Université IBM i 2017

17 et 18 mai – IBM Client Center de Bois-Colombes

S41 – Moderniser DB2 for i pas à pas

Jeudi 18 mai – 13h30-15h00

Philippe Bourgeois – IBM France

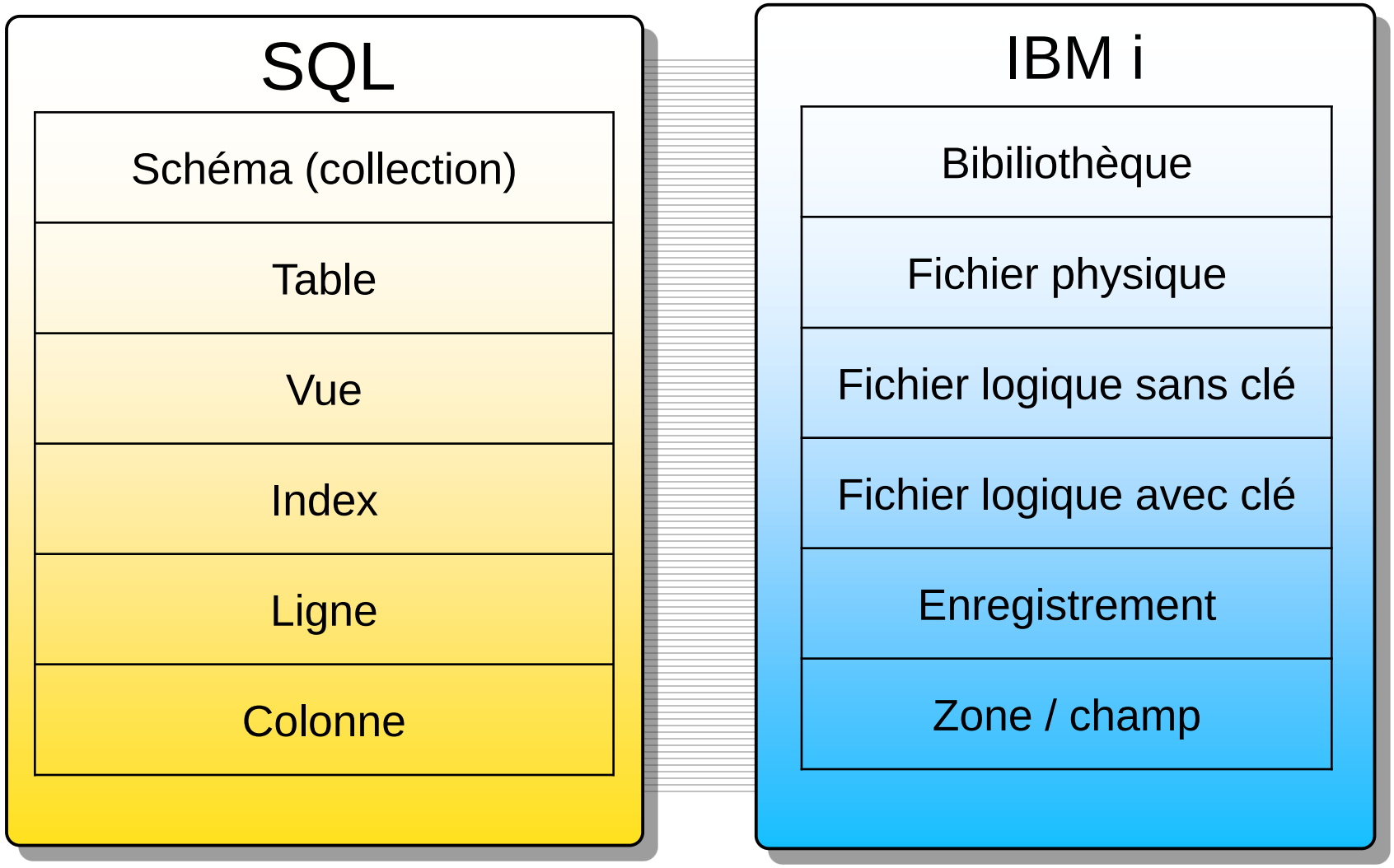
Plan de la présentation

- 1. Présentation de la méthodologie de modernisation de DB2
- 2. Formalisation du besoin
- 3. Définition du périmètre et éléments de volumétrie
- 4. Etapes et impacts d'une migration DDS→SQL manuelle
- 5. Démonstration

