Université IBM i

19 et 20 novembre 2024

IBM Innovation Studio Paris

uii2024
#ibmi
#uii2024

S01 – DB2-SQL : 30 trucs et astuces en 60 mn

19 novembre 11:15 - 12:15

Philippe Bourgeois
IBM France
pbourgeois@fr.ibm.com

IBM

common FRANCE

Plan de la présentation (1/2)

- 1. Clauses LIMIT et OFFSET – Pagination dynamique
- 2. Clause LIMIT sur UPDATE et DELETE
- 3. Clause LATERAL pour sous-requêtes corrélées
- 4. Clause LATERAL pour faciliter l'utilisation d'expressions
- 5. Fonction LOCATE_IN_STRING
- 6. Fonction VARCHAR_FORMAT pour formater des zones numériques
- 7. Fonction VARCHAR_FORMAT pour formater des horodates
- 8. Fonction TRY_CAST
- 9. Fonction FIRST_DAYS
- 10. Fonction VALIDATE_DATA
- 11. Fonction ANY_VALUE
- 12. QCMDEXC en fonction scalaire
- 13. Fonction GET DIAGNOSTICS
- 14. SELF (SQL Error Logging Facility)
- 15. Vues flexibles

Plan de la présentation (2/2)

- 16. Gestion du CURRENT PATH
- 17. Fonction JSON_UPDATE
- 18. La vue SYSFILES de QSYS2
- 19. Exécution d'une commande CL sur une partition distante
- 20. ACS : SELECT FOR UPDATE
- 21. ACS : invite sur variables
- 22. ACS : métadonnées étendues
- 23. ACS : génération du SQL DDL
- 24. ACS : affichage des objets dépendants
- 25. ACS : génération du SQL DML
- 26. VS Code for i : sorties JSON et CSV
- 27. ACS : projets SQL
- 28. VS Code for i : notebooks
- 29. ACS et VS Code for i : exemples SQL
- 30. SQL Tutor

Présentations 2024 **complémentaires** à cette présentation

- **S08** – Les services SQL de SYSTOOLS
  - Mardi 19 novembre – 13h30-14h30
  - SEND_EMAIL, GENERATE_SPREADSHEET

- **S15** – Tout savoir sur le catalogue de DB2 for i
  - Mardi 19 novembre – 14h45-15h45

- **S22** – Regrouper, accumuler et puis diviser à nouveau avec SQL
  - Mardi 19 novembre – 16h00-17h00

- **S43/S48** – Les services SQL pour l'IFS
  - Mercredi 20 novembre – 13h45-14h45 et 15h00-16h00

1. Clauses LIMIT et OFFSET – Pagination dynamique

- **LIMIT x** permet de limiter le résultat à **x** lignes

- **OFFSET y** permet de sauter les **y** premières lignes

```
-- Pagination - Affichage des enregistrements par page de 5
-- La 1ère page
SELECT * FROM tabempl LIMIT 5 OFFSET 0;
-- OU
SELECT * FROM tabempl LIMIT 0, 5;

-- La 2nde page
SELECT * FROM tabempl LIMIT 5, 5;

-- La 3ème page
SELECT * FROM tabempl LIMIT 10, 5;

-- La nième page
SELECT * FROM tabempl LIMIT ((5* :page) - 5), 5;
```

1. Clauses LIMIT et OFFSET – Pagination dynamique

- Encapsulation dans une **procédure**

```
CREATE OR REPLACE PROCEDURE pListeEmployes (IN numPage INT DEFAULT 1)
   SPECIFIC pListEmp
   RESULT SETS 1
   LANGUAGE SQL
 BEGIN
   DECLARE empListe CURSOR WITH RETURN TO CALLER FOR
     SELECT * FROM tabempl ORDER BY nom
      LIMIT (5*numPage - 5), 5;
  OPEN empListe;
 END;
```

```
CALL pListeEmployes(:page);
```

1. Clauses LIMIT et OFFSET – Pagination dynamique

- Encapuslation dans une **fonction table**

```
CREATE OR REPLACE FUNCTION fListeEmployes (numPage INT DEFAULT 1)
   RETURNS TABLE (nom VARCHAR(30), sx CHAR(1), datenai DATE)
   LANGUAGE SQL
   SPECIFIC fListEmp
   CARDINALITY 5
   RETURN SELECT nom, sx, dat_nai FROM tabempl ORDER BY nom
          LIMIT (5*numPage - 5), 5;
```

```
SELECT * FROM TABLE (fListeEmployes(:page));
```

2. Clause LIMIT sur UPDATE et DELETE

- La clause **LIMIT x** permet de limiter le résultat à **x** lignes
  - Valable pour les instructions SELECT, **UPDATE** et **DELETE**

```
-- Suppression des 100 plus anciennes commandes de la table historique des commandes
DELETE FROM histo_commandes
 ORDER BY datecmd LIMIT 100;


-- Mise à jour de la table des fournisseurs pour indiquer OUI dans la colonne TOP_FOURNISSEUR
-- pour les 10 fournisseurs qui ont fait le meilleur CA
UPDATE fournisseurs
 SET top_fournisseur = 'OUI'
 ORDER BY CA DESC
 LIMIT 10;
```

3 et 4. Clause LATERAL

- Les tables utilisées pour l'exemple

```
91 SELECT * FROM tabempl;
```

MAT	NOM	SX	SRV	SAL	DAT_NAI
20	MICHEL	M	911	2140.00	1955-03-17
10	ANNIE	F	911	3080.00	1959-05-12
50	JACQUES	M	911	2360.00	1956-11-11
40	DANIELE	F	977	2810.00	1962-07-13
30	MARC	M	977	3570.00	1972-02-28
60	CLAUDE	M	990	2210.00	1988-05-02
70	RICHARD	M	-	-	1956-03-01

```
90 SELECT * FROM tabserv;
```

SRV	NOMSRV
901	COMPTA
977	MANUFACT
911	VENTES
982	APRES VTES

3. Clause LATERAL et sous-requêtes corrélées

- Rappel – **Corrélation** dans une sous-requête
  - Faire réference à des colonnes qui sont en dehors de la sous-requête
  - Une sous-requête corrélée est permise lorsqu'elle est utilisée dans la position d'une colonne

```
95 -- Quels sont les employés qui ont un salaire supérieur
96 -- à la moyenne des salaires de leur service ?
97 SELECT srv, nom, sal FROM tabempl e
98  WHERE sal > (SELECT AVG(sal) FROM tabempl WHERE srv = e.srv);
```

SRV	NOM	SAL
911	ANNIE	3080.00
977	MARC	3570.00

3. Clause LATERAL et sous-requêtes corrélées

- Une sous-requête corrélée n'est **pas** permise lorsque la sous-requête est utilisée dans la position d'une table (jointure dans la clause FROM)
  - Voici ce que l'on souhaite obtenir :

SRV	NOMSRV	SALMOYEN	NBEMP
901	COMPTA	-	0
977	MANUFACT	3190.00000000000000000000000000	2
911	VENTES	2526.66666666666666666666666666	3
982	APRES VTES	-	0

SRV	NOMSRV
901	COMPTA
977	MANUFACT
911	VENTES
982	APRES VTES

MAT	NOM	SX	SRV	SAL	DAT_NAI
20	MICHEL	M	911	2140.00	1955-03-17
10	ANNIE	F	911	3080.00	1959-05-12
50	JACQUES	M	911	2360.00	1956-11-11
40	DANIELE	F	977	2810.00	1962-07-13
30	MARC	M	977	3570.00	1972-02-28
60	CLAUDE	M	990	2210.00	1988-05-02
70	RICHARD	M	-	-	1956-03-01

3. Clause LATERAL et sous-requêtes corrélées

- Une sous-requête corrélée n'est pas permise lorsque la sous-requête est utilisée dans la position d'une table (jointure dans la clause FROM)
  - Exemple de requête incorrecte :

```
101 SELECT s.srv, s.nomsrv, e2.salmoyen, e2.nbemp
102  FROM tabserv s,
103      (SELECT AVG(e.sal) AS salmoyen, COUNT(*) AS nbemp
104       FROM tabempl e
105       WHERE e.srv = s.srv) AS e2;
```

```
✖ Etat SQL : 42703
Code fournisseur : -206
Message : [SQL0206] La colonne ou la variable globale SRV est introuvable.
```

3. Clause LATERAL et sous-requêtes corrélées

- Permet de définir une "jointure latérale" (LATERAL JOIN)

- Syntaxe :
  - `SELECT … FROM table_principale, LATERAL(sous-requête)`
    OU
  - `SELECT … FROM table_principale CROSS JOIN LATERAL(sous-requête)`

- La sous-requête pourra être une requête corrélée, donc référencer des colonnes indiquées dans la requête "principale"
  - En fait il ne s'agit pas vraiment d'une jointure, mais d'une boucle FOR-EACH :
    - Pour chaque ligne de la table principale exécuter la sous-requête en utilisant les valeurs de la ligne en cours de la table principale
    - Le résultat de la sous-requête est appelé "table dérivée latérale"

3. Clause LATERAL et sous-requêtes corrélées

- Requête correcte

```
114 SELECT s.srv, s.nomsrv, e2.salmoyen, e2.nbemp
115  FROM tabserv s,
116      LATERAL(SELECT AVG(e.sal) AS salmoyen, COUNT(*) AS nbemp
117                FROM tabempl e
118                WHERE e.srv = s.srv) AS e2;
119 -- OU
120 SELECT s.srv, s.nomsrv, e2.salmoyen, e2.nbemp
121  FROM tabserv s CROSS JOIN
122      LATERAL(SELECT AVG(e.sal) AS salmoyen, COUNT(*) AS nbemp
123                FROM tabempl e
124                WHERE e.srv = s.srv) AS e2;
```

SRV	NOMSRV	SALMOYEN	NBEMP
901	COMPTA	-	0
977	MANUFACT	3190.0000000000000000000000000	2
911	VENTES	2526.6666666666666666666666666	3
982	APRES VTES	-	0

3. Clause LATERAL et sous-requêtes corrélées

- Autre exemple

```
135 -- Objets IFS de /home/bourgeois qui ne sont pas en *PUBLIC *EXCLUDE
136
137 SELECT path_name, os.object_type, owner, data_authority
138  FROM TABLE(qsys2.ifs_object_statistics(START_PATH_NAME => '/home/bourgeois',
139                                         SUBTREE_DIRECTORIES => 'YES')) os
140  CROSS JOIN
141  LATERAL (SELECT object_type, owner, data_authority
142           FROM TABLE(qsys2.ifs_object_privileges(PATH_NAME => os.path_name)) op
143           WHERE op.authorization_name = '*PUBLIC' AND data_authority <> '*EXCLUDE');
```

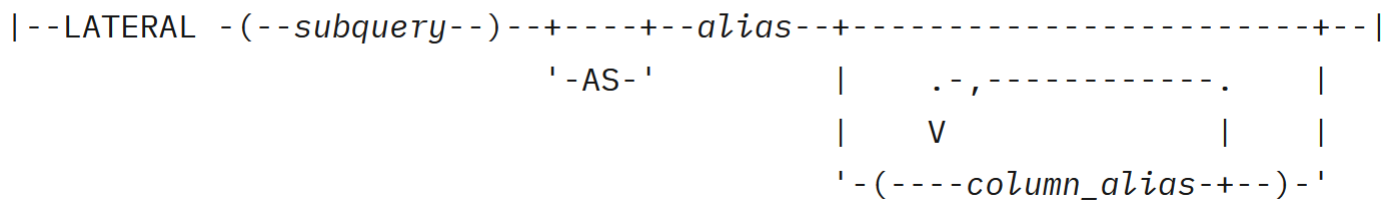
PATH_NAME	OBJECT_TYPE	OWNER	DATA_AUTHORITY
/home/bourgeois	*DIR	QSECOFR	*RX
/home/bourgeois/.eclipse	*DIR	BOURGEOIS	*RWX
/home/bourgeois/.sh_history	*STMF	BOURGEOIS	*NONE

4. Clause LATERAL pour faciliter l'utilisation d'expressions

- Autres utilisations de la clause LATERAL :
  - Eviter de répéter les expressions dans la liste des colonnes et dans les clauses WHERE et ORDER BY
  - Encapsuler de la logique dans les colonnes
    - En utilisant la clause VALUES à la place de la sous-requête

- Syntaxe :

LATERAL derived table



4. Clause LATERAL pour faciliter l'utilisation d'expressions

- Exemple **sans** LATERAL

```
SELECT nom, sal*12 AS salannuel, YEAR(CURRENT DATE - dat_nai) AS age
 FROM tabempl
 WHERE sal*12 > 30000 AND YEAR(CURRENT DATE - dat_nai) > 60
 ORDER BY age, salannuel;
```

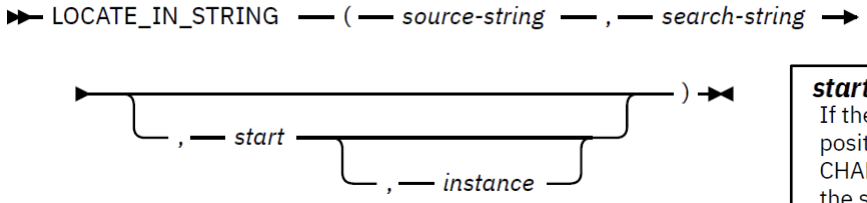
On répète les expressions dans le WHERE

- Exemple **avec** LATERAL

```
SELECT nom, salannuel, age
 FROM tabempl
  CROSS JOIN
 LATERAL (VALUES ((sal*12), YEAR(CURRENT DATE - dat_nai))) AS t(salannuel, age)
 WHERE salannuel > 30000 AND age > 60
 ORDER BY age, salannuel;
```

5. Fonction LOCATE_IN_STRING

- Fournit la position d'une **instance** de chaîne de caractères



start
If the value of the integer is greater than zero, the search begins at *start* and continues for each position to the end of the string. If the value of the integer is less than zero, the search begins at CHARACTER_LENGTH(*source-string*) + *start* + 1 and continues for each position to the beginning of the string.

```
160 SELECT nomsrv, LOCATE_IN_STRING(nomsrv, 'E', 1, 2) FROM tabserv;
```

NOMSRV	00002
COMPTA	0
MANUFACT	0
VENTES	5
APRES VTES	9

5. Fonction LOCATE_IN_STRING

- **Problématique** : à partir de la chaîne de caractères `/pb/json/ws/iws/films.json`, extraire le nom du fichier (`films.json`)

```
25 -- LOCATE_IN_STRING
26
27 CREATE VARIABLE ifs_path VARCHAR(200);
28 SET ifs_path = '/pb/json/ws/iws/films.json';
29
30 VALUES LOCATE_IN_STRING(ifs_path, '/', -1); -- 16
31
32 VALUES SUBSTR(ifs_path, LOCATE_IN_STRING(ifs_path, '/', -1) + 1);
```

```
00001
films.json
```

6. Fonction VARCHAR_FORMAT pour formater des zones numériques

- **VARCHAR_FORMAT** : formate une zone **numérique**, alphanumérique ou horodate dans le format souhaité et la transforme en VARCHAR

```
58 SELECT nom,
59        sal,
60        VARCHAR_FORMAT(sal, '99G999D99MIL') as sal2
61  FROM tabempl;
```

NOM	SAL	SAL2
MICHEL	2140.00	2.140,00 €
ANNIE	3080.00	3.080,00 €

6. Fonction VARCHAR_FORMAT pour formater des zones numériques

- Formatage **numérique** en VARCHAR

Table 68. Format elements for the VARCHAR_FORMAT (Numeric to VARCHAR) function

Format element	Description
0	Each 0 represents a significant digit. Leading zeros in a number are displayed as zeros.
9	Each 9 represents a significant digit. Leading zeros in a number are displayed as blanks. Only group separators that have at least one digit to the left of the separator are generated.
S	Prefix: If *numeric-expression* is a negative number, a leading minus sign (-) is included in the result. If *numeric-expression* is a positive number, a leading plus sign (+) is included in the result.
$	Prefix: A leading dollar sign ($) is included in the result.
MI	Suffix: If *numeric-expression* is a negative number, a trailing minus sign (-) is included in the result. If *numeric-expression* is a positive number, a trailing blank is included in the result.
PR	Suffix: If *numeric-expression* is a negative number, a leading less than character (<) and a trailing greater than character (>) are included in the result. If *numeric-expression* is a positive number, a leading space and a trailing space are included in the result.
,	Specifies that a comma be included in that location in the result. This comma is used as a group separator.
.	Specifies that a period be included in that location in the result. This period is used as a decimal point.
L	Prefix or Suffix: Specifies that the local currency symbol be included in that location in the result. The currency symbol is retrieved from message CPX8416 in message file QCPFMSG in library *LIBL.
D	Specifies that the local decimal point character be included in that location in the result. The decimal character is retrieved from message CPX8416 in message file QCPFMSG in library *LIBL.
G	Specifies that the local group separator character be included in that location in the result. If the local decimal character as retrieved from message CPX8416 in message file QCPFMSG in library *LIBL is a period, the group separator will be a comma. If the local decimal character is a comma, the group separator will be a period.

