IBM Power Systems - IBM i

Modernisation, développement d'applications et DB2 sous IBM i
Technologies, outils et nouveautés 2013-2014

13 et 14 mai 2014 – IBM Client Center Paris, Bois-Colombes

S7 – DB2/SQL : introduction à une programmation centrée sur les données

Mardi 13 mai – 16h00-17h30

Philippe Bourgeois – IBM France

Plan de la présentation

- 1. Introduction à la modernisation base de données
- 2. Introduction à une programmation centrée sur les données
- 3. Déplacer les règles métier au niveau de la base de données
  - Génération automatique de clés
  - Contraintes (d'intégrité référentielle, de vérification)
  - Row and Column Access Control (RCAC)
- 4. Accéder à des ensembles de données en RPG/COBOL
  - SQL statique et dynamique
  - Vues
  - CTE (Common Table Expression)
- 5. Bonnes pratiques et erreurs à éviter

1. Introduction à la modernisation base de données

Les raisons de moderniser la base de données DB2 for i

- 1. Pouvoir utiliser des fonctionnalités qui ne sont pas disponibles avec les DDS et RLA (Record Level Access)
  - Noms longs, colonnes auto-incrémentées, attributs HIDDEN et AS ROW CHANGE TIMESTAMP, support des LOB et de XML, objets SEQUENCE, MQT, index EVI…
  - Traitement de masse, fonctions scalaires, fonctions de colonne, groupage, fonctions OLAP, sous-requêtes, requêtes récursives, etc.

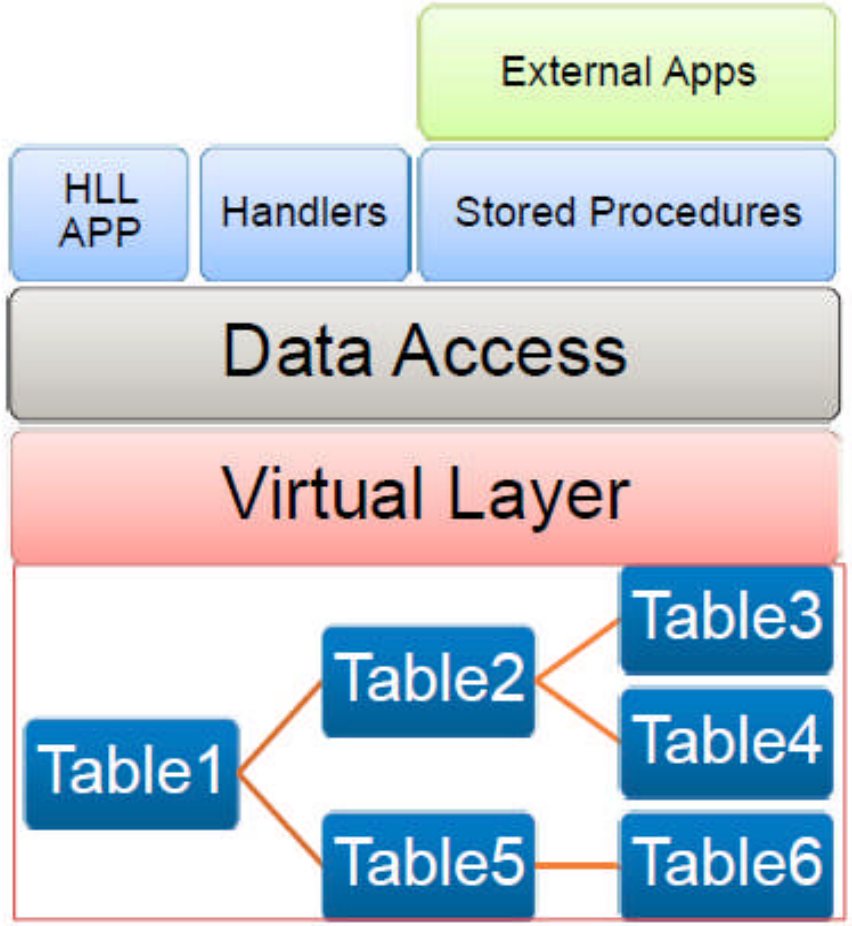
- 2. Etre « en phase » avec le monde des bases de données d'aujourd'hui
  - Disposer d'une base de données aux standards du marché
  - Pouvoir recruter des talents extérieurs ne connaissant pas forcément le i

Moderniser DB2 for i = SQL

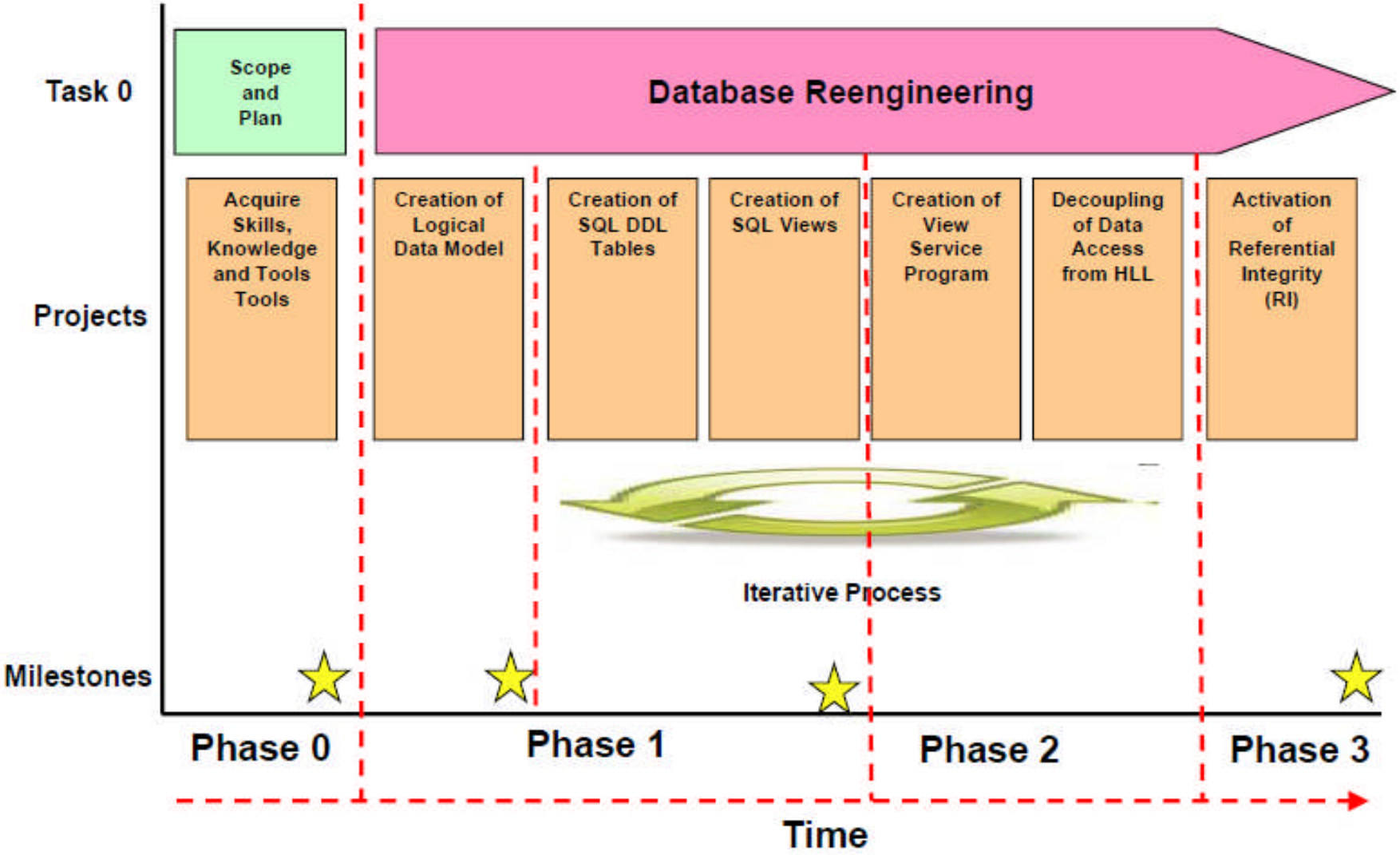
- Pourquoi SQL ?
  - L'interface stratégique sur toutes les plateformes
  - L'interface stratégique sur le i (la seule interface qui évolue)
  - L'assurance de trouver des compétences
  - Un gain en portabilité
  - L'assurance de bonnes performances
  - L'amélioration de l'intégrité
  - De puissantes fonctionnalités
  - Une amélioration de la productivité des développeurs
  - Un (encore) meilleur positionnement de l'IBM i comme serveur base de données