1. Méthodologie – Vocabulaire

- Table SQL
  - C'est un fichier base de données qui contient des données
  - Organisé en lignes / colonnes
  - Contient éventuellement un chemin d'accès (clé / clé primaire)
  - Ordre de création : CREATE TABLE
- Vue SQL
  - Une vue permet de présenter d'une certaine façon (sélection de lignes et/ou de colonnes) les données qui sont stockées dans une ou plusieurs tables
  - Les vues ne contiennent pas de données
  - Les vues ne peuvent pas être ordonnées (pas de chemin d'accès)
  - Ordre de création : CREATE VIEW
- Index SQL
  - Un index permet d'améliorer les performances (clé = chemin d'accès)
  - Un index permet d'assurer l'unicité d'une clé
  - Un index permet de présenter d'une certaine façon les données stockées dans une table, ordonnées sur une clé
  - Ordre de création : CREATE INDEX

1. Méthodologie – Vocabulaire



1. Méthodologie – Vocabulaire

- Clé primaire
  - Permet d'identifier de façon unique les enregistrements
  - UNIQUE et non nulle
  - Définit le chemin d'accès principal de la table
  - Une maximum par table
  - Se définit au niveau de la table
  - Exemple : l'ID employé sur la table des employés

- Clé unique
  - Permet de définir l'unicité d'une colonne
  - UNIQUE mais peut être nulle
  - Définit un chemin d'accès sur la table
  - Une ou plusieurs par table
  - Se définit au niveau de la table
  - Exemple : le numéro de sécurité sociale sur la table des employés

1. Méthodologie – Vocabulaire

- DDS (Data Description Specification)
  - Pour la création des PF et LF : sources DDS + commandes CRTPF et CRTLF
- RLA (Record Level Access)
  - Accès natif aux enregistrements de DB2 à partir de programmes RPG/COBOL : instructions READ, CHAIN, WRITE, UPDATE, DELETE…
  - 1 enregistrement à la fois
- SQL – DDL (Data Definition Language)
  - Création des tables, vues et index : CREATE TABLE, VIEW, INDEX…
- SQL – DML (Data Manipulation Language)
  - Insertion, mise-à-jour, suppression d'enregistrements : ordres INSERT, UPDATE, DELETE, MERGE
  - N enregistrements à la fois
  - Peuvent être inclus dans des programmes RPG/COBOL

1. Méthodologie – SQL versus DDS – Fonctionnalités

- Les avantages des **tables** par rapport aux **PFs** natifs
  - Noms longs (jusqu'à 128 caractères) et noms courts (maximum 10)
  - Plus de choix dans les types de colonnes (INTEGER, CLOB, BLOB, XML…)
  - Colonnes auto-incrémentées (colonne IDENTITY, objet SEQUENCE)
  - Colonnes auto-renseignées (date de dernière mise à jour, profil utilisateur…)
  - Tables temporelles (historisation automatique des données)
  - Cryptage des données (clause FIELDPROC)
  - Meilleure intégrité des données
  - Meilleures performances en lecture
  - Evolutions permanentes
  - Standard de l'industrie

- Les inconvénients
  - Pas de support des multi-membres (mais possibilité de créer un ALIAS sur un membre particulier)
  - "Moins bonnes" performances en écriture (à code équivalent)

1. Méthodologie – SQL versus DDS – Fonctionnalités

- Exemple de table SQL :

```
1  CREATE TABLE employes (matemp numeric(6, 0) not null primary key,
2                         nomemp char(25) not null,
3                         numsrv char(3))
4                rcdfmt fmtemp;
5
6   LABEL ON COLUMN employes (matemp is 'Matricule employé',
7                             nomemp is 'Nom employé',
8                             numsrv is 'Numéro de service');
```