7. Fonction VARCHAR_FORMAT pour formater des horodates

- **VARCHAR_FORMAT** : formate une zone numérique, alphanumérique ou **horodate** dans le format souhaité et la transforme en VARCHAR

```
63 VALUES VARCHAR_FORMAT(NOW(), 'Day DD month YYYY');
```

00001
Dimanche 03 novembre 2024

```
65 VALUES VARCHAR_FORMAT(NOW(), 'Dy DD mon YYYY');
```

00001
Dim 03 nov 2024

7. Fonction VARCHAR_FORMAT pour formater des horodates

- Formatage **horodate** en VARCHAR

Table 67. Format elements for the VARCHAR_FORMAT (Timestamp to VARCHAR) function

Format	Unit
AM or PM [1, 2]	Meridian indicator (morning or evening) without periods. The meridian indicator that is returned is based on the language used for messages in the job. This meridian indicator is retrieved from message CPX9035 in message file QCPFMSG in library *LIBL.
A.M. or P.M. [1, 2]	Meridian indicator (morning or evening) with periods. This format element uses the exact strings 'A.M.' or 'P.M.' and is independent of the language used for messages in the job.
CC	Century (00-99). If the last two digits of the four digit year are zero, the result is the first two digits of the year. Otherwise, the result is the first two digits of the year plus one.
DAY, Day, or day [1, 2]	Name of the day in uppercase, titlecase, or lowercase format. The name of the day that is returned is based on the language used for messages in the job. This name of the day is retrieved from message CPX9034 in message file QCPFMSG in library *LIBL.
DY, Dy, or dy [1, 3]	Abbreviated name of the day in uppercase, titlecase, or lowercase format. The abbreviated name of the day is retrieved from message CPX9039 in message file QCPFMSG in library *LIBL.
D [1]	Day of week (1-7), where 1 is Sunday.
DD	Day of month (01-31).
DDD	Day of year (001-366).

Table 67. Format elements for the VARCHAR_FORMAT (Timestamp to VARCHAR) function (continued)

Format	Unit
FF or FF*n*	Fractional seconds (0-999999999999). The number *n* is used to specify the number of digits to include in the value returned. Valid values for *n* are 1-12. The default is 6.
HH	HH behaves the same as HH12.
HH12	Hour of the day (01-12) in 12-hour format.
HH24	Hour of the day (00-24) in 24-hour format.
ID	ISO day of week (1-7), where 1 is Monday and 7 is Sunday
IW	ISO week of year (01-53). The week starts on Monday and includes 7 days. Week 1 is the first week of the year to contain a Thursday, which is equivalent to the first week of the year to contain January 4
I	ISO year (0-9). The last digit of the year based on the ISO week that is returned.
IY	ISO year (00-99). The last two digits of the year based on the ISO week that is returned.
IYY	ISO year (000-999). The last three digits of the year based on the ISO week that is returned.
IYYY	ISO year (0000-9999). The year based on the ISO week that is returned.
J	Julian date (0000000-9999999).
MI	Minute (00-59).
MM	Month (01-12).

7. Fonction VARCHAR_FORMAT pour formater des horodates

- Formatage **horodate** en VARCHAR

MONTH, Month, or month [1, 3]	Name of the month in uppercase, titlecase, or lowercase format. The name of the month that is returned is based on the language used for messages in the job. This name of the month is retrieved from message CPX3BC0 in message file QCPFMSG in library *LIBL.
MON, Mon, or mon [1, 3]	Abbreviated name of the month in uppercase, titlecase, or lowercase format. The name of the month that is returned is based on the language used for messages in the job. This name of the month is retrieved from message CPX8601 in message file QCPFMSG in library *LIBL.
MS	Milliseconds (000-999). Same as FF3.
NNNNNN	Microseconds (000000-999999). Same as FF6.
Q	Quarter (1-4).
RR	RR behaves the same as YY.
RRRR	RRRR behaves the same as YYYY.
SS	Seconds (00-59).
SSSSS	Seconds since previous midnight (00000-86400).
US	Microseconds (000000-999999). Same as FF6.
W	Week of month (1-5). Week 1 starts on the first day of the month and ends on the seventh day.
WW	Week of the year (01-53), where week 1 starts on January 1 and ends on January 7.

Table 67. Format elements for the VARCHAR_FORMAT (Timestamp to VARCHAR) function (continued)

Format	Unit
Y	Last digit of the year (0-9).
YY	Last two digits of the year (00-99).
YYY	Last three digits of the year (000-999).
YYYY	Year (0000-9999).

8. Fonction TRY_CAST

- Alternative à la fonction CAST

- En cas de donnée **invalide** dans une colonne :
  - **CAST** : génération d'un avertissement
  - **TRY_CAST** : renvoie la valeur nulle

```
183 SELECT z1,
184        CAST(z1 AS INT) AS CAST_Z1,
185        TRY_CAST(z1 AS INT) AS TRY_CAST_Z1
186  FROM trycast1;
```

Z1	CAST_Z1	TRY_CAST_Z1
10	10	10
20	20	20
345	345	345
ABCD	++++++++++++	-
999	999	999

```
⚠ Etat SQL : 01565
Code fournisseur : 420
Message : [SQL0420] Valeur pour l'argument CAST non valide.
```

9. Fonction FIRST_DAY

- Renvoie la date du 1er jour du mois

```
190 VALUES current date,
191        FIRST_DAY(current date),
192        LAST_DAY(current date);
```

00001
2024-11-10
2024-11-01
2024-11-30

```
194 -- Le 1er lundi du mois
195 VALUES CASE
196         WHEN (8 - DAYOFWEEK_ISO(FIRST_DAY(current date)) = 1) THEN FIRST_DAY(current date)
197         ELSE FIRST_DAY(current date) + (8 - DAYOFWEEK_ISO(FIRST_DAY(current date))) DAYS
198        END;
```

00001
2024-11-04

10. Fonctions table VALIDATE_DATA_xxx

- Fonctions **VALIDATE_DATA**, VALIDATE_DATA_FILE et VALIDATE_DATA_LIBRARY
  - Permet de vérifier que les colonnes numériques ne contiennent bien que des chiffres
  - Facile le passage de DDS à SQL (passage de PF à table)

Fichier K7P

Titre	Durée	Année	Genre
ANATOMIE D'UNE CHUTE	129	++++++	0

```
941 SELECT * FROM TABLE(systools.validate_data(library_name => 'SQLWS3',
942                                            file_name    => 'K7P'));
943
```

VALIDATE_TIME	LIBRARY_NAME	FILE_NAME	MEMBER_NAME	RELATIVE_RECORD_NUMBER
2024-03-08 15:21:48.850934	SQLWS3	K7P	K7P	1

SQL_WARNING	REASON_CODE	COLUMN_NAME	WARNING_TEXT
802	6	ANNEE	Erreur de conversion ou de mappage des données.

11. Fonction ANY_VALUE

- Permet de retourner une valeur arbitraire non nulle parmi un ensemble de valeurs

```
788 SELECT ANY_VALUE(nom) FROM tabempl;
```

00001
ANNIE

```
788 SELECT CHAR(srv) AS "Numéro de service",
789        CHAR(COUNT(*)) AS "Nombre d'employés",
790        ANY_VALUE(nom) AS "Exemple d'employé"
791  FROM tabempl WHERE srv IS NOT NULL
792  GROUP BY srv;
```

Numéro de service	Nombre d'employés	Exemple d'employé
911	3	MICHEL
977	2	DANIELE
990	1	CLAUDE

11. Fonction ANY_VALUE

- Intéressant pour optimiser le groupage
  - Evite d'avoir à grouper sur des colonnes à valeur unique ou d'utiliser les fonctions d'agrégation MIN ou MAX
  - Exemple : on part de ce SELECT ci-contre et on veut connaître le nombre de projets par employé

```
791 SELECT mat, nom, projet
792  FROM employes INNER JOIN projets ON mat=rsp
793  ORDER BY mat;
```

MAT	NOM	PROJET
20	MICHEL	DEV100
50	GEORGES	PRD510
50	GEORGES	DEV780
50	GEORGES	PRD112
70	EVE	PRD110
70	EVE	DEV200
90	HELENE	CPT410
110	VINCENT	ORG120
110	VINCENT	ORG330
120	JEAN	CPT113

  - **Sans** ANY_VALUE, solution **1** :

```
795 SELECT mat, nom, COUNT(projet) AS nb_proj
796  FROM employes INNER JOIN projets ON mat=rsp
797  GROUP BY mat, nom
798  ORDER BY mat;
```

MAT	NOM	NB_PROJ
20	MICHEL	1
50	GEORGES	3
70	EVE	2
90	HELENE	1
110	VINCENT	2
120	JEAN	1

- Si l'on souhaite afficher le nom de l'employé, la colonne "nom" dans le GROUP BY est obligatoire (mais non nécessaire car le nom est unique pour un employé)
- DB2 for i a besoin d'un index sur la colonne "nom"

11. Fonction ANY_VALUE

- **Sans** ANY_VALUE, solution **2** :

```
809 SELECT mat, MIN(nom), COUNT(projet) AS nb_proj
810  FROM employes INNER JOIN projets ON mat=rsp
811  GROUP BY mat
812  ORDER BY mat;
```

MAT	00002	NB_PROJ
20	MICHEL	1
50	GEORGES	3
70	EVE	2
90	HELENE	1
110	VINCENT	2
120	JEAN	1

- On utilise la fonction MIN (ou MAX) pour éviter d'indiquer la colonne "nom" dans le GROUP BY
- Traitement par DB2 pour récupérer MIN(nom)
- DB2 for i a besoin d'un index sur la colonne "nom"

- **Avec** ANY_VALUE :

```
814 SELECT mat, ANY_VALUE(nom), COUNT(projet) AS nb_proj
815 FROM employes INNER JOIN projets ON mat=rsp
816 GROUP BY mat
817 ORDER BY mat;
```

MAT	00002	NB_PROJ
20	MICHEL	1
50	GEORGES	3
70	EVE	2
90	HELENE	1
110	VINCENT	2
120	JEAN	1

- Traitement optimisé
- Pas besoin d'index sur la colonne "nom"

12. QCMDEXC en fonction scalaire

- Deux utilisations de **QCMDEXC** en **SQL** :
  - 1. **Procédure** stockée **QCMDEXC**
    - `CALL QCMDEXC('commande_CL'))`
    - Appelable uniquement par CALL – Ne peut pas être utilisée dans un SELECT
  - 2. **Fonction** scalaire (UDF) **QCMDEXC**
    - Peut être utilisée dans un SELECT
    - `SELECT z1, QCMDEXC('commande_CL')), Z2... FROM ...`
    - La commande CL est exécutée pour chacune des lignes du SELECT
    - Renvoie :
      - **1** si la commande CL s'est bien exécutée
      - **-1** si une erreur s'est produite

12. QCMDEXC en fonction scalaire

- Exemple

```
-- Création des répertoires /home/rpg4xx pour les profils RPG401 à RPG414

SELECT authorization_name,
       qsys2.qcmdexc('CRTDIR ''/home/' CONCAT TRIM(authorization_name) CONCAT '''') ,
       qsys2.qcmdexc('CHGOWN OBJ(''/home/' CONCAT TRIM(authorization_name) CONCAT ''')
                     NEWOWN(' CONCAT TRIM(authorization_name) CONCAT ')')
  FROM qsys2.user_info_basic
  WHERE authorization_name BETWEEN 'RPG401' AND 'RPG414';
```

12. QCMDEXC en fonction scalaire

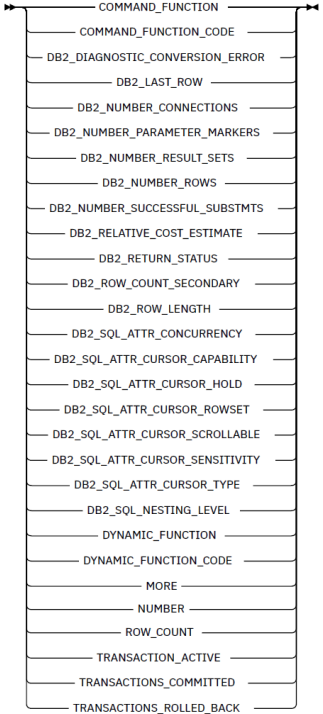
- Exemple

```
-- Suppression des fichiers dans /tmp qui n'ont pas été utilisés depuis au moins 2 ans

SELECT qsys2.qcmdexc('RMVLNK OBJLNK(''' CONCAT TRIM(Path_Name) CONCAT  ''')')
 FROM TABLE (ifs_object_statistics(START_PATH_NAME => '/tmp',
                                                      SUBTREE_DIRECTORIES => 'YES',
                                                      OBJECT_TYPE_LIST  => '*ALLSTMF'))
  WHERE last_used_timestamp < current timestamp - 2 years;
```