Re-engineering de la base de données – Framework suggéré

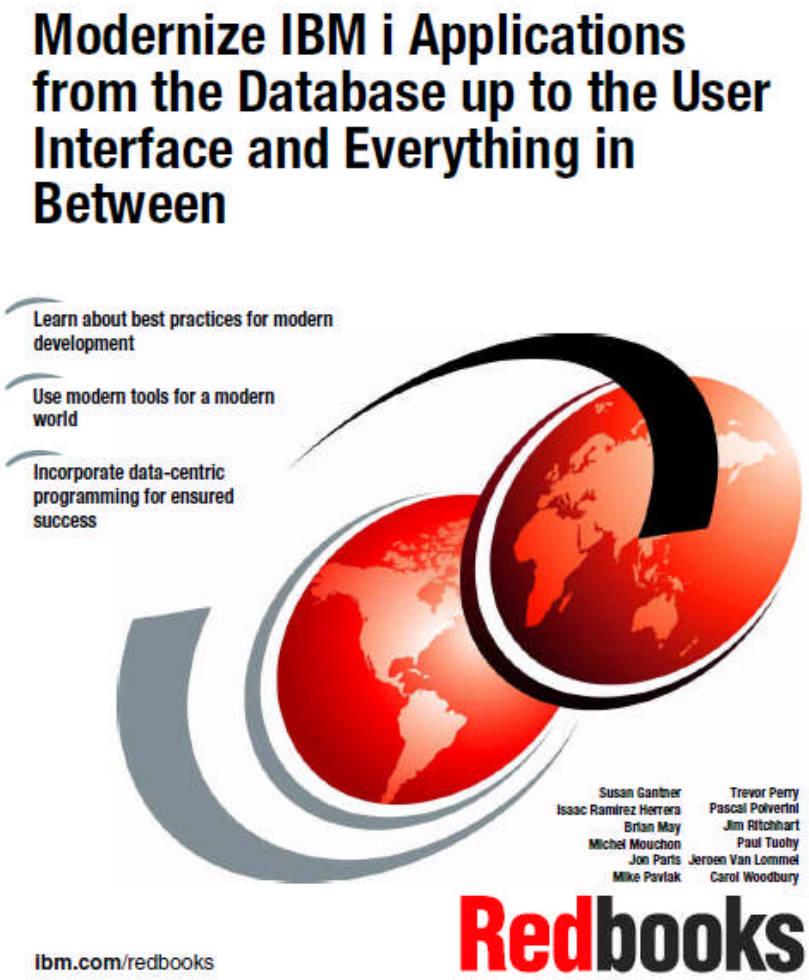
- Application Layer
  - Host programs
  - External access via Stored Procedures
- Data Access Layer
  - SQL DML
  - SQL Routines (procedures,functions,etc)
- Virtual or Data Abstract Layer
  - SQL Views
    - Instead of Triggers
  - Mask complexity
- Physical Data Model
  - DB2 for i
  - Highly normalized
  - Indexes, triggers, constraints



Re-engineering de la base de données – Etapes



Re-engineering de la base de données – Documentation



Modernisation de la base de données – Outils

- Outils IBM
  - IBM Navigator for i and System i Navigator
  - IBM Data Studio
    - Fourni en standard avec RDi
  - IBM InfoSphere Data Architect

- Outils partenaires
  - Xcase for i (Itheis / Resolution Software)
    - Découverte et implémentation des relations
    - Modélisation graphique
    - Conversion de DDS à SQL
  - Transformer DB (Arcad Software)
    - Vérification de l'intégrité des données
    - Renommage de zones BD
    - Conversion de DDS à SQL

14 mai **11h00** **12h30**	**Session 20 - Longchamp** Découverte des relations base de données DB2 et passage de DDS à SQL avec Xcase for i

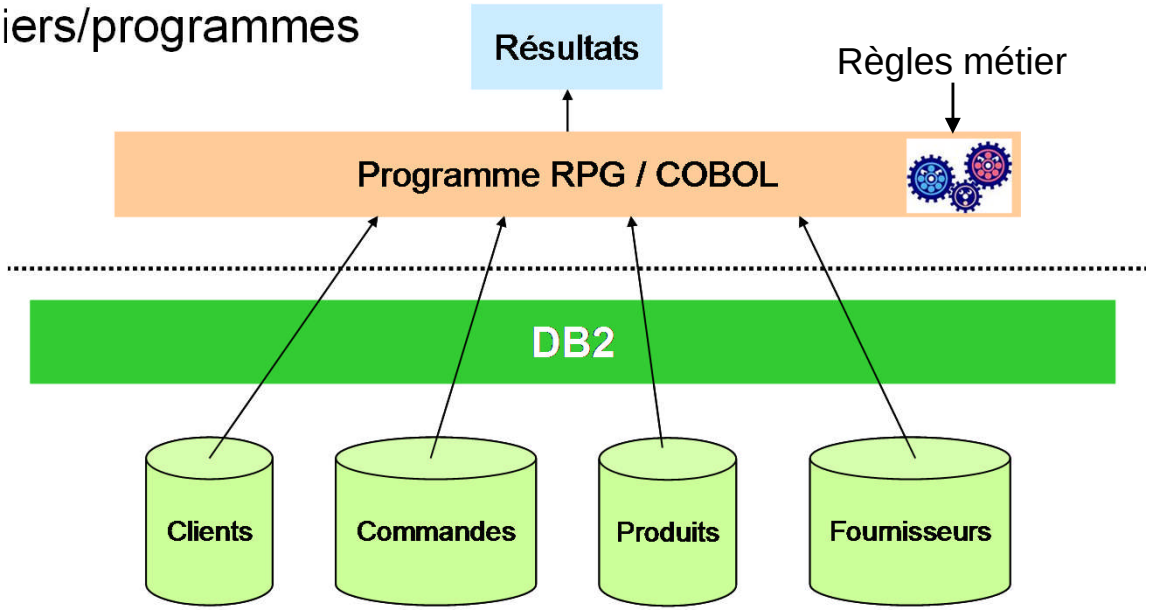
2. Introduction à une programmation centrée sur les données

Qu’est-ce qu’une programmation centrée sur les données ?

- C’est une programmation :
  - Qui permet de résoudre des problématiques métier en utilisant les possibilités de la base de données
  - Qui laisse le SGBD faire ce que programmiez auparavant
  - Qui implémente un maximum de logique métier au niveau de la base de données
  - Qui isole les règles métier des programmes

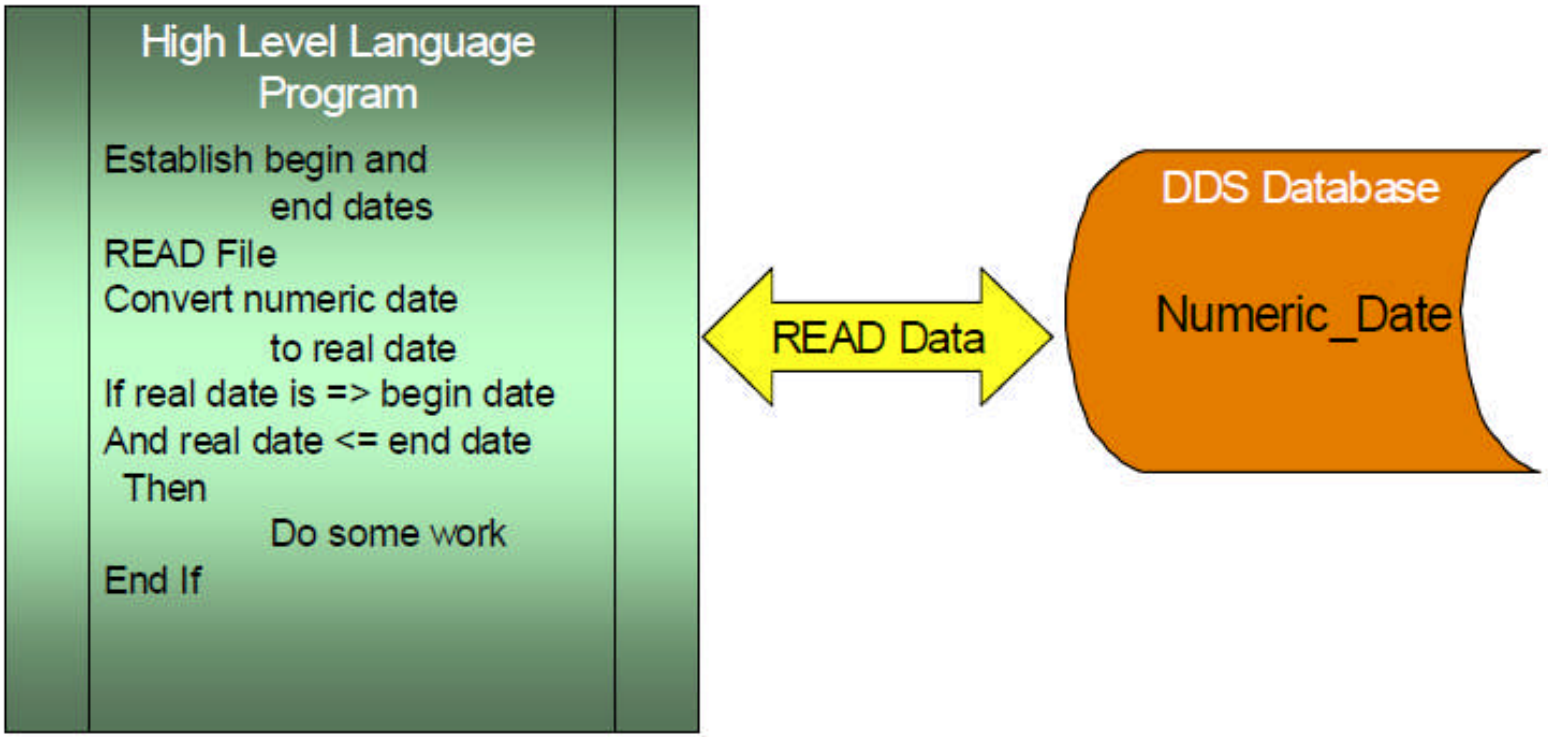
Programmation traditionnelle (centrée application)