Des exemples montrant les avantages des tables SQL par rapport aux PFs seront détaillés plus loin dans le support

- ~Equivalent DDS :

```
 Colonnes  . . :    1  71  Edition                                BIB2/QDDSSRC
 SEU==>                                                           EMPLOYES
 FMT PF .....A.........T.Name++++++RLen++TDpB......Functions++++++++++++++++++
        ***************  Début des données  ***********************************
0001.00                                              UNIQUE
0002.00       A          R FMTEMP
0003.00       A            MATEMP          6S 0      COLHDG('Matricule employé')
0004.00       A            NOMEMP         25         COLHDG('Nom employé')
0005.00       A            NUMSRV          3         ALWNULL
0006.00                                              COLHDG('Numéro de service')
0007.00       A          K MATEMP
        ****************  Fin des données  ************************************
```

1. Méthodologie – SQL versus DDS – Fonctionnalités

- Les avantages des **vues** par rapport aux **LFs** natifs
  - Une vue = une requête SELECT (CREATE VIEW AS SELECT…)
  - 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)
    - Clause CASE
    - Fonctions OLAP (CUBE, ROLLUP…)
    - Tous types de jointure et d'unions
    - CTE, sous-requêtes
    - Vues de vues
    - Standard de l'industrie

- Les inconvénients
  - Il n'est pas possible de définir une clé dans une vue
    - Pas d'équivalent SQL strict des LFs avec clé
      - Équivalence possible par les index pour les LFs non joints
  - Pas de support des multi-formats

1. Méthodologie – SQL versus DDS – Fonctionnalités

- Exemple de vue SQL :

```
CREATE VIEW empD11D21 AS SELECT nomemp, numsrv FROM employes
                                  WHERE numsrv IN ('D11', 'D21')
                RCDFMT fmtd11d21;
```

- Equivalent DDS :

```
R FMTD11D21                     PFILE(EMPLOYES)
  NOMEMP
  NUMSRV
S NUMSRV                        VALUES('D11'  'D21')
```

Des exemples montrant les avantages des vues SQL par rapport aux LFs seront détaillés plus loin dans le support

- Pas de clause ORDER BY dans le SELECT d'un CREATE VIEW (pas d'équivalent du K des LF)
  - C'est dans le SELECT final que l'on indiquera la clause ORDER BY

```
32 SELECT * FROM btsref.empD11D21 ORDER BY nomemp;
33
```

NOMEMP	NUMSRV
AZOULAY	D11
CIGNOT	D21
DUPONT	D11
DURAND	D21

1. Méthodologie – SQL versus DDS – Fonctionnalités

- Les avantages des **index** par rapport aux **LFs** natifs
  - Possibilité de créer des index dérivés
    - Possibilité de sélectionner certaines colonnes et certaines lignes
      - Pour avoir l'équivalent des LFs non joints
  - Possibilité des définir des expressions dans les valeurs de clé
  - Possibilité de créer des EVI (Encoded Vector Index)
    - Index très performants dans le cas de requêtes de type analytique (fonctions OLAP)

- Les inconvénients
  - Pas de possibilité d'ordonner les clés en FIFO, LIFO ou FCFO
  - Pas de jointures
  - Pas de possibilité d'utiliser les index dans une requête SELECT
    - En interactif
    - En SQL embarqué dans les programmes RPG / COBOL
    - Dans un QRYDFN ou par la commande RUNQRY (pour les index dérivés)
    - Dans un OPNQRYF (pour les index dérivés)

1. Méthodologie – SQL versus DDS – Fonctionnalités

- Exemples d'index SQL :

```
CREATE INDEX em_by_nom1 ON employes (nomemp) RCDFMT fmtemp;
```

```
R FMTEMP                    PFILE(EMPLOYES)
K NOMEMP
```