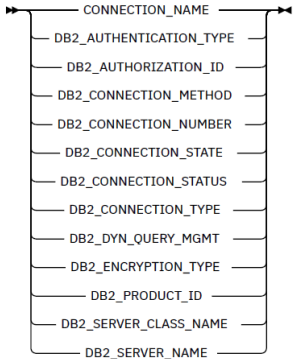
13. Fonction GET DIAGNOSTICS

- Permet de récupérer des informations sur l'instruction SQL qui vient d'être exécutée

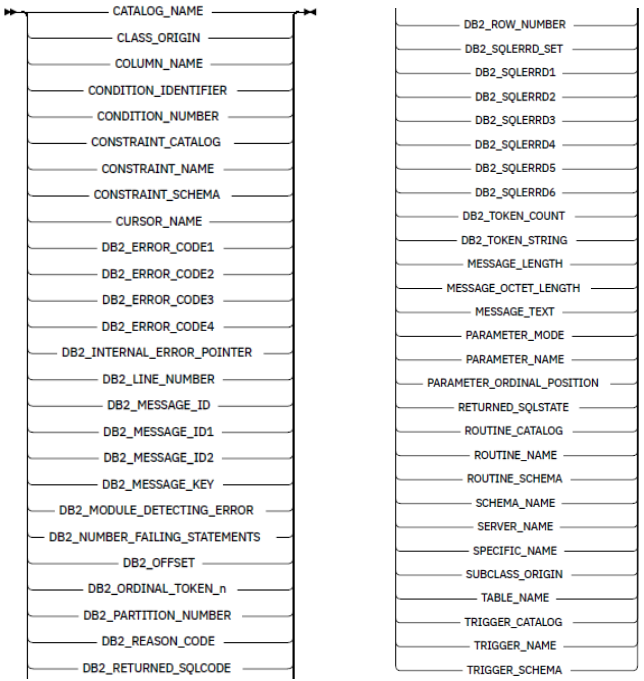
statement-information-item



connection-information-item



condition-information-item



13. Fonction GET DIAGNOSTICS

```
BEGIN
  DECLARE nbenr INT;
  DECLARE lgenr INT;
  UPDATE tabempl SET sal=sal*1.1 WHERE sx='F';
  GET DIAGNOSTICS nbenr=ROW_COUNT, lgenr=DB2_ROW_LENGTH;
  -- . . .
END;
```

14. SELF

- **SELF** (SQL Error Logging Facility) permet d'intercepter / tracer des erreurs SQL, y compris en production (moins consommateur que les moniteurs car n'intercepte que les instructions en erreur)

- Mise en œuvre :

  - 1. Enregistrement des codes d'erreur SQL à intercepter
    - En alimentant la variable globale **`SYSIBMADM.SELFCODES`**

  - 2. Exécution des applications

  - 3. Visualisation des erreurs
    - Par la vue QSYS2**`.SQL_ERROR_LOG`**

14. SELF

- Mise en œuvre :
  - **1**. Enregistrement des codes d'erreur SQL à intercepter :
    - Liste de SQLCODE
    - OU une valeur spéciale :
      - *ERROR : erreurs graves SQL (SQLCODE < 0)
      - *WARN : erreurs d'avertissement (SQLCODE > 0)
      - *ALL : les deux (*ERROR + *WARN)
      - *NONE : désactive SELF

```
-- Enregistrement des SQLCODE à monitorer - Niveau SYSTEME
CREATE OR REPLACE VARIABLE SYSIBMADM.SELFCODES VARCHAR(256) DEFAULT '-204, -501, -514';
VALUES SYSIBMADM.SELFCODES;

-- Enregistrement des SQLCODE à monitorer - Niveau JOB
SET SYSIBMADM.SELFCODES = '420';
VALUES SYSIBMADM.SELFCODES;
```

14. SELF

- Mise en œuvre :
  - **3**. Visualisation des erreurs par la vue QSYS2**.SQL_ERROR_LOG**

```
419 SELECT LOGGED_SQLCODE, LOGGED_SQLSTATE, NUMBER_OCCURRENCES, STATEMENT_TEXT,
420        STATEMENT_OPERATION_DETAIL, PROGRAM_LIBRARY, PROGRAM_NAME,
421        PROGRAM_TYPE, LOGGED_TIME, JOB_NAME, USER_NAME, CLIENT_APPLNAME
422 FROM QSYS2.SQL_ERROR_LOG;
423
```

SQLCODE BEING LOGGED	SQLSTATE CORRESPONDING TO SQLCODE	NUMBER OF OCCURRENCES	SQL STATEMENT TEXT	STATEMENT OPERATION DETAIL	LIBRARY CONTAINING PROGRAM NAME	PROGRAM ENCOUNTERING SQLCODE	OBJECT TYPE OF PROGRAM_NAME
-204	42704	2	SELECT z1, CAST(z1 AS INT) AS ...	PREPARE	QSYS	QZDASRV	*SRVPGM
420	01565	1	SELECT z1, CAST(z1 AS INT) AS ...	OPEN	QSYS	QZDASRV	*SRVPGM
-204	42704	1	OPEN FILMS	OPEN	RPG4COR	SQL1BFREEA	*PGM
-501	24501	1	FETCH FILMS INTO : H , : H	FETCH	RPG4COR	SQL1BFREEA	*PGM
-514	26501	1	OPEN FILMS	OPEN	RPG4COR	SQL2AFREEA	*PGM
-501	24501	1	FETCH FILMS INTO : H , : H	FETCH	RPG4COR	SQL2AFREEA	*PGM

MOST RECENT OCCURRENCE	MOST RECENT JOB	USER SPECIAL REGISTER	CURRENT CLIENT_APPLNAME SPECIAL REGISTER
2023-03-06 18:35:47.170000	691995/QUSER/QZDASOINIT	BOURGEOIS	IBM i Access Client Solutions - Run SQL Scripts
2023-03-06 18:37:03.927000	691995/QUSER/QZDASOINIT	BOURGEOIS	IBM i Access Client Solutions - Run SQL Scripts
2023-03-07 10:35:10.873000	695754/PB/QPADEV0002	PB	-
2023-03-07 10:35:10.920000	695754/PB/QPADEV0002	PB	-
2023-03-07 10:35:52.330000	695753/BOURGEOIS/QPADEV0001	BOURGEOIS	-
2023-03-07 10:35:52.406000	695753/BOURGEOIS/QPADEV0001	BOURGEOIS	-

15. Vues flexibles

- Par l'utilisation de variables globales

```
CREATE OR REPLACE VARIABLE as425f.numsrv NUM(3);

CREATE OR REPLACE VIEW as425f.empbysrv
 AS SELECT nom, sx, YEAR(CURRENT DATE - DAT_NAI) AS age, nomsrv
    FROM tabempl INNER JOIN tabserv USING(srv)
    WHERE srv = as425f.numsrv;

SET as425f.numsrv=911;
SELECT * FROM as425f.empbysrv;

SET as425f.numsrv=977;
SELECT * FROM as425f.empbysrv;
```

NOM	SX	AGE	NOMSRV
MICHEL	M	69	VENTES
ANNIE	F	65	VENTES
JACQUES	M	67	VENTES

NOM	SX	AGE	NOMSRV
DANIELE	F	62	MANUFACT
MARC	M	52	MANUFACT

16. Gestion du CURRENT PATH

- Rappel : le **CURRENT PATH** est utilisé lors de la recherche non qualifiée :
  - Des procédures (CALL **proc1** …)
  - Des fonctions (SELECT nomemp, **age**(datenai) …)
  - Des variables globales (SET **ifs_path** = …)

- Manipulation du CURRENT PATH

```
-- Visualisation du CURRENT PATH
VALUES CURRENT PATH;
-- Si NAMING(*SYS) : *LIBL
-- Si NAMING(*SQL) : QSYS, QSYS2, SYSPROC, SYSIBMADM, bib_portant_le_nom_du_USER

-- Définition d'un nouveau CURRENT PATH
SET PATH pb, video; -- PB, VIDEO
```

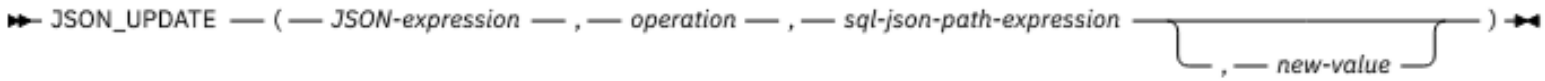
16. Gestion du CURRENT PATH

```
-- Ajout d'une bibliothèque au PATH actuel
SET PATH CURRENT PATH, as425f; -- PB, VIDEO, AS425F

-- Définition d'un CURRENT PATH constitué de la partie système du PATH
--                                + nos propres biliothèques
SET PATH SYSTEM PATH, as425f, pb; -- QSYS, QSYS2, SYSPROC, SYSIBMADM, AS425F, PB

-- Pour revenir au PATH par défaut
SET PATH SYSTEM PATH, USER;
-- Si NAMING(*SYS) : *LIBL
-- Si NAMING(*SQL) : QSYS, QSYS2, SYSPROC, SYSIBMADM, bib_portant_le_nom_du_USER

-- Pour définir *LIBL comme CURRENT PATH quelque soit la valeur de NAMING
SET PATH *LIBL;
```

17. Fonction JSON_UPDATE

- Permet de **modifier** le contenu d'une expression **JSON**

- Syntaxe :



- Avec :
  - `JSON-expression` :
    - Expression JSON à modifier (constante, variable, colonne DB2, etc.)
  - `operation` :
    - SET : ajout ou modification d'une valeur (ajout si la valeur indiquée n'existe pas)
    - REMOVE : suppression d'une valeur (par clé / poste de tableau)
  - `sql-json-path-expression` :
    - Pour localiser l'élément à mettre à jour ou à supprimer (syntaxe *strict* uniquement)
  - `new-value` :
    - Nouvelle valeur pour insertion ou mise à jour

17. Fonction JSON_UPDATE

```
CREATE OR REPLACE VARIABLE json_var VARCHAR(1000);

SET json_var = '{ "films": [
   { "titre" : "MATRIX", "annee" : 1979, "acteurs" : ["REEVES", "MOSS"] },
   { "titre" : "SPIDERMAN", "annee" : 2002, "acteurs" : ["MAGUIRE", "DUNST", "DAFOE"] },
   { "titre" : "INCEPTION", "annee" : 2010, "acteurs" : ["DI CAPRIO", "COTILLARD"] },
   { "titre" : "GLADIATOR", "annee" : 2000, "acteurs" : ["CROWE", "PHOENIX"] } ] }';
VALUES json_var;
```

```
{"films":[{"titre":"MATRIX","annee":1979,"acteurs":["REEVES","MOSS"]},{"titre":"SPIDERMAN","annee":20
```

```
SET json_var = JSON_UPDATE( json_var, 'SET', '$.films[0].annee', '1999');
VALUES json_var;
```

```
{"films":[{"titre":"MATRIX","annee":1999,"acteurs":["REEVES","MOSS"]},{"titre":"SPIDERMAN","annee":20
```

```
SET json_var = JSON_UPDATE( json_var, 'SET', '$.films[2].acteurs[2]', 'GORDON-LEVITT');
VALUES json_var;
```

```
},{"titre":"INCEPTION","annee":2010,"acteurs":["DI CAPRIO","COTILLARD","GORDON-LEVITT"]},{"titre":
```

```
SET json_var = JSON_UPDATE( json_var, 'REMOVE', '$.films[2].acteurs[2]');
VALUES json_var;
```

```
},{"titre":"INCEPTION","annee":2010,"acteurs":["DI CAPRIO","COTILLARD"]},{"titre
```

```
SET json_var = JSON_UPDATE( json_var, 'SET', '$.films[0].duree', '136');
VALUES json_var;
```

```
{"films":[{"titre":"MATRIX","annee":1979,"acteurs":["REEVES","MOSS"],"duree":136},{"titre":"SPIDERM...
```

```
SET json_var = JSON_UPDATE( json_var, 'REMOVE', '$.films[0].duree');
VALUES json_var;
```

```
{"films":[{"titre":"MATRIX","annee":1979,"acteurs":["REEVES","MOSS"]},{"titre":"SPIDERMAN","annee"...
```

18. La vue SYSFILES de QSYS2

- Informations détaillées sur les fichiers DB2

```
-- PFs avec clé UNIQUE
SELECT * FROM qsys2.sysfiles WHERE table_schema='xxx'
                                AND native_type='PHYSICAL'
                                AND file_type = 'DATA'
                                AND sql_object_type IS NULL
                                AND access_path_keyed ='YES'
                                AND access_path_type='KEYED UNIQUE';
```

```
-- PFs multi-membres
SELECT * FROM qsys2.sysfiles WHERE table_schema='xxx'
                                AND native_type='PHYSICAL'
                                AND file_type = 'DATA'
                                AND sql_object_type IS NULL
                                AND number_members > 1;
```

```
-- Tables avec clé primaire
SELECT * FROM qsys2.sysfiles WHERE table_schema='xxx'
                                AND native_type='PHYSICAL'
                                AND file_type = 'DATA'
                                AND sql_object_type = 'TABLE'
                                AND primary_key = 'YES';
```

- Fichiers avec colonnes de type DATE/TIME, LOB, IDENTITY, Unicode…
- Fichiers en REUSEDLT(*NO)
- LFs qui ont des fichiers dépendants
- etc.

```
-- LFs  avec S/O
SELECT * FROM qsys2.sysfiles WHERE table_schema='xxx'
                                AND native_type='LOGICAL'
                                AND sql_object_type IS NULL
                                AND select_omit='YES';
```

19. Utilisation des noms SQL en 3 parties

- **Noms qualifiés en 3 parties**
  - `Instruction_SQL  nom_de_la_base_de_données_distante.schéma.objet_DB2`
  - Nom de la base de données distante : cf commande **WRKRDBDIRE**
  - Enregistrement des profil et mot de passe : cf commande **ADDSVRAUTE**

19. Utilisation des noms SQL en 3 parties

```sql
-- Lecture de données sur une table distante
SELECT * FROM k70ab740.pb.employes;
SELECT * FROM k70ab740.qsys2.user_info;

-- Ajout de données sur une table locale à partir d'une table distante
INSERT INTO pb.emplocal (SELECT * FROM k70ab740.pb.employes);
CREATE TABLE pb.emplocal AS (SELECT * FROM k70ab740.pb.employes) WITH DATA;

-- Gestion d'objets DB2 distant (schéma, table, vue, index, variable globale...)
CREATE TABLE k70ab740.pb.empdist (nom CHAR(20), actif BOOLEAN);
INSERT INTO k70ab740.pb.empdist VALUES ('MICHEL', TRUE), ('JACQUES', FALSE);
UPDATE k70ab740.pb.empdist SET actif=TRUE WHERE nom='JACQUES';
DELETE k70ab740.pb.empdist WHERE nom='JACQUES';
DROP TABLE k70ab740.pb.empdist;

-- Exécution d'une fonction table (UDTF) distante
SELECT * FROM REMOTE TABLE(k70ab740.qsys2.job_info(JOB_TYPE_FILTER => '*INTERACT'));

-- Exécution d'une commande CL distante
CALL k70ab740.qsys2.qcmdexc('CRTDTAARA DTAARA(PB/DTAARA1) TYPE(*CHAR) LEN(10)');
```