- Le programme fait, connaît et contrôle tout
- Le programmeur décide des fichiers à traiter et détermine sur quelles clés le programme accèdera aux données
- Le programme accède aux données au niveau enregistrement (RLA – Record Level Access) et traite un enregistrement à la fois
- Les règles métier sont imbriquées dans les programmes
- Il y une dépendance forte fichiers/programmes
- Très souvent les relations entre les données se limitent à la connaissance du développeur et aux clés primaires définies sur des données métier (ex. numéro de client) et répétées autant de fois que nécessaire dans les PFs



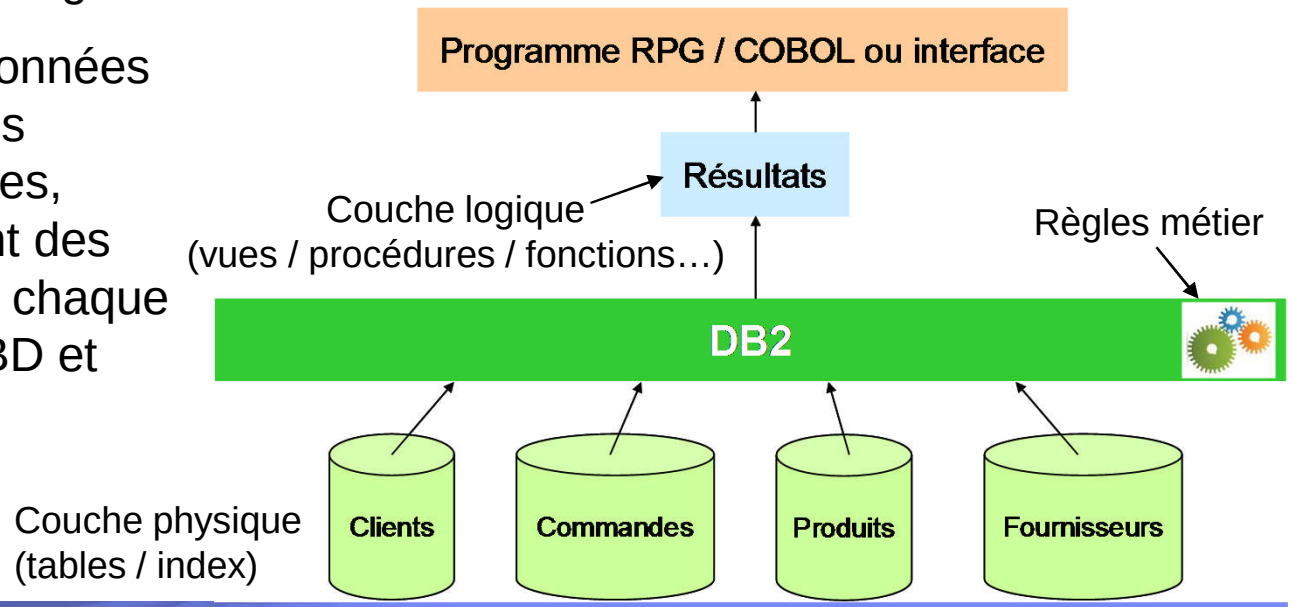
Programmation traditionnelle (centrée application)

- La logique est dans les programmes
- Lecture enregistrement par enregistrement
- N'utilise pas l'intelligence du SGBD



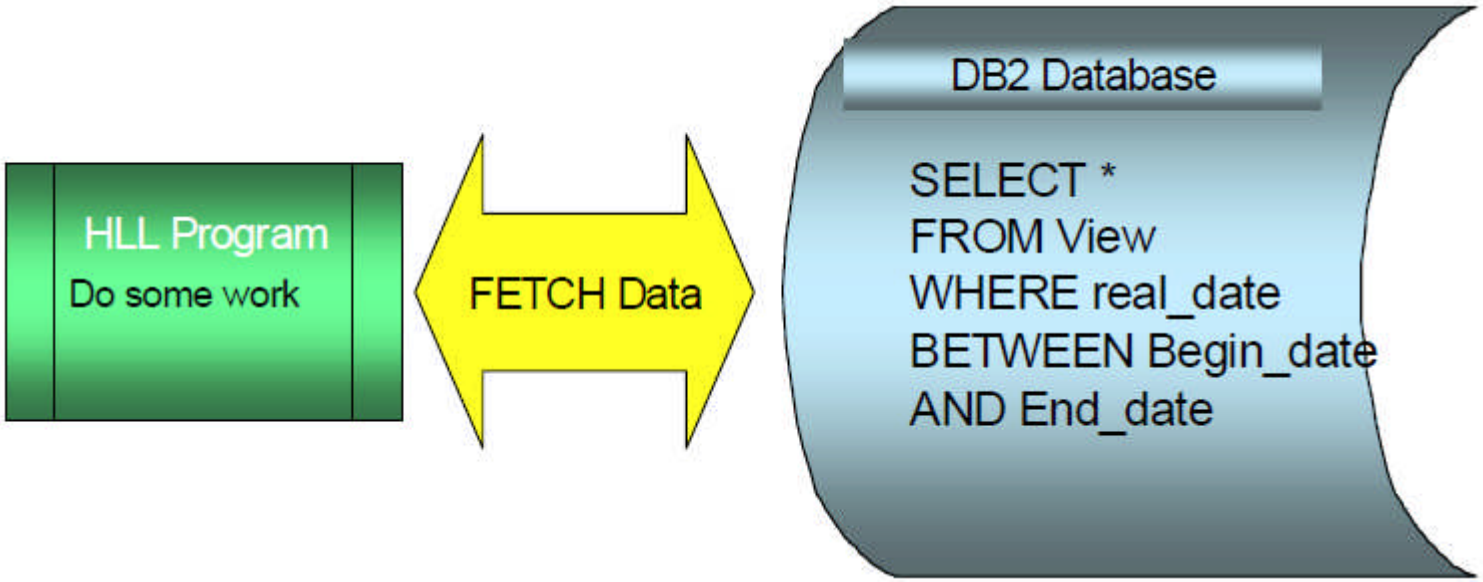
Programmation centrée sur les données

- Le programme interroge les données au travers de requêtes SQL qui renvoient un ensemble de données
- Le SGBD décide quelle est la meilleure méthode pour accéder aux données
  - Les chemins d'accès sont créés parce que le SGBD les conseille et non parce que le programmeur le décide
- Les règles métier communes sont créées comme contraintes au niveau de la base pour assurer une cohérence de l'intégrité
- La dépendance fichiers/programmes est faible
- Les relations entre les données sont définies par des clés primaires / clés étrangères, généralement en utilisant des colonnes d'identité dans chaque table, gérées par le SGBD et auto-incrémentées



Programmation centrée sur les données

- Moins de logique applicative dans les programmes
  - Moins de développement, moins de tests
  - L'exécution et le test de SQL peut se faire sans à avoir à programmer
- Traitement de multiples lignes en une seule opération
- Utilise l'intelligence de la base de données



Mise en œuvre d'une programmation centrée sur les données

- Deux axes de travail
  - 1. Déplacer les règles métier au niveau de la base de données
    - Contraintes de clé primaire et de clé unique
    - Génération automatique de valeurs
    - Contraintes d'intégrité référentielle
    - Contraintes de vérification
    - Row and Column Access Control (RCAC)
    - Déclencheurs (triggers)
  - 2. Accéder à des ensembles de données (data sets ou sets)
    - SQL statique et dynamique
    - Vues, unions, jointures
    - CTE (Common Table Expressions)
    - Sous-requêtes
    - Procédures, UDFs et UDTFs, triggers…

SQL et la programmation centrée sur les données

SQL est le bon langage pour une programmation centrée sur les données

SQL est le langage idéal pour traiter des **ensembles de données** (data sets)

- Il faut penser « ensemble de données »
  - SQL est un langage ensembliste : il faut penser en termes de définition d'ensemble de données et d'opérations sur ces ensembles de données
  - DDL (Data Definition Language)
    - Définition des ensembles de données
  - DML (Data Manipulation Language)
    - Opérations sur ces ensembles de données

3. Déplacer les règles métier au niveau de la base de données

Déplacer les règles métier au niveau de la base de données

- Normalisation
- Contraintes de clé primaire et de clé unique
- Contraintes d'intégrité référentielle
- Contraintes de vérification
- Génération automatique de valeurs
- Row and Column Access Control (RCAC)
- Contrôle de validation
- Déclencheurs (triggers)