Equivalent DDS

```
CREATE INDEX em_by_nom2 ON employes (numsrv, nomemp) RCDFMT fmtemp;
```

```
R FMTEMP                    PFILE(EMPLOYES)
K NUMSRV
K NOMEMP
```

Equivalent DDS

```
CREATE INDEX em_by_nom3 ON employes (nomemp) RCDFMT fmtselfld ADD numsrv;
```

```
R FMTSELFLD                 PFILE(EMPLOYES)
  NOMEMP
  NUMSRV
K NOMEMP
```

Equivalent DDS

```
CREATE INDEX em_by_nom4 ON employes (nomemp) WHERE numsrv IN ('D11', 'D21') RCDFMT fmtselrec ADD numsrv;
```

```
R FMTSELRCD                 PFILE(EMPLOYES)
  NOMEMP
  NUMSRV
K NOMEMP
S NUMSRV                    VALUES('D11' 'D21')
```

Equivalent DDS

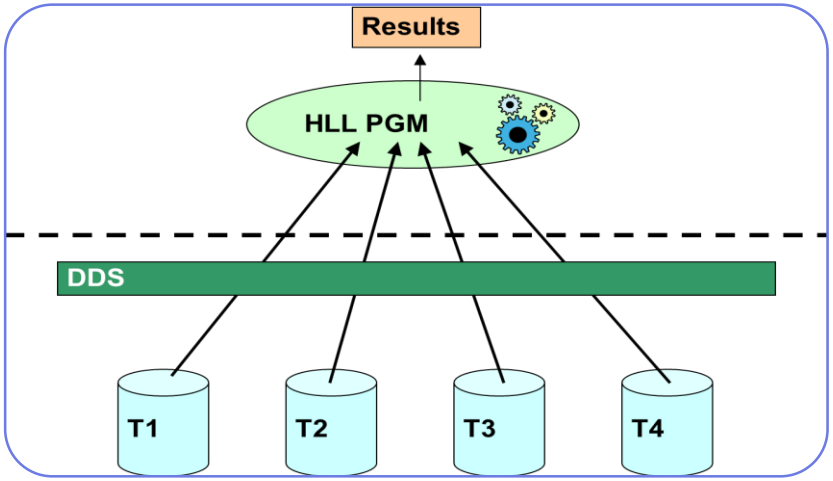
Des exemples montrant les avantages des index SQL par rapport aux LFs seront détaillés plus loin dans le support

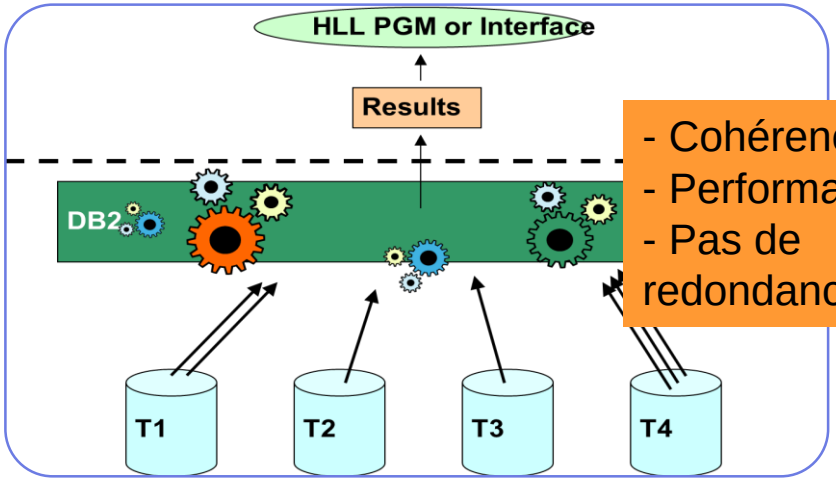
1. Méthodologie – SQL versus DDS – Mise en œuvre

- DDS
  - Création d'un source de type PF ou LF
  - Compilation de ce source par les commandes CRTPF / CRTLF pour créer les objets de type *FILE