20. ACS : SELECT FOR UPDATE

- Permet de mettre à jour les valeurs du SELECT directement dans le résultat
  - Pas d'ajout, pas de suppression

```
369  SELECT * FROM tabempl FOR UPDATE;
```

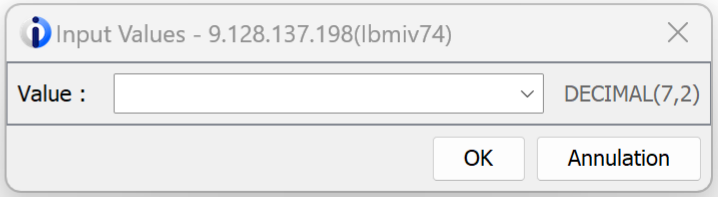
MAT	NOM	SX	SRV	SAL	DAT_NAI
20	MICHEL	M	911	2140.00	1955-03-17
10	ANNIE	F	911	3080.00	1959-05-12
50	JACQUES	M	911	2360.00	1956-11-11
40	DANIELE	F	977	2810.00	1962-07-13
30	MARC	M	977	3570.00	1972-02-28
60	CLAUDE	M	990	2210.00	1988-05-02
70	RICHARD	M	-	-	1956-03-01

Double-clic pour pouvoir éditer le contenu

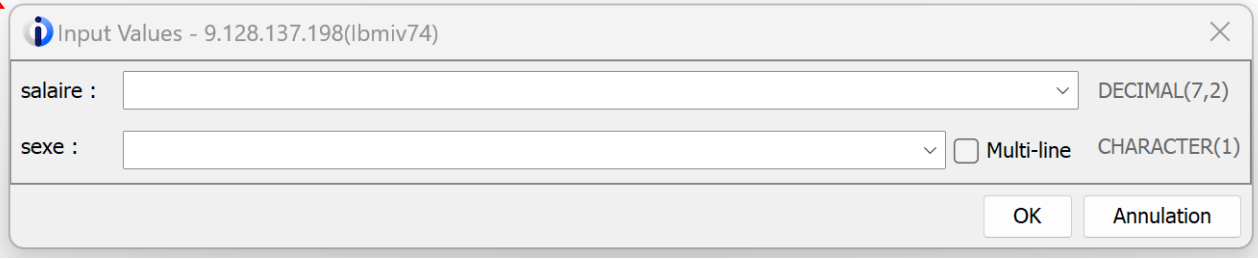
21. ACS : invite sur variables

- Par utilisation de marqueurs (**?** ou **:variable**)

```
SELECT * FROM tabempl WHERE sal > ?;
```



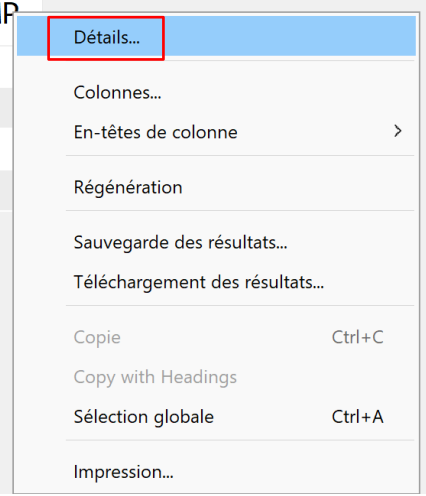

```
SELECT * FROM tabempl WHERE sal > :salaire AND sx = :sexe;
```



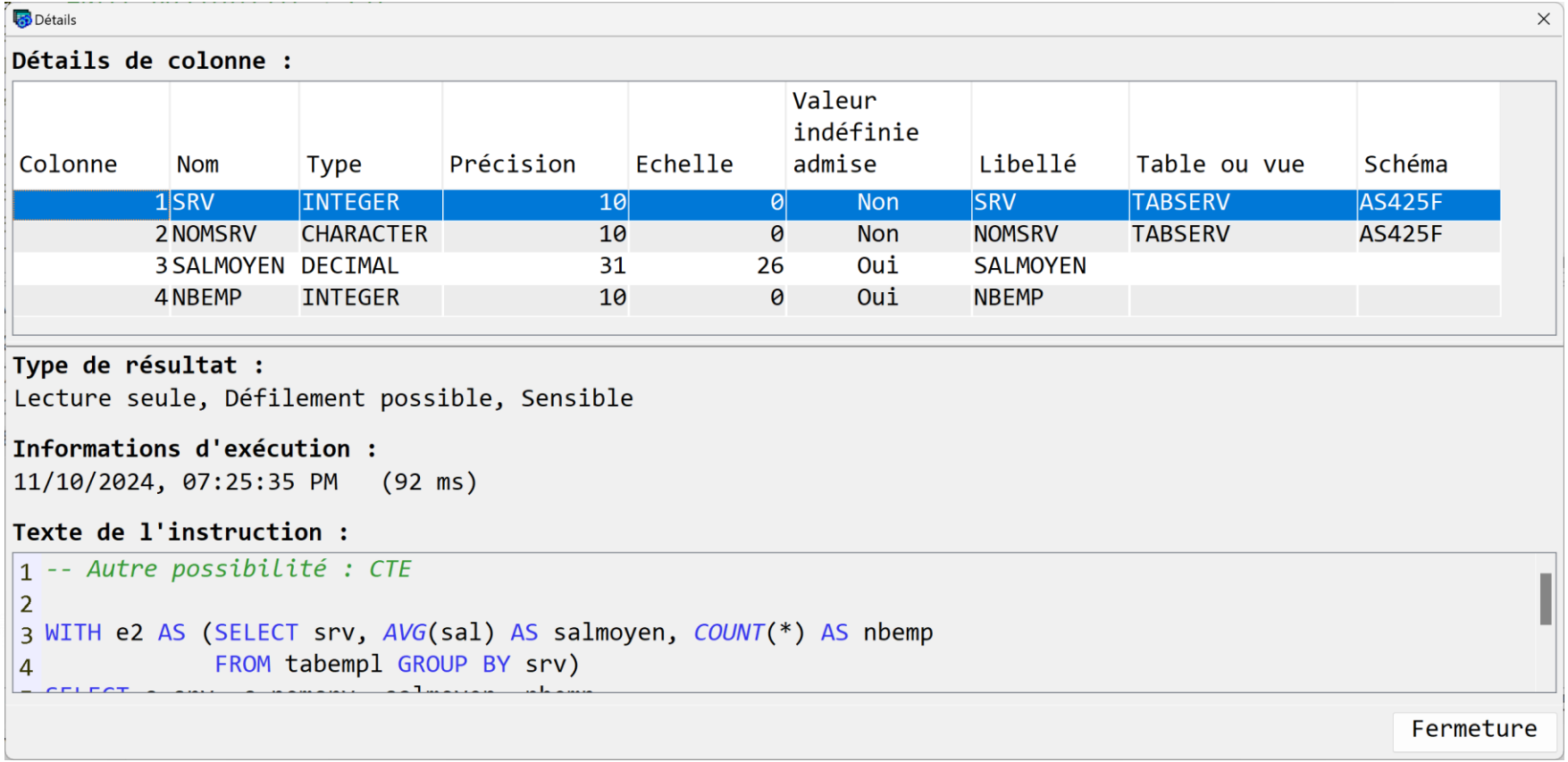
22. ACS : métadonnées étendues

```
111 WITH e2 AS (SELECT srv, AVG(sal) AS salmoyen, COUNT(*) AS nbemp
112             FROM tabempl GROUP BY srv)
113 SELECT s.srv, s.nomsrv, salmoyen, nbemp
114  FROM tabserv s
115  LEFT OUTER JOIN e2
116  ON s.srv = e2.srv;
```

SRV	NOMSRV	SALMOYEN	NBEMP
901	COMPTA	-	
977	MANUFACT	3190.000000000000000000000000	
911	VENTES	2526.666666666666666666666666	
982	APRES VTES	-	

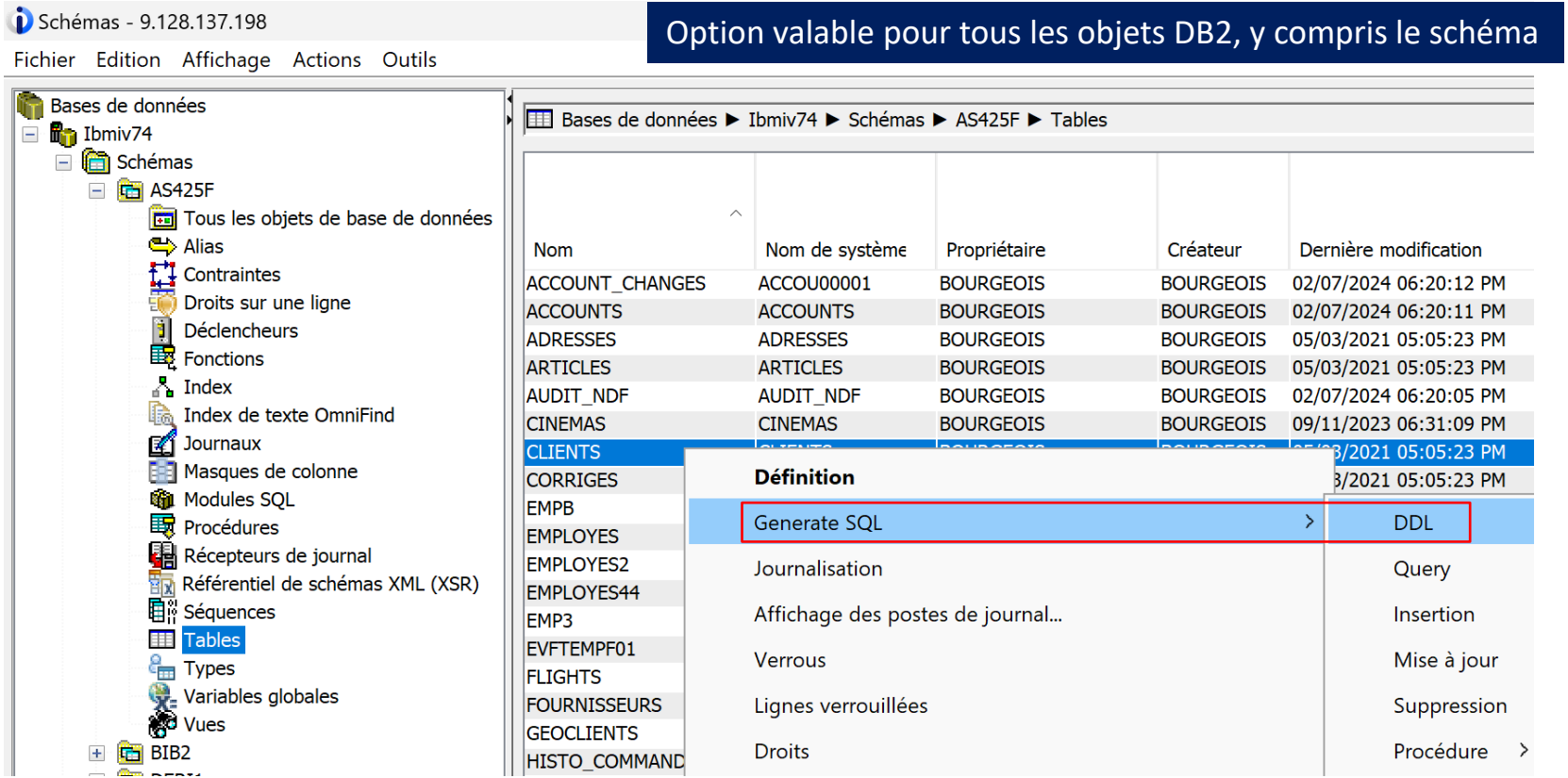


22. ACS : métadonnées étendues

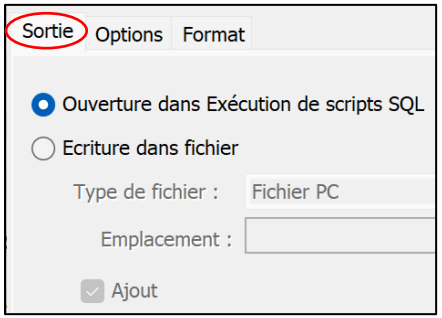


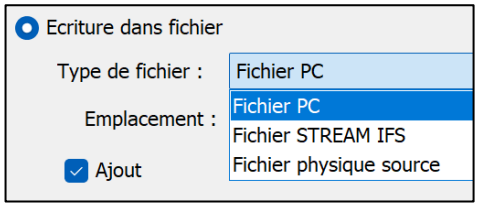
23. ACS : génération du SQL DDL

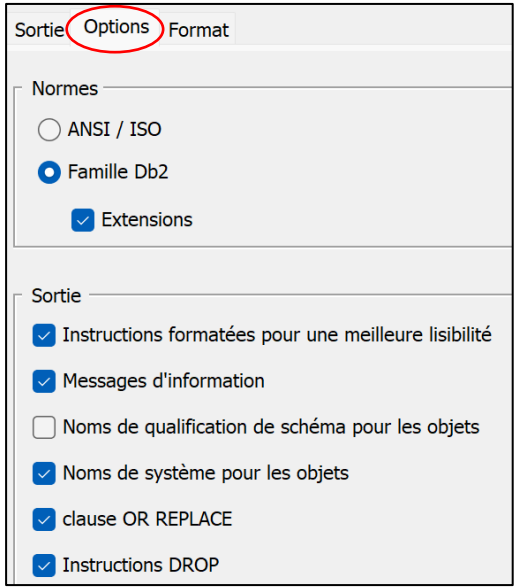
Option valable pour tous les objets DB2, y compris le schéma