Normalisation – Formes normales

- Objectifs
  - Eviter la redondance
  - Eviter les anomalies de relation

EMPLOYES

NOEMP	NOMEMP	INDICE	NOSRV
028	ROUX	55	D11
033	MIGNOT	52	D21
034	ETTORI	52	D21

SERVICES

NOSRV	NOMSRV
D11	Etudes
D21	Exploitation

EMPL_PROJ

NOEMP	NOPRJ	DUREE
028	P1	3
028	P2	6
033	P3	12
034	P2	5

PROJETS

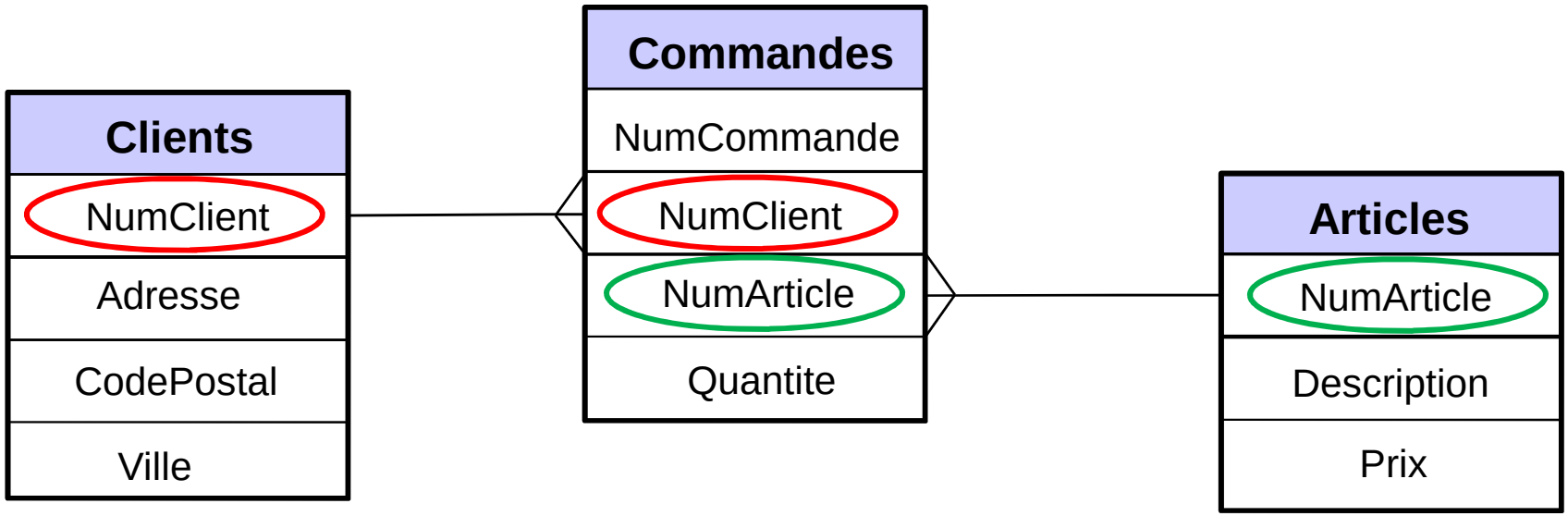
NOPRJ	NOMPRJ
P1	GPAO
P2	CAO
P3	CFAO

1ère forme normale
L'intersection d'une ligne et d'une colonne ne doit contenir qu'une seule valeur

2nde forme normale
Chaque zone non-clé doit être liée à la totalité de la clé

3ème forme normale
Une zone non-clé ne doit pas avoir de valeur unique possible vis-à-vis d'une autre zone non-clé

L'intégrité des données



- Que se passe-t-il si j'ajoute ou met à jour une commande avec un numéro de client ou d'article invalide ?

- Si l'on supprime un client, qu'advient-il des commandes de ce client ?

- Si l'on supprime un article, qu'advient-il des commandes concernant cet article ?

Intégrité des données – Définition de contraintes

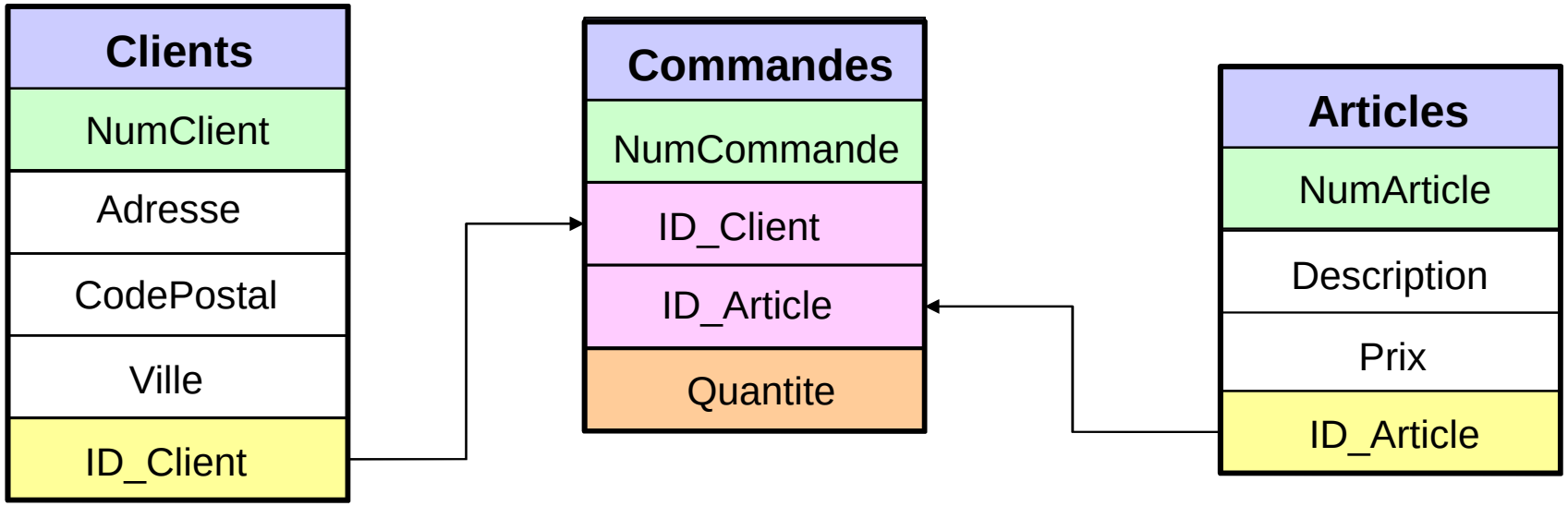
- De clé unique
  - Les valeurs de clé d'une table sont valides uniquement si elles sont uniques
  - Une ou plusieurs contraintes de clé unique peuvent être définies par table
- De clé primaire
  - Les valeurs de clé d'une table sont valides uniquement si elles sont uniques et non nulles
  - Une seule contrainte de clé primaire peut être définie par table
- De clé étrangère / d'intégrité référentielle
  - Les valeurs de la « clé étrangère » d'une table ne sont valides que si elles existent comme valeurs de « clé parente » d'une autre table ou bien sont nulles
    - Pour s'assurer que les données sont techniquement cohérentes entre elles
- De vérification
  - Limite les valeurs autorisées dans une ou plusieurs colonnes d'une table

Une contrainte est une règle contrôlée par le SGBD pour limiter les valeurs des données qui peuvent être insérées, modifiées ou supprimées dans une table

Intégrité des données – Définition de contraintes

- Les contraintes se définissent par SQL :
  - A la création de la table : CREATE TABLE
  - Après avoir créé la table : ALTER TABLE
- On peut les visualiser/gérer par
  - IBM Navigator for i
  - System i Navigator
- Elles peuvent également être définies en CL
  - ADDPFCST
  - CHGPFCST (permet également de les activer / désactiver)
  - RMVPFCST
  - WRKPFCST

Intégrité des données – Définition de contraintes



Contrainte	Règle
UNIQUE	Identifie la clé unique
PRIMARY KEY	Identifie la clé parente
FOREIGN KEY	Correspondance avec la clé parente (règle implicite) Définition des actions en cas d'ajout, mise à jour et suppression
CHECK	Assurance de valeurs correctes (exemple Quantité > 0)

Intégrité des données – Définition de contraintes

- Clé unique et clé primaire (unique key – primary key)
  -
```
create table articles (
   id_article integer generated always as identity,
   numarticle integer,
   description char(50),
   prix decimal(9, 2),
   constraint articles_pk primary key (id_article),
   constraint articles_uk unique (numarticle))
```

- Vérification (check)
  -
```
create table commandes (
   numcommande ... / ...
   constraint quantite_positive check (quantite > 0))
```