- SQL
  - Création directe des objets de type *FILE par les commandes CREATE TABLE / VIEW / INDEX (ou par assistant d'IBM i Navigator)
  - Possibilité de placer les instructions CREATE OR REPLACE xxx dans un source (dans un fichier source ou dans l'IFS) puis d'utiliser la commande RUNSQLSTM pour exécuter les instructions et donc créer les objets de type *FILE
  - Ou alors encapsulation dans des procédures stockées SQL et exécution par CALL
  - Il est possible de générer le source SQL à partir des objets *FILE

1. Méthodologie – Approche Data Centric





- Cohérence
- Performances
- Pas de redondances

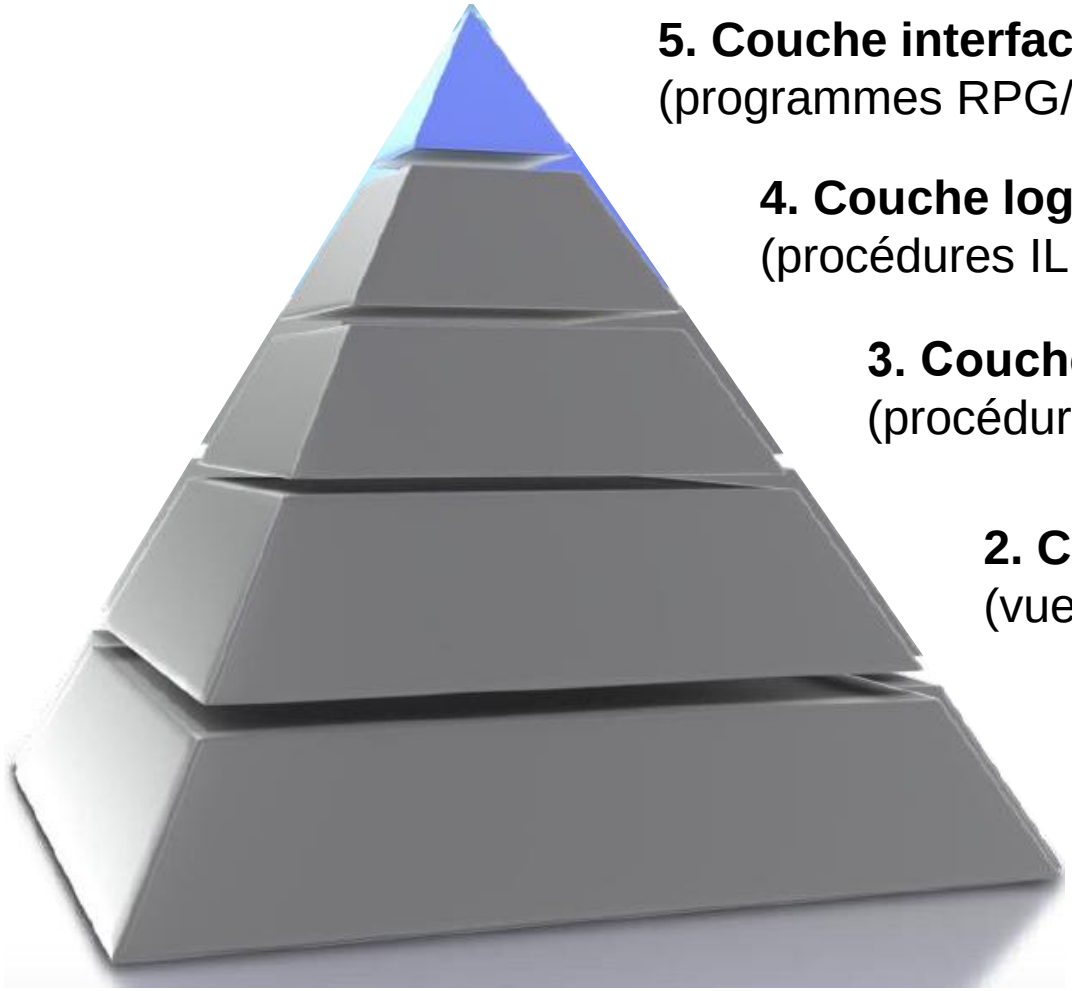
Application Centric

- Le programme s'occupe :
  - de l'ordre d'accès aux données
  - des règles métier
  - de l'intégrité des données
- Traitement d'un enregistrement à la fois (RLA : Record Level Access)