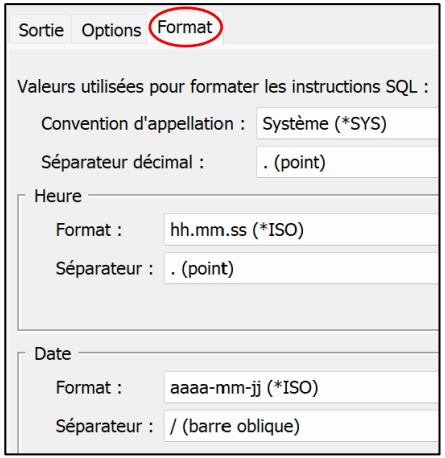


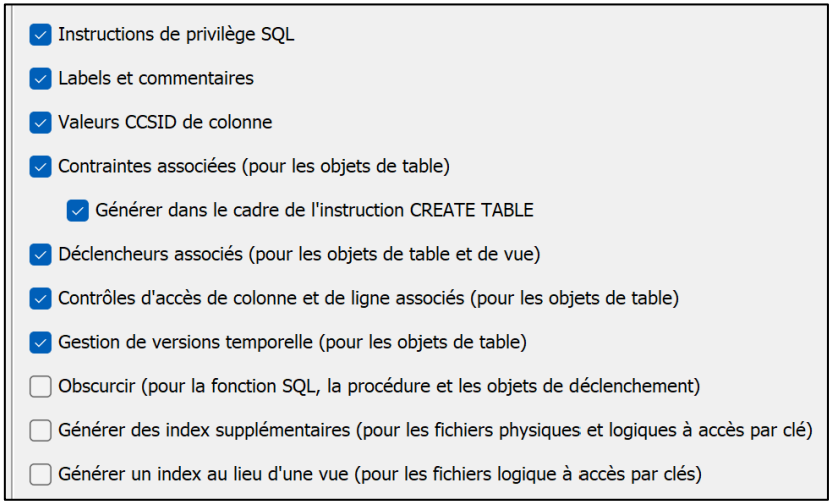
23. ACS : génération du SQL DDL



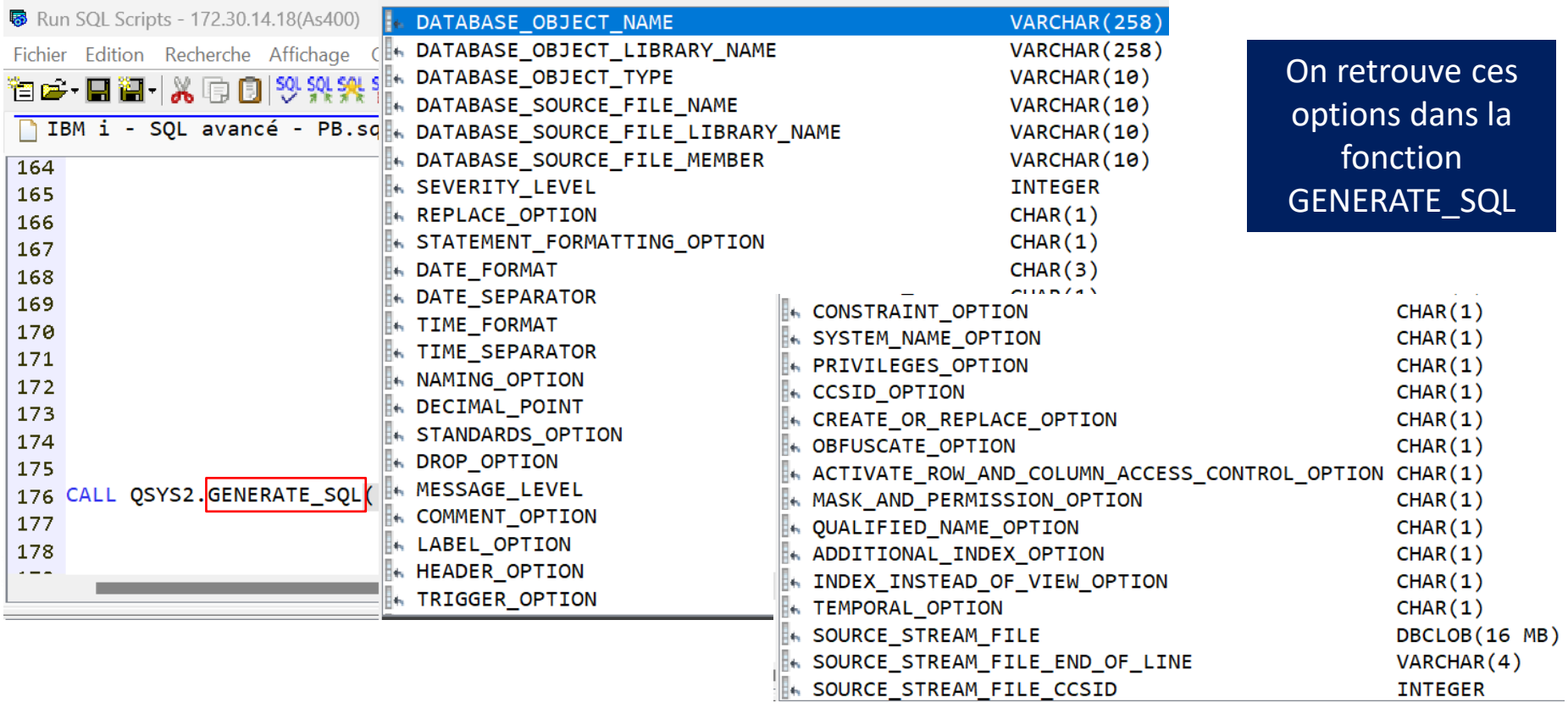








23. ACS : génération du SQL DDL



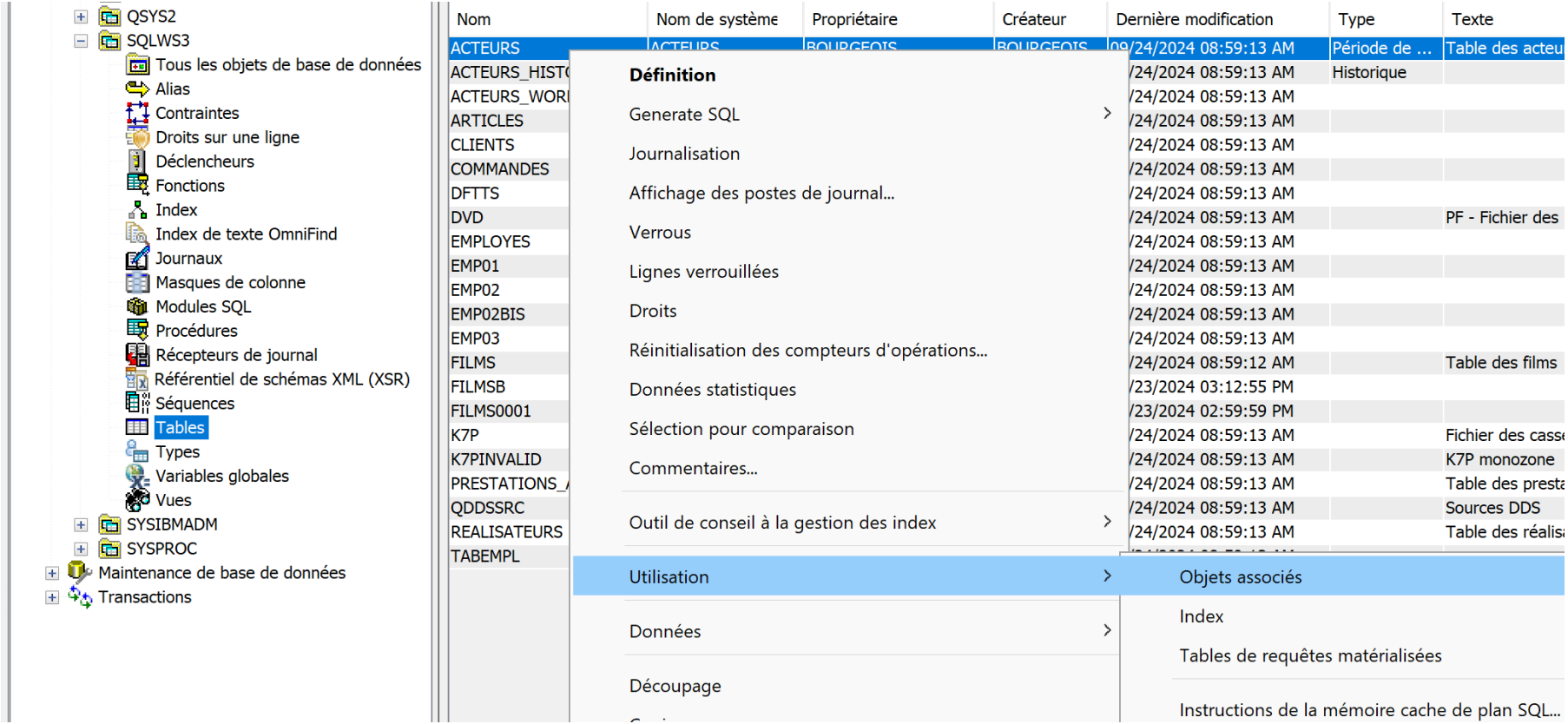
On retrouve ces options dans la fonction GENERATE_SQL

23. ACS : génération du SQL DDL

- Résultat

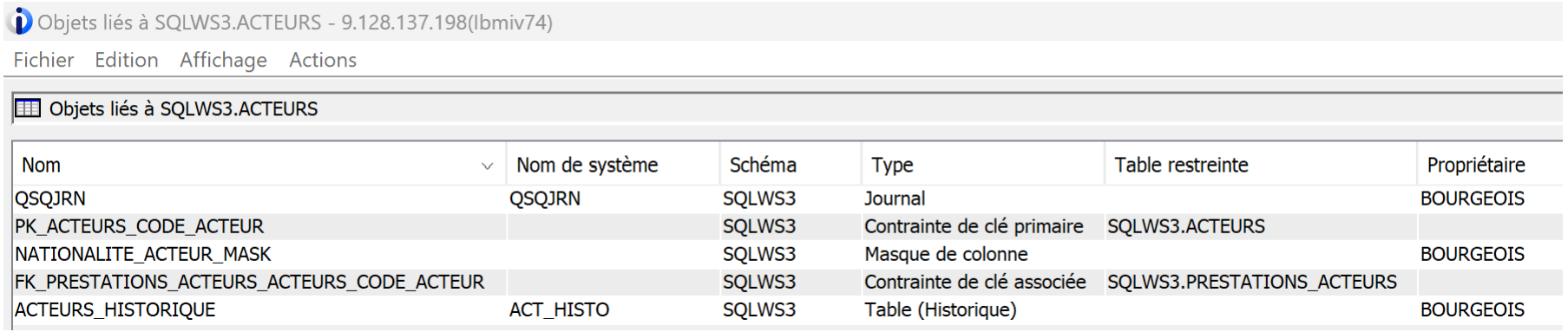
```
1  --  Générer SQL
2  --  Version :                    V7R4M0 190621
3  --  Générée le :               11/11/24 11:35:18
4  --  Base données relation :    IBMIV74
5  --  Option normes :            Db2 for i
6  DROP TABLE IF EXISTS AS425F/CLIENTS ;
7
8  CREATE OR REPLACE TABLE AS425F/CLIENTS (
9      CLIENT_ID SMALLINT GENERATED ALWAYS AS IDENTITY (
10     START WITH 1 INCREMENT BY 1
11     NO MINVALUE NO MAXVALUE
12     NO CYCLE NO ORDER
13     CACHE 20 )
14     IMPLICITLY HIDDEN ,
15     NOM VARCHAR(50) CCSID 1147 DEFAULT NULL ,
16     CONSTRAINT AS425F/Q_AS425F_CLIENTS_CLIENT_ID_00001 PRIMARY KEY( CLIENT_ID ) )
17
18     RCDFMT CLIENTS    ;
19
20 GRANT ALTER , DELETE , INDEX , INSERT , REFERENCES , SELECT , UPDATE
21 ON AS425F/CLIENTS TO BOURGEOIS WITH GRANT OPTION ;
```

24. ACS : affichage des objets dépendants



24. ACS : affichage des objets dépendants

- Résultat

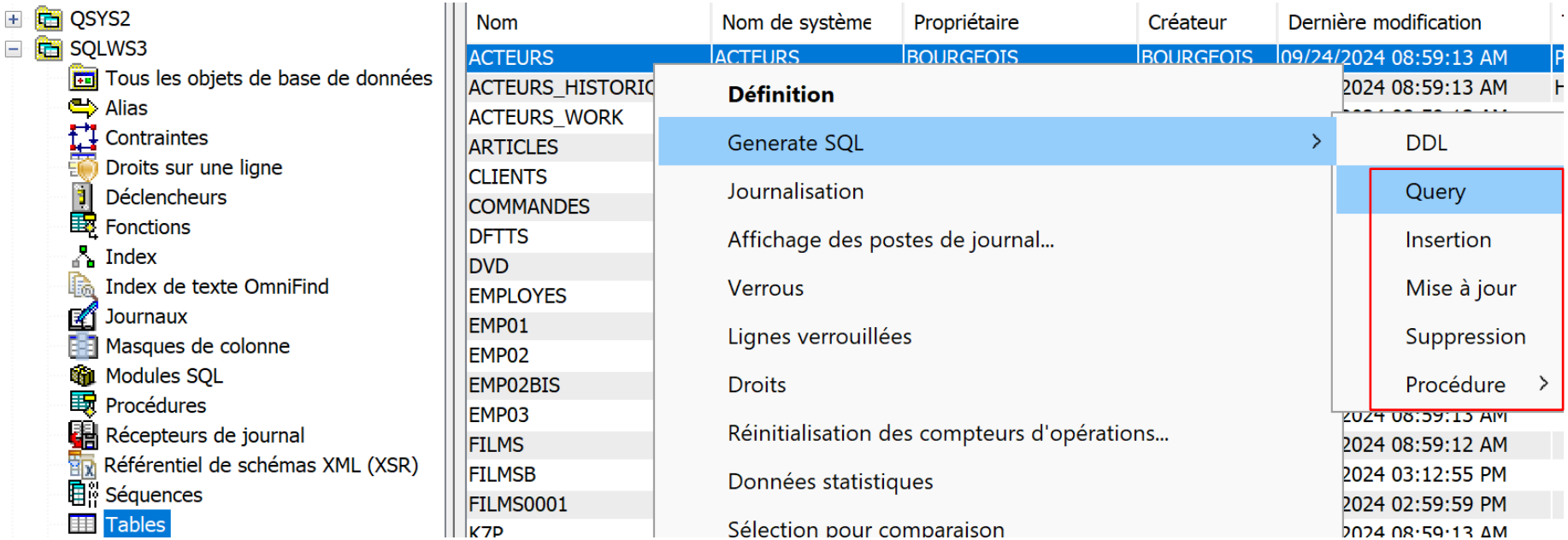


Nom	Nom de système	Schéma	Type	Table restreinte	Propriétaire
QSQJRN	QSQJRN	SQLWS3	Journal		BOURGEOIS
PK_ACTEURS_CODE_ACTEUR		SQLWS3	Contrainte de clé primaire	SQLWS3.ACTEURS	
NATIONALITE_ACTEUR_MASK		SQLWS3	Masque de colonne		BOURGEOIS
FK_PRESTATIONS_ACTEURS_ACTEURS_CODE_ACTEUR		SQLWS3	Contrainte de clé associée	SQLWS3.PRESTATIONS_ACTEURS	
ACTEURS_HISTORIQUE	ACT_HISTO	SQLWS3	Table (Historique)		BOURGEOIS

```
189 SELECT * FROM TABLE(SYSTOOLS.RELATED_OBJECTS(LIBRARY_NAME=>'SQLWS3',
190                                              FILE_NAME =>'ACTEURS'));
```