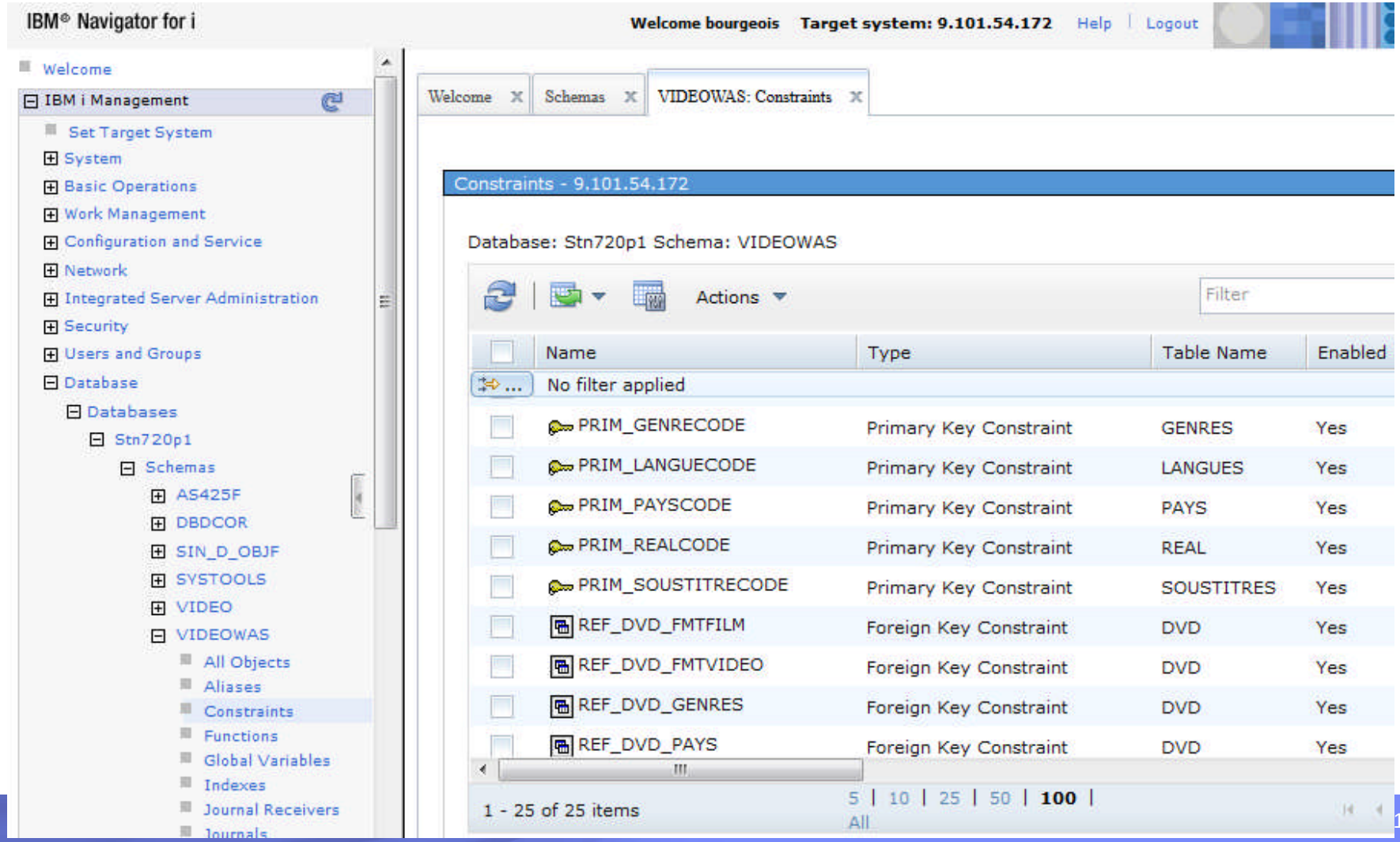
Intégrité des données – Définition de contraintes

- Clé étrangère (foreign key) – Intégrité référentielle
  -
```
create table commandes (
    numcommande .../ ...
    constraint cmd_fk_cli foreign key (id_client) references
clients (id_client) on delete cascade,
    constraint cmd_fk_art foreign key (id_article) references
article (id_article) on delete cascade)
```

  - Règles en cas de suppression dans la table parente (ON DELETE)
    - RESTRICT : pas de suppression si clé étrangère associée
    - NOACTION : idem mais déclencheur AFTER possible
    - CASCADE : suppression dans les tables parente et dépendante
    - SETNULL : suppression dans la table parente – Valeur indéfinie dans la table dépendante
    - SETDFT : suppression dans la table parente – Valeur par défaut dans la table dépendante
  - Règles en cas de mise à jour dans la table parente (ON UPDATE)
    - RESTRICT : pas de mise à jour si clé étrangère associée
    - NOACTION : idem mais déclencheur AFTER possible

Intégrité des données – Définition de contraintes

- Visualisation / gestion des contraintes par IBM i Navigator



Intégrité des données – Intégrité fonctionnelle - Triggers

- Pour assurer l'intégrité des données non pas techniquement mais fonctionnellement, on ne pas définir une contrainte d'intégrité référentielle. On peut dans ce cas passer par un trigger (exemple : valider l'état du crédit d'un client avant d'insérer une commande)

- Déclencheur (trigger)
  - Un déclencheur est un programme, appelé avant et/ou après ajout, mise-à-jour, suppression ou après lecture d'enregistrements
  - Déclenchement
    - Pour chaque ligne ou pour l'ensemble des lignes concernées
    - Avant : validation, autorisation, complémentation
    - Après : historisation, log, action dépendante

```
CREATE TRIGGER asurveiller
  AFTER INSERT ON ndfs
  REFERENCING NEW AS n
  FOR EACH ROW
  MODE DB2ROW
    WHEN (n.totalndf > 5000)
    BEGIN
      DECLARE nomemp CHAR(30);
      SET nomemp = (SELECT nom FROM employes
                    WHERE mat = n.mat);
      INSERT INTO auditndf
      VALUES(n.mat, nomemp, n.numndf, n.totalndf);
    END
```

Génération automatique de valeurs – Attribut IDENTITY

- Pour l'incrémentation automatique d'une colonne numérique
- La valeur de départ et l'incrément peuvent être spécifiés
  - La valeur par défaut est 1 pour les deux
- Cette colonne étant en général destinée à identifier chaque ligne, elle est candidate à être clé primaire
- Exemple :
  - `create table articles (id_article integer generated always as identity, ... / ...`
- Il existe également les objets de type SEQUENCE, qui permettent de disposer d'une valeur auto-incrémentée extérieure à une table particulière et partageable entre tables
  - `create sequence s1 start with 1 no max value`
  - `insert into commandes (…) values(next value for s1, …)`

RCAC – Row and Column Access Control

- Nouveauté IBM i 7.2
  - Couche additionnelle de sécurité, complémentaire à la sécurité niveau table
  - Permet de limiter l'accès à certaines données (certaines lignes et/ou certaines colonnes d'une table)
  - Deux types de règle :
    - PERMISSION pour restreindre l'accès aux lignes
    - MASK pour restreindre l'accès aux colonnes
  - Se définit au niveau de la base de données (approche « Data Centric »)
    - S'applique quelque soit l'interface d'accès à la table
    - Ne nécessite pas la modification des applications
    - Personne n'y échappe
  - Nécessite l'option 47 de SS1 (IBM Advanced Data Security for i)

```
5770SS1    47      IBM Advanced Data Security for i
```

Gratuit

RCAC – Row and Column Access Control

- RCAC – Restreindre l'accès à certaines colonnes (CREATE MASK)

```
CREATE MASK MASQUE_NUMSECU ON EMPLOYES FOR COLUMN NUMSECU RETURN
  CASE
   WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER,'PAYE') = 1)THEN
NUMSECU
   WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER,'MGR') = 1)
        THEN '*******' CONCAT SUBSTR(NUMSECU,8,6)
   ELSE NULL
  END
  ENABLE;
ALTER TABLE EMPLOYES ACTIVATE COLUMN ACCESS CONTROL;
```

- La fonction VERIFY_GROUP_FOR_USER permet de vérifier qu'un utilisateur est bien associé à un profil de groupe

RCAC – Row and Column Access Control

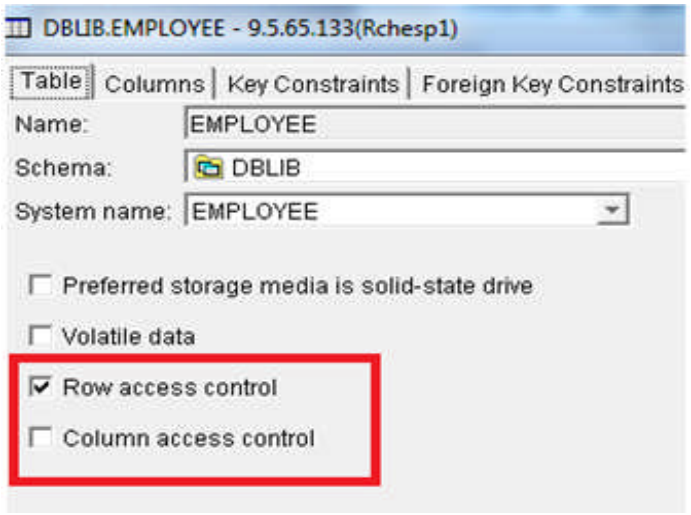
- RCAC – Restreindre l'accès à certaines lignes (CREATE PERMISSION)

```
CREATE PERMISSION PERMISSION1_EMPLOYES ON EMPLOYES FOR ROWS
WHERE (VERIFY_GROUP_FOR_USER(SESSION_USER, 'RH') = 1)
ENFORCED FOR ALL ACCESS ENABLE;

ALTER TABLE EMPLOYES ACTIVATE ROW ACCESS CONTROL;
```

```
CREATE PERMISSION PERMISSION1_SALAIRES ON SALAIRES FOR ROWS
WHERE CURRENT TIME BETWEEN '8:00' AND '17:00'
AND SYSIBM.ROUTINE_SPECIFIC_NAME = 'PROC1'
AND SYSIBM.ROUTINE_SCHEMA = 'HRPROCS'
AND SYSIBM.ROUTINE_TYPE = 'P'FOR ALL ACCESS ENABLE;

ALTER TABLE SALAIRES ACTIVATE ROW ACCESS CONTROL;
```

