation = clé étrangère
  - Ie : les enregistrements d'une table référence la clé d'une autre table
  - ➔ nécessitent que les tables aient des clés !

- En SQL
  - Clé primaire vs Clé unique
    - Gestion des valeurs nulles diffère

- Les clés sont nécessaires pour manipuler les données
  - Accès aux lignes
  - Jointures, sélections, tris

Relations & Clés

- Formes Normales
  - https://fr.wikipedia.org/wiki/Forme_normale_(bases_de_donn%C3%A9es_relationnelles)

- En synthèse
  - Chaque information est unique
  - Identifiée par une clé
  - Pas de redondance

Relations & Clés

- Pour que DB2 fonctionne bien il faut de « bonnes clés »
  - INT

- Souvent vous utilisez des clés « fonctionnelles »
  - N° SS
  - IBAN
  - Année / chrono / type / sous-type

- Souvent vous calculez les clés
  - Dernière + 1
  - Pose des problèmes d'accès, verrouillage, valeur perdue en cas de ROLLBACK de l'insertion …

Relations & Clés

- DB2 dispose de fonctions pour vous aider
  - Clé auto-générées
    - La valeur de la clé est définie par DB2 à l'insertion
    - Possibilité de retrouver la dernière clé générée

```
2  CREATE OR REPLACE TABLE NB.U23_SESSION (
3             "ID" INTEGER GENERATED ALWAYS AS IDENTITY,
4             TITRE VARCHAR(100) NOT NULL NOT HIDDEN
5          ) ;
6
7  insert into NB.U23_SESSION(titre) values('S01 - Intégrez les fonctions géospatiales de DB2 dans vos applications') ;
8  VALUES IDENTITY_VAL_LOCAL() ;
```

```
00001
    1
```

Conception des tables

- Souvent de trop nombreuses colonnes dans les tables
  - Le moteur SQL sait extrêmement bien faire les relations (jointures)

- Donc pas besoin d'avoir +200 colonnes !
  - Faites plusieurs tables
  - Faites des jointures
  - Faites des vues

Vue

- Intérêt
  - Réduit la complexité des grosses requêtes
  - Fournit des éléments réutilisables et bien maitrisés, optimisés
  - Permet de gérer les droits
    - Une vue sur les employés sans les salaires

- Aucun impact sur la performance
  - La vue ne contient pas de ligne (ce n'est pas un LF au sens DDS ni un index)
  - Le SELECT est rejoué à chaque interrogation de la vue

Vue

- Exemple

```
-- Les employés embauchés à partir de 1970 dans les départements ayant un manager
-- Qui ne sont pas eux-mêmes manager
-- Et qui gagnent plus que la moyenne des employés (non manager) dans leur département
WITH dept_mgr AS (
        SELECT cte_d.deptno,
               cte_d.deptname,
               cte_d.mgrno
            FROM department cte_d
            WHERE cte_d.mgrno IS NOT NULL
    ),
    dept_avg_sal AS (
        SELECT cte_e.WORKDEPT,
               AVG(cte_e.salary) AS avgsal
            FROM employee cte_e
                 LEFT EXCEPTION JOIN department ct2_d
                     ON cte_e.WORKDEPT = ct2_d.DEPTNO AND
                         cte_e.empno <> ct2_d.mgrno
            GROUP BY cte_e.WORKDEPT
    )
```

```
CREATE VIEW department_manager AS
        (SELECT cte_d.deptno,
                cte_d.deptname,
                cte_d.mgrno
            FROM department cte_d
            WHERE cte_d.mgrno IS NOT NULL);

CREATE VIEW department_average_salary AS
        (SELECT cte_e.WORKDEPT,
                AVG(cte_e.salary) AS avgsal
            FROM employee cte_e
                 LEFT EXCEPTION JOIN department ct2_d
                     ON cte_e.WORKDEPT = ct2_d.DEPTNO AND
                         cte_e.empno <> ct2_d.mgrno
            GROUP BY cte_e.WORKDEPT);
```