SOURCE_SCHEMA_NAME	SOURCE_SQL_NAME	SQL_OBJECT_TYPE	SCHEMA_NAME	SQL_NAME
SQLWS3	ACTEURS	FOREIGN KEY	SQLWS3	PRESTATIONS_ACTEURS
SQLWS3	ACTEURS	HISTORY TABLE	SQLWS3	ACTEURS_HISTORIQUE
SQLWS3	ACTEURS	MASK	SQLWS3	NATIONALITE_ACTEUR_MASK

25. ACS : génération du SQL DML



25. ACS : génération du SQL DML

- Résultat

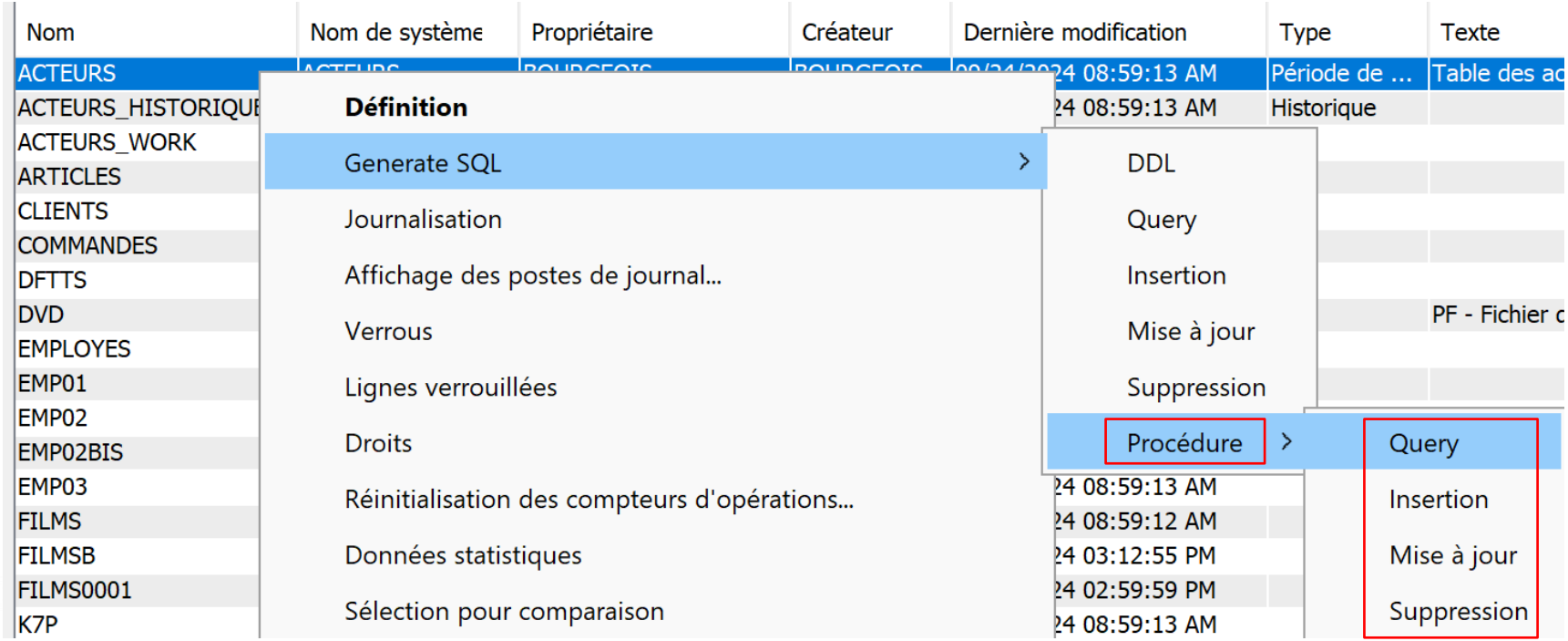
```
SELECT
  CODE_ACTEUR,             /* CODE_ACT    INTEGER          */
  NOM_ACTEUR,              /* NOM_ACT     VARCHAR(30)      */
  PRENOM_ACTEUR,           /* PRENOM_ACT  VARCHAR(30)      */
  NATIONALITE_ACTEUR,      /* NAT_ACT     CHARACTER(5)     */
  DATE_NAISSANCE_ACTEUR,   /* DATNAI_ACT  DATE             */
  DATE_OPERATION,          /* DATE_OPER   TIMESTAMP        */
  TYPE_OPERATION,          /* TYPE_OPER   CHARACTER(1)     */
  UTILISATEUR_OPERATION,   /* UTIL_OPER   VARCHAR(128)     */
  DEBUT_CYCLE,             /* DEBCYCLE    TIMESTAMP(12)    */
  FIN_CYCLE,               /* FINCYCLE    TIMESTAMP(12)    */
  DEBUT_TRANSACTION        /* DEBTRANSAC  TIMESTAMP(12)    */
FROM SQLWS3.ACTEURS;
```

QUERY

INSERT

```
INSERT INTO SQLWS3.ACTEURS (
  CODE_ACTEUR,             /* CODE_ACT    INTEGER          */
  NOM_ACTEUR,              /* NOM_ACT     VARCHAR(30)      */
  PRENOM_ACTEUR,           /* PRENOM_ACT  VARCHAR(30)      */
  NATIONALITE_ACTEUR,      /* NAT_ACT     CHARACTER(5)     */
  DATE_NAISSANCE_ACTEUR,   /* DATNAI_ACT  DATE             */
  DATE_OPERATION,          /* DATE_OPER   TIMESTAMP        */
  TYPE_OPERATION,          /* TYPE_OPER   CHARACTER(1)     */
  UTILISATEUR_OPERATION,   /* UTIL_OPER   VARCHAR(128)     */
  DEBUT_CYCLE,             /* DEBCYCLE    TIMESTAMP(12)    */
  FIN_CYCLE,               /* FINCYCLE    TIMESTAMP(12)    */
  DEBUT_TRANSACTION        /* DEBTRANSAC  TIMESTAMP(12)    */
)
VALUES (
  :CODE_ACTEUR,            /* INTEGER        Aucune valeur par défaut                          */
  :NOM_ACTEUR,             /* VARCHAR(30)    Aucune valeur par défaut                          */
  :PRENOM_ACTEUR,          /* VARCHAR(30)    Aucune valeur par défaut                          */
  :NATIONALITE_ACTEUR,     /* CHARACTER(5)   Aucune valeur par défaut                          */
  :DATE_NAISSANCE_ACTEUR,  /* DATE           Aucune valeur par défaut                          */
  DEFAULT,                 /* TIMESTAMP      Valeur générée : Changement de ligne              */
  DEFAULT,                 /* CHARACTER(1)   Valeur générée : Opération de changement de données */
  DEFAULT,                 /* VARCHAR(128)   Valeur générée : SESSION_USER                     */
  DEFAULT,                 /* TIMESTAMP(12)  Valeur générée : Début de ligne                   */
  DEFAULT,                 /* TIMESTAMP(12)  Valeur générée : Fin de ligne                     */
  DEFAULT                  /* TIMESTAMP(12)  Valeur générée : ID de début de transaction       */
);
```

25. ACS : génération du SQL DML



25. ACS : génération du SQL DML

- Résultat

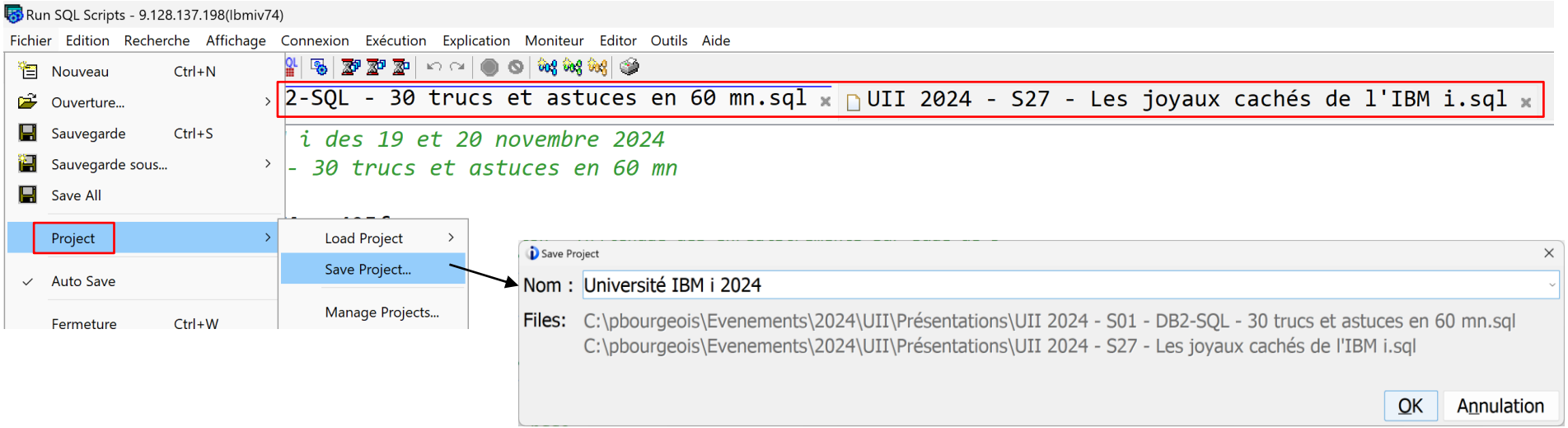
```
1  CREATE OR REPLACE PROCEDURE SQLWS3.SELECT_FROM_ACTEURS ()
2  READS SQL DATA
3  PROGRAM TYPE SUB
4  DYNAMIC RESULT SETS 1
5  SPECIFIC SQLWS3.SELECT_FROM_ACTEURS
6  SET OPTION
7    COMMIT = *NONE,
8    DYNUSRPRF = *USER,
9    USRPRF = *USER
10
11 BEGIN
12   DECLARE RS CURSOR WITH RETURN FOR
13     SELECT
14       CODE_ACTEUR,             /* CODE_ACT    INTEGER          */
15       NOM_ACTEUR,              /* NOM_ACT     VARCHAR(30)      */
16       PRENOM_ACTEUR,           /* PRENOM_ACT  VARCHAR(30)      */
17       NATIONALITE_ACTEUR,      /* NAT_ACT     CHARACTER(5)     */
18       DATE_NAISSANCE_ACTEUR,   /* DATNAI_ACT  DATE             */
19       DATE_OPERATION,          /* DATE_OPER   TIMESTAMP        */
20       TYPE_OPERATION,          /* TYPE_OPER   CHARACTER(1)     */
21       UTILISATEUR_OPERATION,   /* UTIL_OPER   VARCHAR(128)     */
22       DEBUT_CYCLE,             /* DEBCYCLE    TIMESTAMP(12)    */
23       FIN_CYCLE,               /* FINCYCLE    TIMESTAMP(12)    */
24       DEBUT_TRANSACTION        /* DEBTRANSAC  TIMESTAMP(12)    */
25     FROM SQLWS3.ACTEURS;
```

```
26   DECLARE EXIT HANDLER FOR SQLEXCEPTION
27     BEGIN
28       DECLARE LOCAL_SQLCODE INTEGER;
29       DECLARE LOCAL_SQLSTATE CHAR(5) FOR SBCS DATA;
30       DECLARE LOCAL_MESSAGE_TEXT VARCHAR(200) FOR SBCS DATA;
31       DECLARE JOBLOG_MESSAGE VARCHAR(1000) FOR SBCS DATA;
32       GET DIAGNOSTICS CONDITION 1
33         LOCAL_SQLCODE = DB2_RETURNED_SQLCODE,
34         LOCAL_SQLSTATE = RETURNED_SQLSTATE,
35         LOCAL_MESSAGE_TEXT = MESSAGE_TEXT;
36       SET JOBLOG_MESSAGE = 'SQLWS3.SELECT_FROM_ACTEURS() failed.
37         CONCAT ' SQLCODE=' CONCAT LOCAL_SQLCODE
38         CONCAT ' SQLSTATE=' CONCAT LOCAL_SQLSTATE
39         CONCAT ' MESSAGE_TEXT=' CONCAT LOCAL_MESSAGE_TEXT;
40       CALL SYSTOOLS.LPRINTF(JOBLOG_MESSAGE);
41     END;
42
43   OPEN RS;
44 END;
45
46 GRANT ALTER, EXECUTE
47 ON SPECIFIC PROCEDURE SQLWS3.SELECT_FROM_ACTEURS
48 TO BOURGEOIS WITH GRANT OPTION;
```

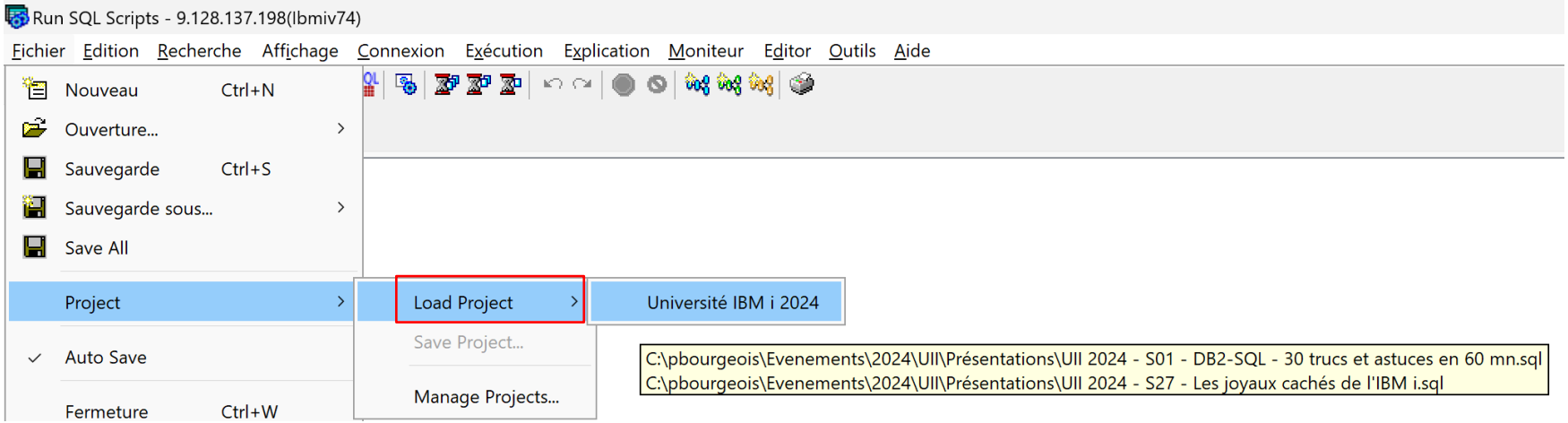
```
49
50 STOP;
51
52 CALL SQLWS3.SELECT_FROM_ACTEURS ();
```