RCAC – Row and Column Access Control

- RCAC – Compléments
  - Seuls les utilisateurs avec la fonction « Administration de la sécurité base de données » (QIBM_DB_SECADM) pourront gérer la sécurité RCAC (WRKFCNUSG)
    - Ces utilisateurs n'auront pas forcément accès aux données des tables…
  - Nouveaux catalogues, nouveaux codes journal et audit, intégration dans System i Navigator et IBM Navigator for i



Contrôle de validation

- Le contrôle de validation assure l'accomplissement d'une transaction selon le principe du "tout ou rien"
  - Transaction : ensemble d'opérations pouvant être considéré comme une seule opération
- Utilisation des ordres COMMIT et ROLLBACK dans les programmes
  - Fichiers CT_COURANT et CT_EPARGNE ouverts sous contrôle de validation
    - Lecture CT_COURANT
    - Débit CT_COURANT
    - Lecture CT_EPARGNE
    - Crédit CT_EPARGNE
    - COMMIT
- Une transaction incomplète est annulée dans les cas suivants :
  - Fin anormale du travail
  - Arrêt du système
  - Ordre ROLLBACK du programme

4. Accéder à des ensembles de données (data sets ou sets)

Définition et accès à des ensembles de données (sets)

- SQL statique et dynamique
- Vues, unions, jointures
- CTE (Common Table Expressions)
- Sous-requêtes
- Procédures
- UDTFs

Définir les « sets » avec l'instruction SELECT

SELECT	Liste des colonnes retournées
FROM	Source ou résultat d'une requête
WHERE	Critères de sélection des lignes
GROUP BY	Critères de groupage / agrégation
HAVING	Critères de sélection des lignes sur les groupes / agrégats
ORDER BY	Critères de tri du result set final
FETCH FIRST	Limiter le nombre de lignes retournées

Rappels sur le SQL embarqué dans les programmes

- Pourquoi ?
  - Simplification des développements
    - Bénéficier de toute la puissance du DML (Data Manipulation Language)
      - Fonctions scalaires, fonctions de colonne, groupage, sous-requêtes…
    - Gestion plus souple des sélections
      - Sélections multicritères ou conditionnées de façon dynamique
    - Plus globalement, toutes les opérations ensemblistes que l'on peut réaliser en dehors du mode « ligne à ligne »
  - Permet de réduire le lien programme <-> fichier

- Comment ?
  - Membre source de type SQLxxx (SQLRPGLE, SQLCBLLE…)
  - Compilation par CRTSQLxxx (CRTSQLRPGI, CRTSQLCBLI…)

Rappels sur le SQL embarqué dans les programmes

- Comment ?
  - Dans les sources RPGLE :
    - En format libre
      - `exec sql instruction SQL;`
    - En format fixe

```
c/exec sql
c+ instruction SQL
c/end-exec
```

  - Dans les sources CBLLE :

```
exec sql
 instruction SQL
end-exec
```

- Variables hôte
  - Variable (simple, DS ou tableau de DS) du programme RPG/COBOL utilisable dans les instructions SQL statiques
  - Le nom de la variable est précédé par « : »

Rappels sur le SQL embarqué dans les programmes

- SQL statique
  - L'instruction SQL est interprétable à la compilation (pas de zones ou de critères définis dynamiquement)
  - 1. SQL statique **sans** curseur – Exemples de SELECT INTO

```
dcl-s wnumemp char(6);
dcl-s wnomemp char(15);
dcl-s wnumsrv char(3);

wnumemp = '00001';

exec SQL  select nomemp, numsrv into :wnomemp, :wnumsrv
           from  employes where numemp = :wnumemp;
```

Le SELECT ne renvoie qu'un seul enregistrement

Avec une DS

```
dcl-s wnumemp int(10);
dcl-ds dsemp extname(employes) end-ds;

wnumemp = 40;

exec SQL  select * into :dsemp from  employes  where numemp = :wnumemp;
```

SQL statique sans curseur

- Exemples pour traiter un **ensemble** de données

```
dcl-s wprime packed(9:2);
dcl-s wnumsrv char(3);

wnumsrv = 'D21';
wprime = 100;

exec sql UPDATE employes
      SET salaire = salaire + (salaire * .1) ,
          prime  = :wprime
      WHERE numsrv = :wnumsrv ;
```

INSERT, UPDATE et DELETE de masse

```
Fdemod01   cf   e             workstn
C     *entry        plist
C                   parm                    nsrv
C
 /free
   exec sql SELECT MIN(salaire), DECIMAL(AVG(salaire), 7, 2), MAX(salaire)
            INTO :salmin, :salmoy, :salmax FROM tabempl WHERE srv = :nsrv;

  if sqlcode = 0;
    exfmt f1;
  endif;
  *inlr = *on;
 /end-free
```

Utilisation de la puissance du langage SQL (fonctions scalaires, de colonne, groupage, etc.)

```
   ANALYSE D'UN SERVICE

Service . . . . : 911
Salaire minimum : 2.140,00
Salaire moyen   : 2.260,00
Salaire maximum : 2.360,00
```

Rappels sur le SQL embarqué dans les programmes

- 2. SQL statique **avec** curseur
  - Les curseurs sont utiles pour lire plusieurs enregistrements ligne à ligne
  - Cinématique
    - Déclarer le curseur
      - **DECLARE c1** <SCROLL> **CURSOR FOR SELECT …** <FOR READ ONLY / FOR UPDATE OF>
    - Ouvrir le curseur
      - **OPEN c1**
      - Exécute la requête SELECT et crée un RESULT SET (liste d'enregistrements)
    - Lire les enregistrements par une boucle
      - **FETCH** <NEXT / PRIOR / FIRST / LAST / BEFORE / AFTER / CURRENT / RELATIVE > **FROM c1** <FOR n ROWS> **INTO…**
    - Fermer le curseur
      - **CLOSE c1**

SQL statique avec curseur

- Exemples pour traiter un **ensemble** de données

```
Fdemod03   cf   e            workstn
 /free
   exec sql DECLARE employes CURSOR FOR SELECT srv,
            DECIMAL(SUM(salaire), 9, 2) AS total
            FROM employes GROUP BY srv;

   exec sql OPEN employes;

   dow sqlcod <> 100;
    exec sql FETCH FROM employes INTO :srv, :total;
    exfmt f1;
   enddo;
   exec sql CLOSE employes;
   *inlr = *on;
 /end-free
```

Utilisation de la puissance du langage SQL (fonctions scalaires, de colonne, groupage, etc.)

Ajout groupé d'enregistrements

```
dcl-ds dsemp extname(employes) qualified dim(10) end-ds;
dcl-s  nbemp int(10);

nbemp = 8;
exec sql INSERT INTO employes :nbemp ROWS VALUES(:dsemp)
```

```
dcl-ds dsemp extname(employes) qualified dim(10) end-ds;

exec sql DECLARE c1 CURSOR FOR
         SELECT * FROM employees WHERE srv = 'D21';

exec sql OPEN c1 ;

exec sql FETCH c1 FOR 10 ROWS INTO :dsemp;
// SQLErrd(3) contient le nombre de lignes effectivement retournées

exec sql CLOSE C1 ;
```

Lecture groupée d'enregistrements

Rappels sur le SQL embarqué dans les programmes

- SQL dynamique
  - L'instruction est construite dynamiquement
  - 3 types :
    - Pour un SELECT
      - Alimentation d'une variable avec la requête : **req1 = 'SELECT…'**
      - Préparation de l'ordre SQL : **PREPARE ordre1 FROM :req1**
      - Déclaration du curseur : **DECLARE c1 CURSOR FOR ordre1**
      - Puis **OPEN USING valeurs des paramètres**, FETCH, CLOSE
    - Pour autre chose qu'un SELECT
      - Si l'instruction comporte des paramètres :
        - Alimentation d'une variable avec la requête : **req1 = 'UPDATE…'**
        - Préparation de l'ordre SQL : **PREPARE ordre1 FROM :req1**
        - Exécution : **EXECUTE ordre1 USING valeurs des paramètres**
      - Si l'instruction ne comporte pas de paramètres :
        - Alimentation d'une variable avec la requête : **req1 = 'UPDATE…'**
        - Exécution : **EXECUTE IMMEDIATE req1**

SQL dynamique – Réutilisation des instructions préparées

- SQL dynamique
  - L'instruction est construite dynamiquement
  - Pas de possibilité d'utiliser des variables hôte
  - Pour définir les paramètres, on utilise le marqueur « ? »
    - Les valeurs spécifiées dans la clause USING (instructions EXCUTE et OPEN) sont insérées dynamiquement lors de l'exécution de l'instruction
  - Une instruction qui a été préparée (instruction PREPARE) peut être exécutée plusieurs fois avec des valeurs de paramètres différentes

```
dcl-s req1 char(100) varying inz('UPDATE employes SET sal = +
                                  sal*(100+?)/100 WHERE sx = ?');
dcl-s sx char(1);
dcl-s pctf int(5);
dcl-s pcth int(5);

exec sql PREPARE ordre1 FROM :req1;

sx = 'F';
pctf = 6;
exec sql EXECUTE ordre1 USING :pctf, :sx;

sx = 'M';
pcth = 4;
exec sql EXECUTE ordre1 USING :pcth, :sx;
```