Vue

- Exemple

```
SELECT e.EMPNO,
       e.FIRSTNME,
       e.LASTNAME,
       e.WORKDEPT,
       e.HIREDATE,
       e.JOB,
       e.salary,
       d.DEPTNAME,
       d.mgrno,
       INT(s.avgsal) AS salaire_moyen_dept
   FROM employee e
        JOIN dept_mgr d
            ON e.WORKDEPT = d.DEPTNO AND
               e.EMPNO <> d.MGRNO
        LEFT JOIN dept_avg_sal s
            ON e.workdept = s.WORKDEPT AND
               e.salary < s.avgsal
   WHERE e.hiredate > '1970-01-01'
   ORDER BY (s.avgsal - e.salary) DESC;
```

```
SELECT e.EMPNO,
       e.FIRSTNME,
       e.LASTNAME,
       e.WORKDEPT,
       e.HIREDATE,
       e.JOB,
       e.salary,
       d.DEPTNAME,
       d.mgrno,
       INT(s.avgsal) AS salaire_moyen_dept
   FROM employee e
        JOIN department_manager d
            ON e.WORKDEPT = d.DEPTNO AND
               e.EMPNO <> d.MGRNO
        LEFT JOIN department_average_salary s
            ON e.workdept = s.WORKDEPT AND
               e.salary < s.avgsal
   WHERE e.hiredate > '1970-01-01'
   ORDER BY (s.avgsal - e.salary) DESC;
```

Conserver des données calculées …

- Rendre persistante la donnée calculée
  - Exemple des dates
  - Peut violer la normalisation …
- Pour une date
  - Les attributs ne changent pas
  - On a souvent besoin
  - On peut les calculer et les stocker

```
-- Combien de ventes par trimestre ?
SELECT YEAR(s.sales_date) AS annee,
       QUARTER(s.sales_date) AS trimestre,
       SUM(sales) as total_vente
    FROM sales s
    GROUP BY YEAR(s.sales_date),
             QUARTER(s.sales_date)
    ORDER BY annee,
             trimestre ;
```

Conserver des données calculées …

- Exemple

```
CREATE TABLE SQLSAMPLE.CALENDRIER (
            DATE_JOUR DATE NOT NULL NOT HIDDEN,
            ANNEE INTEGER NOT NULL NOT HIDDEN,
            MOIS INTEGER NOT NULL NOT HIDDEN,
            JOUR INTEGER NOT NULL NOT HIDDEN,
            SEMAINE INTEGER NOT NULL NOT HIDDEN,
            TRIMESTRE INTEGER NOT NULL NOT HIDDEN,
            OUVREE CHARACTER(1) DEFAULT 'N' NOT NULL NOT HIDDEN,
            PRIMARY KEY (DATE_JOUR)
        )
    NOT VOLATILE UNIT ANY KEEP IN MEMORY NO;
```

```
insert into calendrier
with nbrs ( n ) as (
   select 0 from (values 0) as a union all
   select 1 + n from nbrs where n < 365 ),
    detail as ( select current date + n days as datew from nbrs )
select datew , year(datew), month(datew), day(datew), week(datew), quarter(datew), 'O'
from detail ;
```

Conserver des données calculées …

- Exemple

```
-- Combien de ventes par trimestre ?
SELECT YEAR(s.sales_date) AS annee,
       QUARTER(s.sales_date) AS trimestre,
       SUM(sales) as total_vente
    FROM sales s
    GROUP BY YEAR(s.sales_date),
             QUARTER(s.sales_date)
    ORDER BY annee,
             trimestre ;
```

```
SELECT c.annee,
       c.trimestre,
       SUM(sales) as total_vente
    FROM sales s
    join calendrier c on s.sales_date = c.date_jour
    GROUP BY c.annee, c.trimestre
    ORDER BY c.annee,
             c.trimestre ;
```

  - Permet également
    - Sélection sur le calendrier
    - Données « métier » supplémentaires
    - Indexation pour performance
    - …

Et encore

- Index dérivés
- MERGE vs UPDATE
- DISTINCT
- …

IBM

Infrastructures du futur

7 et 8 novembre 2023

Université IBM i

7 novembre 2023

IBM i 35 YEARS

Let's Create

Q/R

54