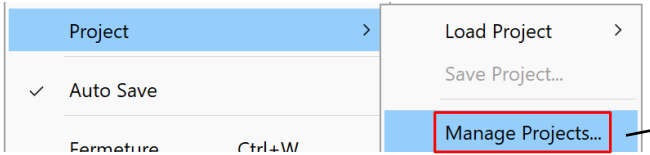
26. ACS : projets SQL

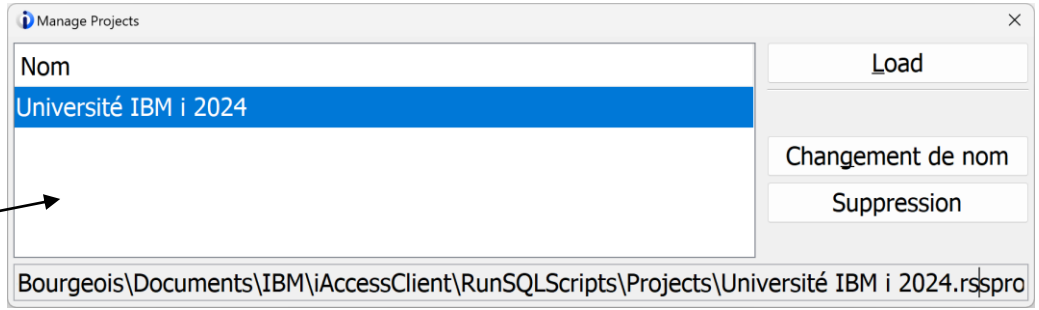
- **Projet** : Regroupement de scripts SQL qui se trouvent :
  - Sur le PC (.sql)
  - Et/ou dans l'IFS (.sql)
  - Et/ou dans des membres source de fichiers source



26. ACS : projets SQL







27. VS Code for i : sorties JSON et CSV

```
3   json: SELECT * FROM as425F.employes;
4
5   csv: SELECT * FROM as425F.employes;
```

Untitled-1 | {} Untitled-2

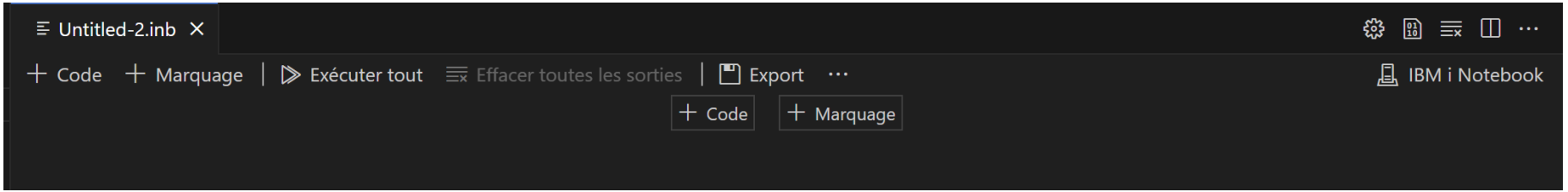
```
1  [
2    {
3      "MAT": 10,
4      "NOM": "CHRISTIAN",
5      "SRV": "A00",
6      "POSTE": "DIR. GENERAL",
7      "SX": "M",
8      "SALAIR": 6728,
9      "PRIME": 100,
10     "DATNAI": null
11   },
12   {
13     "MAT": 110,
14     "NOM": "VINCENT",
15     "SRV": "B00",
16     "POSTE": "DIR. ADJOINT",
```

Untitled-1 | Untitled-2

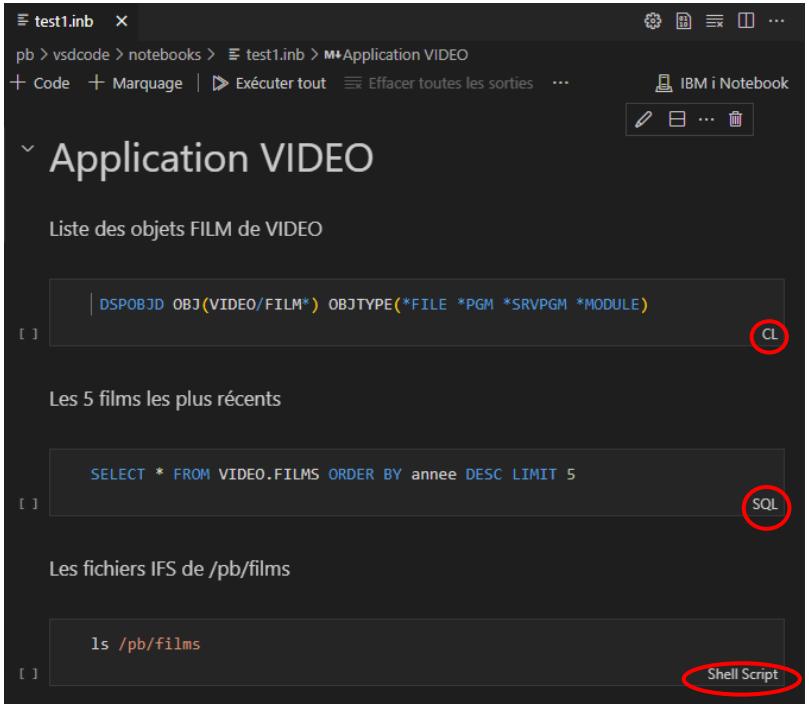
```
1  "MAT","NOM","SRV","POSTE","SX","SALAIR","PRIME","DATNAI"
2  10,"CHRISTIAN","A00","DIR. GENERAL","M",6728,100,
3  110,"VINCENT","B00","DIR. ADJOINT","M",5514,30,"21/07/47"
4  120,"JEAN","E00","COMPTABLE","M",3192,60,"04/01/52"
5  20,"MICHEL","C00","CHEF SERVICE","M",4478,80,"01/03/56"
6  30,"JEANNE","C01","CHEF SERVICE","F",4328,80,"17/09/53"
7  130,"DOLORES","C01","ANALYSTE","F",3700,50,"06/12/61"
```

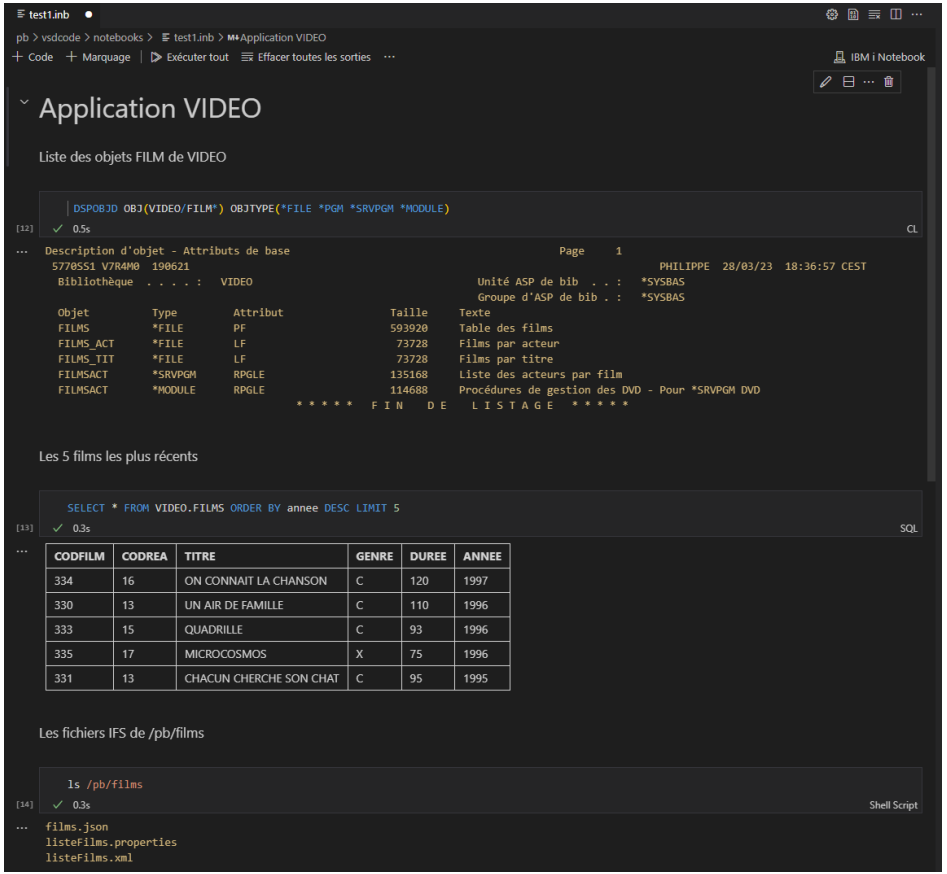
28. VS Code for i : notebooks

- **Notebook VS Code**
  - Fichier de type .INB
  - Mélange de texte et de code exécutable (CL, SQL, scripts SHELL)
  - Possibilité de générer des graphiques (sorties BAR, LINE, PIE)
  - Export HTML



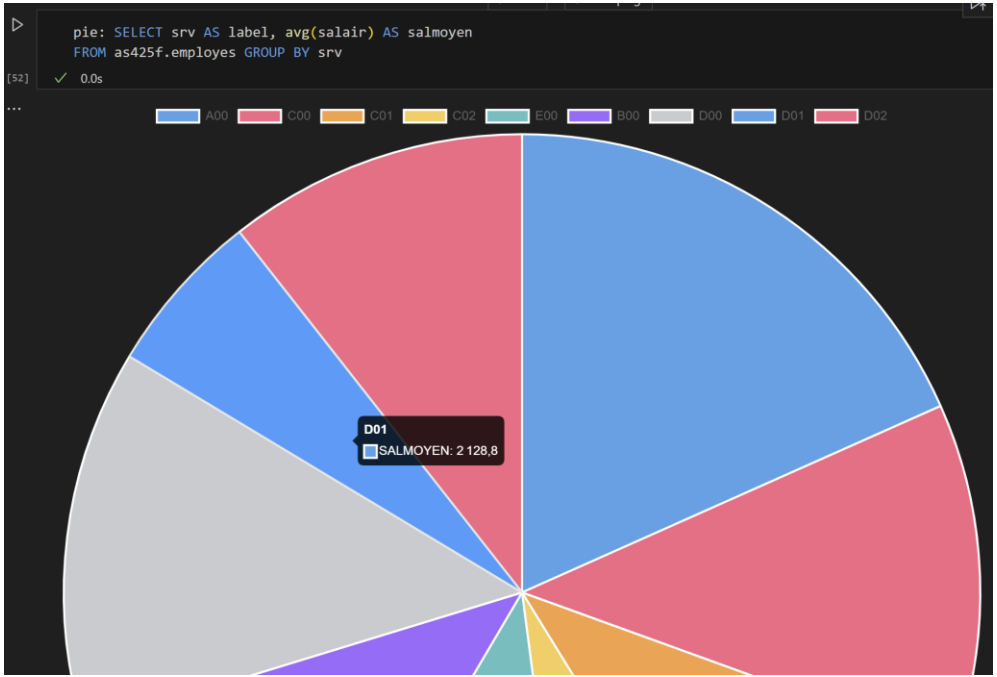
28. VS Code for i : notebooks

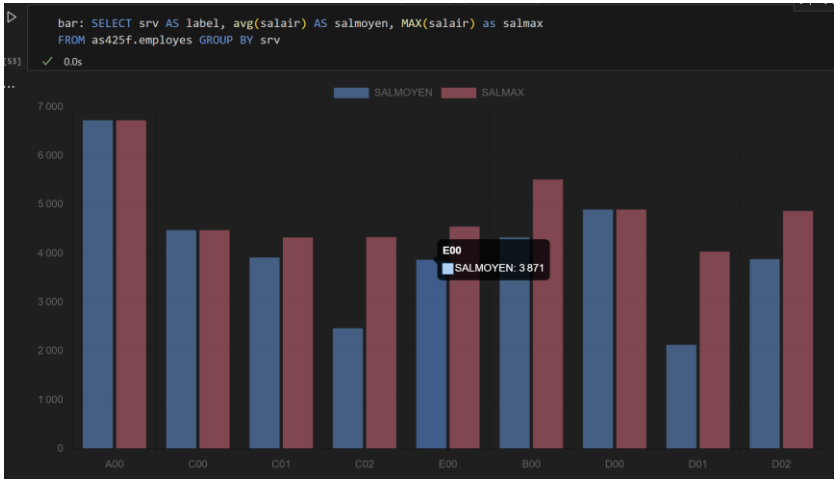


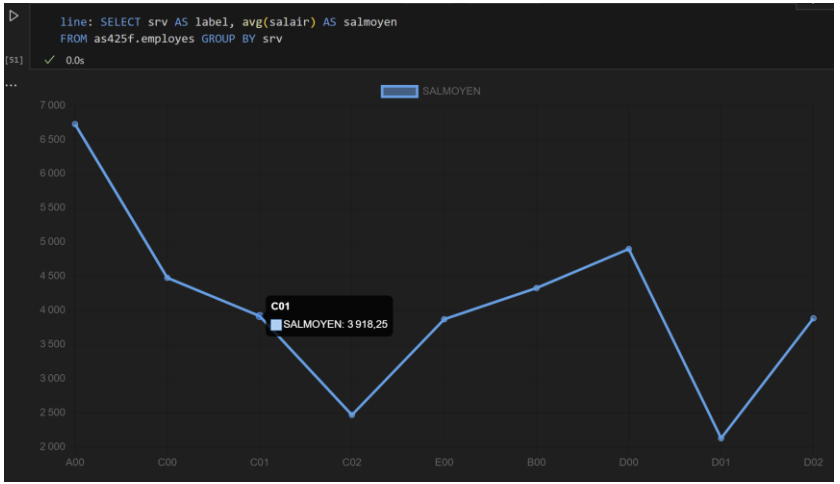


28. VS Code for i : notebooks

- Sorties **graphiques** (PIE, LINE, BAR…)