Utiliser les fonctionnalités de SQL pour réduire le nombre d'instructions

Multiple SQL Statements

```
DECLARE CURSOR cursor1 FOR
SELECT col1, col2, ... col9
FROM t1
WHERE cust_id = 1234
  AND transaction_date = '2012.04.01';


OPEN cursor1;

DO
  READ cursor1 INTO :c1, :c2, ..., :c9;

  INSERT INTO t2 VALUES( :c1, :c2, ..., :c9
);
UNTIL ( no more data );

CLOSE cursor1;
```

SINGLE SQL Request

```
INSERT INTO t2
  SELECT col1, col2, ... col9
  FROM t1
  WHERE cust_id = 1234
    AND transaction_date =
                  '2012.04.01';
```

Utiliser les fonctionnalités de SQL pour réduire le nombre d'instructions

Multiple SQL Statements

```
DECLARE CURSOR cursor1 FOR
SELECT custid FROM order_table
WHERE ord_date  = '2012/11/03';

OPEN cursor1;
DO
 FETCH cursor1 INTO :v_custid;

 SELECT cust_name, cust_address
  INTO :v_name, :v_address
  FROM cust_table
        WHERE custid= :v_custid;

 /* Process customer data */
UNTIL ( no more data );

CLOSE cursor1;
```

SIMPLIFIED SQL Request

```
DECLARE CURSOR cursor1 FOR
SELECT c.cust_name, c.cust_address
  FROM order_table o, cust_table c
WHERE ord_date  = '2012/11/03'
        AND
           o.custid = c.custid;

OPEN cursor1;

DO
 FETCH cursor1 INTO
        :v_name, v_address;
 /* Process customer data */
UNTIL ( no more data );

CLOSE cursor1;
```

Mise-à-jour d'ensembles de données

- Modification d'une valeur dans plusieurs lignes d'une table

```
Exec sql
   Update table-a
   set column-a = somevalue;
```

- Modification d'une valeur dans une table si une ligne existe dans une autre table

```
Exec sql
   update table-a
   Set column-a = somevalue
   Where exists (select * from table-b
      Where table-a-keyfield = table-b-keyfield);
```

- Modification d'une valeur dans une table avec la valeur provenant d'une ligne d'une autre table

```
Exec sql
   Update table-a
   Set column-a = (select column-a from table-b
                   where table-a-keyfield = table-b-keyfield)
Where exists (select * from table-b
      Where table-a-keyfield = table-b-keyfield);
```

Les vues SQL

- Les vues SQL sont utilisées pour définir une table virtuelle
  - **`CREATE VIEW schema1.vue1 AS SELECT * FROM schema1.table1 WHERE colonne1 = 'xxx'`**
- Les vues SQL ne contiennent pas de données
  - Elles définissent un ensemble logique de données
  - Elles ne peuvent pas être ordonnées
  - Elles ne peuvent pas être indexées
- Dans les instructions SQL, les vues sont référencées comme les tables
  - **`SELECT * FROM schema1.vue1 ORDER BY colonne 2`**
- Les vues peuvent être utilisées pour définir des structures de données externes
- Les vues peuvent être ouvertes par des programmes RPG/COBOL comme des fichiers logiques sans clé

Les vues SQL

- Les vues apportent plus de flexibilité en termes de sélection et de traitement des données :
  - Fonctions de colonne (SUM, AVG, COUNT, MIN, MAX…)
  - Fonctions scalaires (alphanumériques, de temps..)
  - Groupage (GROUP BY)
  - Fonctions OLAP (CUBE, ROLLUP…)
  - Tous types de jointure et d'unions
  - Sous-requêtes…
- Les vues permettent
  - La réutilisation de requêtes communes ou complexes
  - De diviser une requête complexe en plusieurs étapes
  - De faire des opérations de contrôle ou de cosmétique
  - D'isoler les applications des changements dans les tables
- Les vues peuvent être jointes avec d'autres vues
- Les vues peuvent être directement adressées en ODBC/JDBC

Exemples de vues – Flexibilité - Réutilisation

- **CREATE VIEW** stat_sal

```
AS SELECT srv,
SUM(sal) AS sal_total,
MIN(sal) AS sal_min,
MAX(sal) AS sal_max,
AVG(sal) AS sal_moyen

FROM employes GROUP BY
srv
```

MAT	NOM	SX	SRV	SAL	DAT_NAI
20	MICHEL	M	911	2140	1955-03-17
10	ANNIE	F	911	2280	1949-05-12
50	JACQUES	M	911	2360	1956-11-11
40	DANIELE	F	977	2210	1962-07-13
30	MARC	M	977	2010	1972-02-28
60	CLAUDE	M	990	2210	1968-05-05
70	RICHARD	M	-	-	1953-01-01

Table EMPLOYES

- **SELECT * FROM** stat_sal **ORDER BY** srv

SRV	SAL_TOTAL	SAL_MIN	SAL_MAX	SAL_MOYEN
911	6.780,00	2.140,00	2.360,00	2,260,00…
977	4.220,00	2.010,00	2.210,00	2.110,00…
990	2.210,00	2.210,00	2.210,00	2.210,00…
-		-	-	-

Vue STAT_SAL

Autres exemples d'utilisation des vues

Complex Query

```
SELECT ... FROM T1, T2, T3,T4...
GROUP BY... ORDER BY...
```

Masking Complexity

```
CREATE VIEW COMPLEX_QUERY
AS SELECT ... FROM T1, T2, T3,T4... GROUP BY...;

SELECT * FROM COMPLEX_QUERY ORDER BY...;
```

Program Described Structure

```
T1_RESULT DS
  A  CHAR(10)
  B  DECIMAL(11,0)
  C  DATE
SELECT A, B, C INTO  :T1_RESULT FROM T1
```

Externally Described Structure

```
CREATE VIEW  T1_VIEW
AS SELECT A, B, C FROM T1;

T1_RESULT DS EXTNAME(T1_VIEW)

SELECT *  INTO  :T1_RESULT FROM T1_VIEW
```

Multiple Views and Data Structures

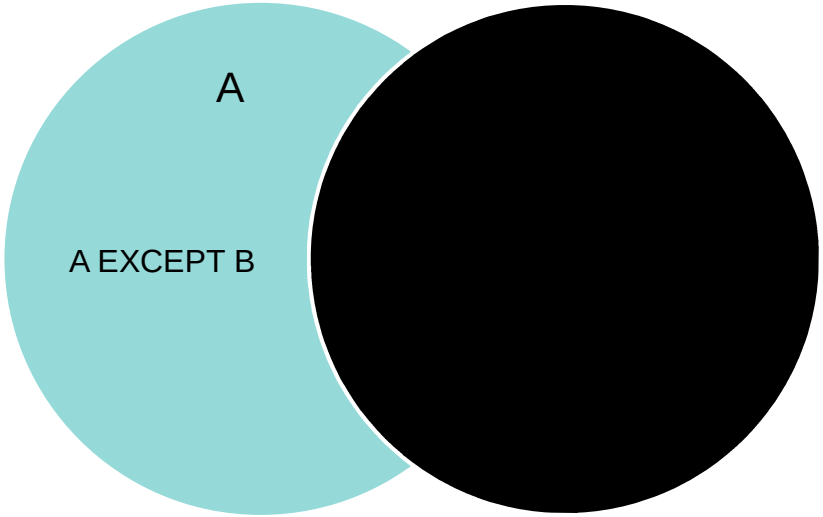
```
T1_RESULT DS EXTNAME T1_VIEW
T2_RESULT DS EXTNAME T2_VIEW

SELECT * INTO  :T1_RESULT FROM T1_VIEW
SELECT * INTO  :T2_RESULT FROM T2_VIEW
WHERE T2.A = :T1_RESULT.A
```

Joined View One Structure

```
CREATE VIEW  JOIN_VIEW AS
SELECT A, B, C, D, E, F FROM T1 JOIN T2 USING(A);

JOIN_RESULT DS EXTNAME(JOIN_VIEW)

SELECT * FROM JOIN_VIEW INTO  :JOIN_RESULT
```

Unions et jointures – Assemblage de lignes/colonnes

- Union – Assemblage de lignes de différentes tables
  - Syntaxe : select… **UNION / UNION ALL / EXCEPT / INTERSECT** select…
  - Opérateurs
    - UNION : toutes les lignes des 2 ensembles sans les doublons
    - UNION ALL : toutes les lignes des 2 ensembles avec les doublons
    - EXCEPT : les lignes du 1er ensemble non présentes dans le 2nd
    - INTERSECT : les lignes communes aux 2 ensembles