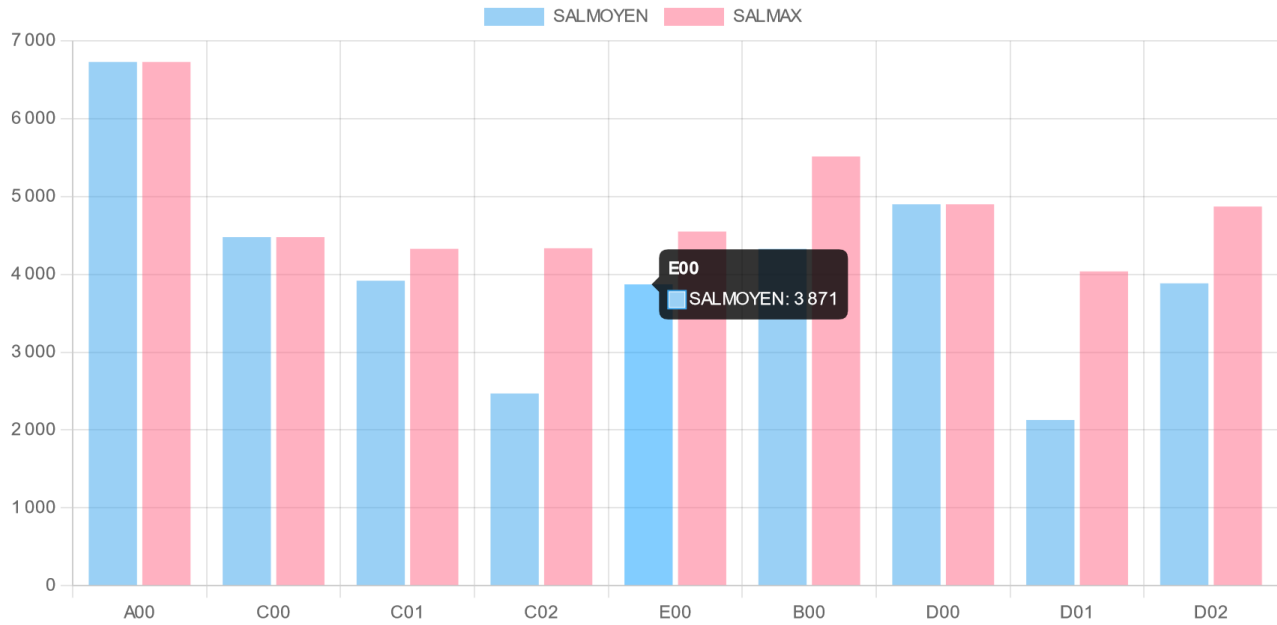


28. VS Code for i : notebooks

- Sortie HTML

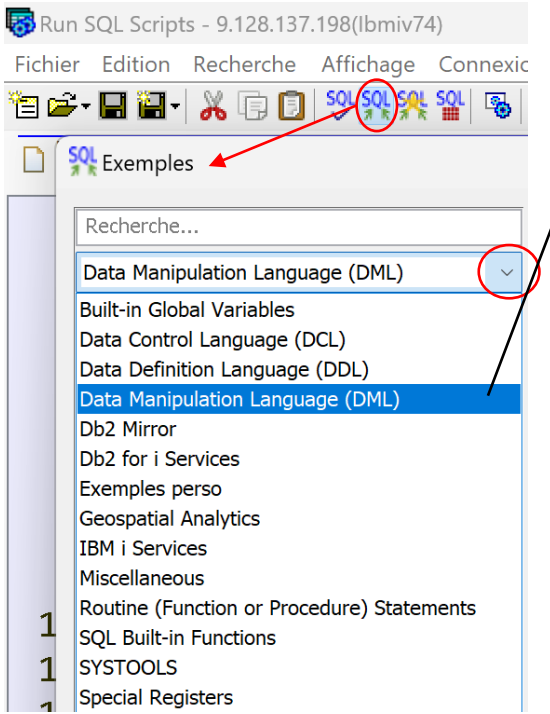
file:///C:/pbourgeois/Evenements/2024/UII/Présentations/Stats_salaires_par_service.html

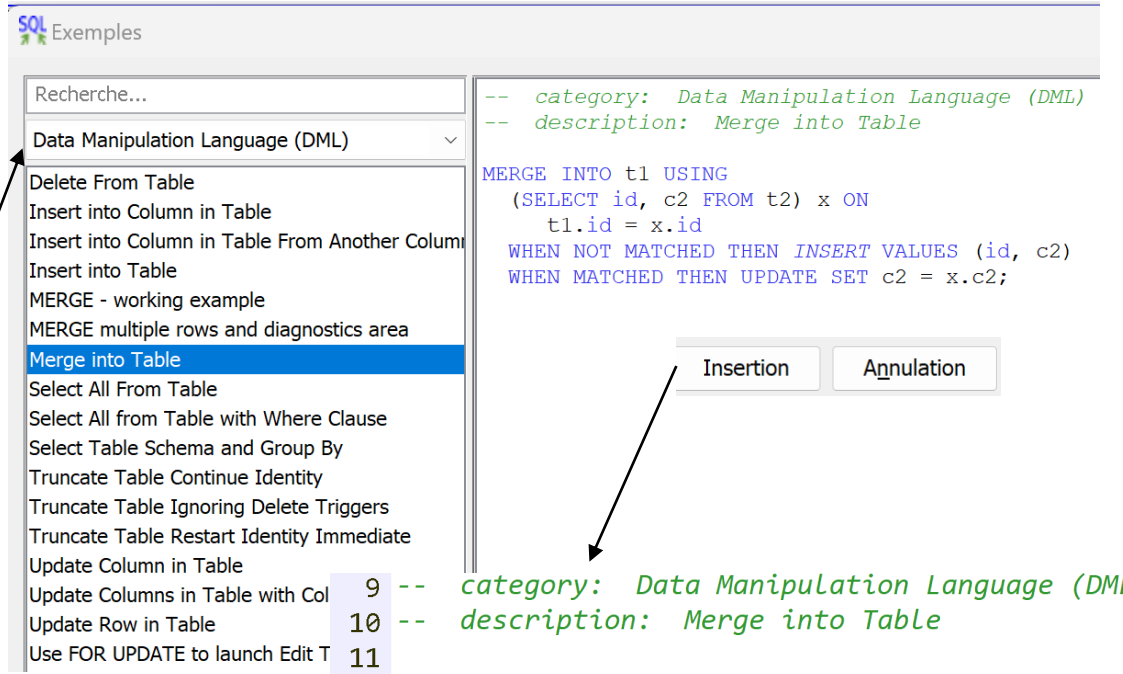
```
SELECT srv AS label, avg(salair) AS salmoyen, MAX(salair) as salmax
FROM as425f.employes GROUP BY srv
```



29. ACS et VS Code for i : exemples SQL

- Par **ACS**

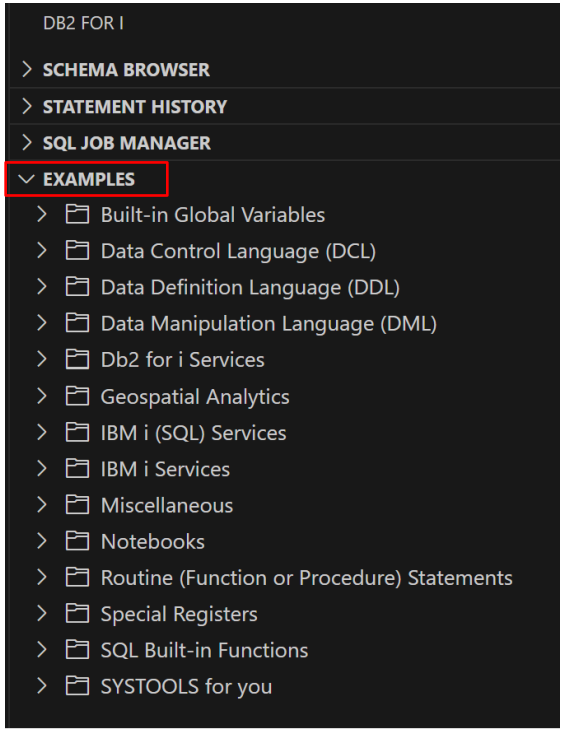


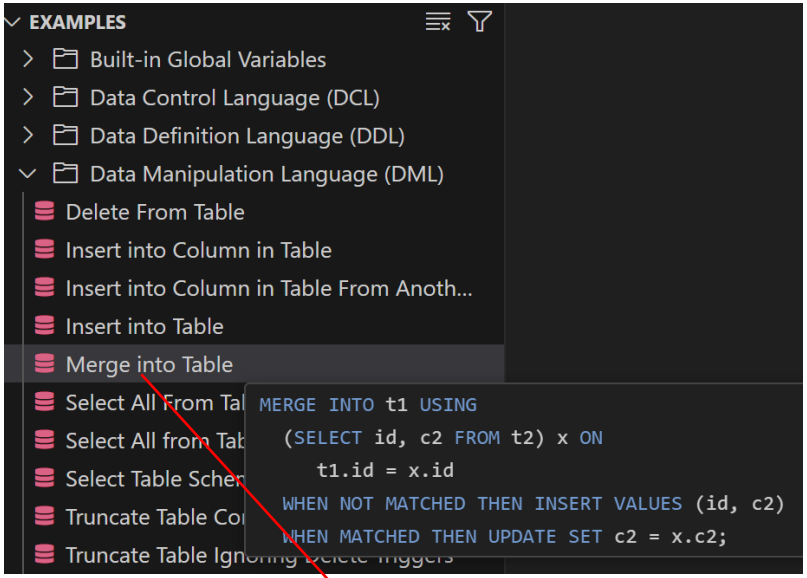


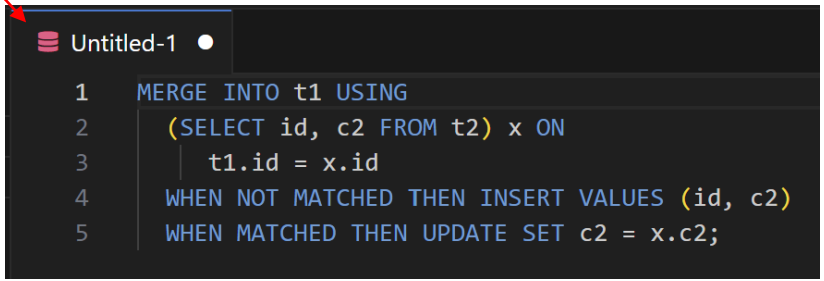

```
 9 --  category:  Data Manipulation Language (DML)
10 --  description:  Merge into Table
11
12 MERGE INTO t1 USING
13   (SELECT id, c2 FROM t2) x ON
14      t1.id = x.id
15   WHEN NOT MATCHED THEN INSERT VALUES (id, c2)
16   WHEN MATCHED THEN UPDATE SET c2 = x.c2;
```

29. ACS et VS Code for i : exemples SQL

- Par **VS Code for i**







30. SQL Tutor

- SQL Tutor : https://www.ibm.com/support/pages/ibm-i-tutorials-demos-and-sql-examples-0

IBM i Tutorials, Demos, and SQL examples

News

Abstract

The Db2 for i SQL Tutor landing page. This page aggregates resources to help you be successful with SQL.

Content

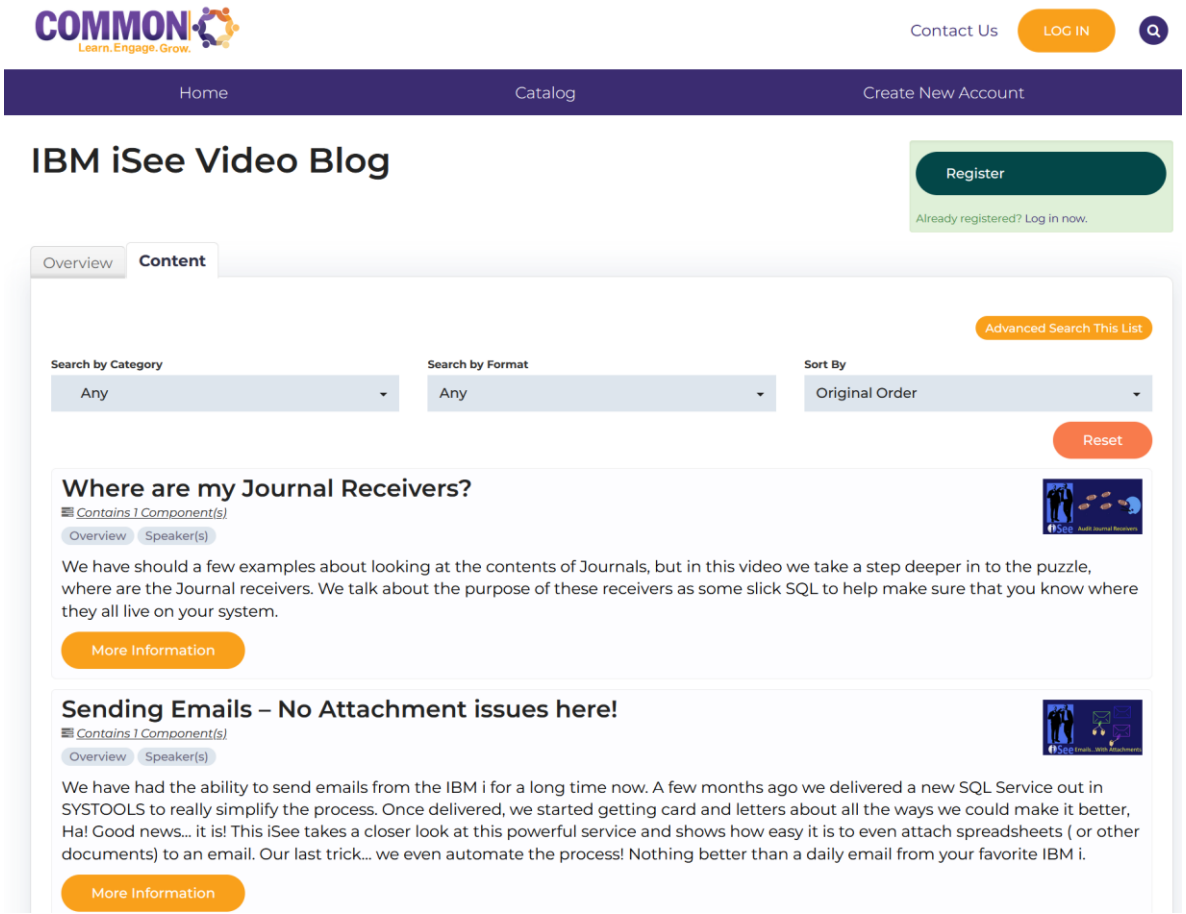
You are in: IBM i Tutorials, Demos, and SQL examples

List of all of the Db2 for i SQL Gists by Scott Forstie
List of all of the IBM i COMMON Tutorials by Scott Forstie & Tim Rowe

Categories
Access Client Solutions (ACS)
Database Topics
Db2 for i Services
IBM i Services
IBM i Topics
Integrated File System (IFS)
Limits
SQL Procedural Language (SQL PL)
SQL Examples
Security

30. SQL Tutor

- IBM iSee Video Blog
  - https://learn.common.org/products/ibm-isee-video-blog#tab-product_tab_content



DB2 et SQL : formez-vous !

- **Formation IBM + VOLUBIS**
  - SQLWS3 : SQL avancé sur IBM i
  - 3 jours
  - Théorie + travaux pratiques
  - Sur site à la carte
  - Dans les locaux d'IBM Bois-Colombes (sessions planifiées)
    - La prochaine session : **18, 19 et 20 mars 2025**

- **Workshops IBM**
  - Durée : 4 heures
  - A distance ou sur site
  - Possibilité d'utiliser vos vouchers IBM i (avant fin 2025)
  - Workshops :
    - DB2 – SQL DDL avancé
    - DB2 – SQL DML avancé
    - DB2 – SQL procédural
    - DB2 – Support XML/JSON
    - DB2 – Nouveautés V7
    - VS Code for i – Bases
    - VS Code for i – Compléments
    - RDi, RPG…

Contact : Philippe Bourgeois – pbourgeois@fr.ibm.com


MERCI