Unions et jointures – Assemblage de lignes/colonnes

- Jointures – Assemblage de colonnes de différentes tables
  - Syntaxe : select… from table1 **INNER / LEFT-RIGHT-FULL OUTER / LEFT-RIGHT EXCEPTION JOIN** table2 **ON / USING zone(s) de jointures** …
  - INNER JOIN
    - Uniquement les lignes de table1 qui ont une correspondance dans table2
  - LEFT OUTER JOIN
    - Toutes les lignes de table1 même si elles n'ont pas de correspondance dans table2
  - RIGHT OUTER JOIN
    - Toutes les lignes de table2 même si elles n'ont pas de correspondance dans table1
  - FULL OUTER JOIN
    - Toutes les lignes de table1 et de table2
  - LEFT EXCEPTION JOIN
    - Les lignes de table1 qui n'ont pas de correspondance dans table2
  - RIGHT EXCEPTION JOIN
    - Les lignes de table2 qui n'ont pas de correspondence dans table1

Exemples de jointures

INNER JOIN 2 or more base tables

```
SELECT a.col1, b.col2, c.col3
FROM T1 a JOIN T2 b USING (ID)
  JOIN T3 c ON a.ID = c.ID
```

INNER JOIN notes

USING shorthand if join column names are same

ON if names are different

FROM T1, T2 WHERE T1.ID = T2.ID (also works)

No difference in implementation of the above

OUTER JOIN examples

```
SELECT a.col1, b.col2, c.col3
FROM T1 a LEFT JOIN T2 b USING (ID)
  JOIN T3 c ON a.ID = c.ID

SELECT a.col1, b.col2, c.col3
FROM T1 a RIGHT JOIN T2 b USING (ID)
```

OUTER JOIN notes

Unmatched columns from LEFT or RIGHT tables returns NULL values

IFNULL or COALESCE can be used to return value

Example: SELECT a.col1, IFNULL(b.col2, 'No Match')

FROM T1 a LEFT JOIN T2 b USING (ID

JOINing/ORDERing Multiple Table Types

```
SELECT a.col1, b.col2, c.col3
FROM Table a LEFT JOIN View b USING (ID)
  JOIN TABLE(SELECT ID, MAX(val) col3 FROM T3) c
    ON a.ID = c.ID
ORDER BY a.col1, b.col2
```

Multiple Table Type notes

DB2 can sort the result set to satisfy ORDER BY requirement

TABLE is optional (required for table functions)

LATERAL can be used instead of TABLE

Vues – Réutilisation des jointures courantes

```
CREATE VIEW JoinView(orderyear, ordermonth, orderdate, country,
      last_name, part_name, supplier_name, quantity, revenue)
 AS SELECT t.year,t.month,i.orderdt,c.country,c.cust
        p.part,s.supplier,i.quantity,i.revenue
   FROM item_fact i
     INNER JOIN part_dim p ON (i.partid = p.partid)
     INNER JOIN time_dim t ON (i.orderdt = t.datekey)
     INNER JOIN cust_dim c ON (i.custid = c.custid)
     INNER JOIN supp_dim s ON (i.suppid = s.suppid)
   WHERE t.year > 2005
```


```
SELECT * FROM JoinView
  WHERE orderyear = 2009 AND ordermonth IN (10,11,12)
```

Les expressions table

- Instruction SELECT utilisée à la place d'une table/vue pour définir dynamiquement un ensemble logique de données

- Deux types d'expressions table
  - 1. Common Table Expression (CTE)
    - Définies avec la clause WITH avant le SELECT
    - Exemple :
      -
```
WITH temp1 AS (SELECT c1, c2 FROM t1 WHERE c1>0)
SELECT * FROM temp1
```

  - 2. Tables dérivées ou Nested Table Expression (NTE)
    - SELECT dans la clause FROM à la place d'une table/vue
    - Exemple :
      -
```
SELECT c1, c2 FROM (SELECT * FROM t1 WHERE c1>0)
```

Les CTE (Common Table Expression)

- Caractéristiques
  - Contiennent une instruction SELECT
  - Sont disponibles le temps de la requête
  - Plusieurs CTE possibles par requête
  - Peuvent se référencer entre elles
    - Dans l'ordre de leur déclaration

- Utilisation :
  - Diviser une requête complexe en plusieurs étapes
  - SQL récursif

- Syntaxe

```
WITH temp1 (id, nom) AS (SELECT c1, c2 FROM t1),
     temp2           AS (SELECT nom, count(*) FROM temp1)
SELECT * FROM temp2 ;
```

Les CTE – Penser « ensembles de données »

- On désire la liste du "top 10" des clients basé sur le total des ventes

- Requête sans CTE :

```
SELECT customer_name, SUM(sales) FROM SALES
      GROUP BY customer_name
      ORDER BY SUM(sales) DESC
      FETCH FIRST 10 ROWS ONLY
```

- Requête avec CTE :

```
WITH top10 (customer_name, total_sales) AS
(SELECT customer_name, SUM(sales) FROM SALES
              GROUP BY customer_name
              ORDER BY SUM(sales) DESC
              FETCH FIRST 10 ROWS ONLY)
SELECT * from top10
```

Les CTE – Penser « ensembles de données »

- On désire la liste des clients qui ont été dans le "top10" deux années consécutives

```
WITH top10_2010 (customer_name, total_sales) AS
 (SELECT customer_name, SUM(sales) FROM SALES
            WHERE YEAR=2010
            GROUP BY customer_name
            ORDER BY SUM(sales) DESC
            FETCH FIRST 10 ROWS ONLY) ,
      top10_2011 (customer_name, total_sales) AS
 (SELECT customer_name, SUM(sales) FROM SALES
            WHERE YEAR=2011
            GROUP BY customer_name
            ORDER BY SUM(sales) DESC
            FETCH FIRST 10 ROWS ONLY)
 SELECT Y1.customer_name,
    Y1.total_sales sales2010 Y2.total_sales sales2011
    FROM top10_2010 Y1 INNER JOIN top10_2011 Y2
     ON Y1.customer_name = Y2.customer_name
```

Le CTE – Simplification des requêtes

- Elimination des tables temporaires

Application Centric - 3 steps

```
DECLARE GLOBAL TEMPORARY TABLE
Step1 AS
(SELECT shipdate, customer, phone,
orderkey, linenumber
 FROM item_fact i, cust_dim c
 WHERE c.custkey=i.custkey AND
discount=0.08) WITH DATA;

DECLARE GLOBAL TEMPORARY TABLE
Step2 AS
(SELECT customer, phone, orderkey,
linenumber, year, quarter
 FROM Step1 , star1g.time_dim t
 WHERE t.datekey=shipdate ) WITH DATA;

SELECT * FROM Step2 ;
```

Data Centric – 1 step

```
WITH EXP1 AS
 (SELECT shipdate, customer, phone,
orderkey, linenumber
 FROM item_fact i, cust_dim c
 WHERE c.custkey = i.custkey AND
discount=0.08),

 EXP2 AS
 (SELECT customer, phone, orderkey,
linenumber, year, quarter
  FROM EXP1 , star1g.time_dim t
  WHERE t.datekey = shipdate)

SELECT * FROM EXP2 ;
```

5. Bonnes pratiques et erreurs à éviter

Assurer l'indépendance programme / fichier

- Ne pas faire de SELECT * à partir de tables
- Ne pas déclarer les objets SQL en spécification F de vos programmes
- Utiliser des SELECT sur des vues ou des appels à des procédures stockées pour accéder aux données
- Si la déclaration en spécification F est néanmoins nécessaire, déclarer celui-ci avec le mot-clé TEMPLATE

```
Femployes  if   e           k disk    template
```

Clés primaires, colonnes d'identité, clés uniques

- Chaque table (non statique) devrait avoir une clé primaire et cette clé primaire devrait être une colonne d'identité
  - Pour éviter la duplication de clés métier dans les tables
  - Pour éviter des impacts importants si la longueur de ces clés devait être modifiée
  - Pour définir les relations d'intégrité référentielle (Primary Key – Foreign Key)
    - Pas d'impact sur les programmes en cas de modification de la longueur de la clé, puisque non utilisée dans les programmes
- Créer des contraintes de clé unique ou des unique index pour assurer l'unicité de colonnes
  - Evite de le faire dans les programmes

Autres bonnes pratiques

- Déclarer des zones de type DATE à la place de zones numériques ou alphanumériques de longueur 8
  - Pour bénéficier des fonctions de gestion des dates de SQL
- Définir des valeurs par défaut pour les colonnes
  - CURRENT_DATE…
  - Permet d'avoir une valeur renseignée dans la table si la colonne a été omise lors de l'INSERT
- Définir la complexité des requêtes SQL en dehors des programmes en utilisant des vues
  - Lisibilité
  - Réutilisation
- → mais faire le CASTING dans la requête SQL

Pour en savoir plus

- Le nouveau redbook « Modernize your IBM i applications »

- Evénement Modernisation IBM i 2011
  - S3 – SQL embarqué : profitez de la puissance de SQL dans votre RPG
  - S11 – La modernisation de la base de données DB2 : passage des DDS à SQL

- Evénement Modernisation IBM i 2012
  - S3 – DB2/SQL - Un tour d'horizon des possibilités actuelles
  - S18 – DB2/SQL - Requêtes récursives - Cryptage de colonnes

- Evénement Modernisation IBM i 2013
  - S2 – DB2/SQL - Les procédures cataloguées
  - S17 – DB2/SQL - Trucs et astuces