Data Centric

- DB2 s'occupe :
  - de l'ordre d'accès aux données
  - des règles métier
  - de l'intégrité des données
- Traitement d'ensembles d'enregistrements

1. Méthodologie – Architecture Data Centric



5. Couche interface utilisateur
(programmes RPG/COBOL, Java, etc.)

4. Couche logique métier
(procédures ILE)

3. Couche d'accès aux données
(procédures stockées, fonctions)

2. Couche logique de données
(vues)

1. Couche physique de données
(tables, index)

1. Méthodologie – Architecture Data Centric

1. Couche physique de données
(tables, index)

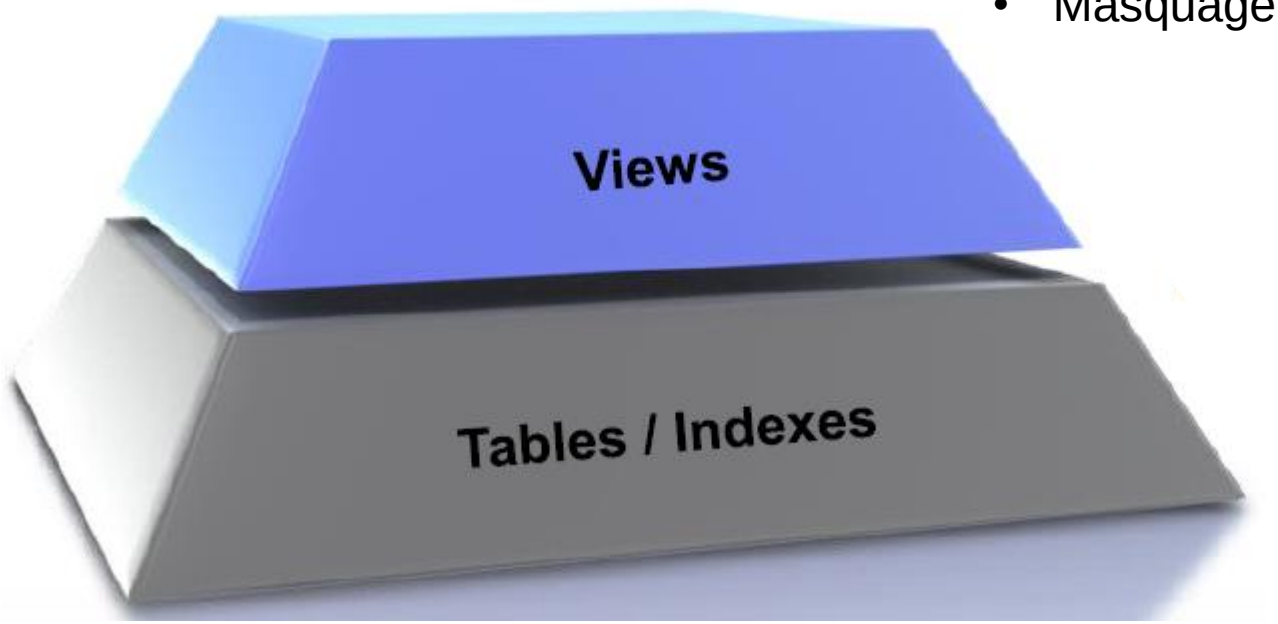
- Tables normalisées (3ème forme normale)
- Contraintes de clé primaire et d'unicité
- Contraintes de clé étrangère (intégrité référentielle)
- Contraintes de vérification (règles métier communes)

1. Méthodologie – Architecture Data Centric

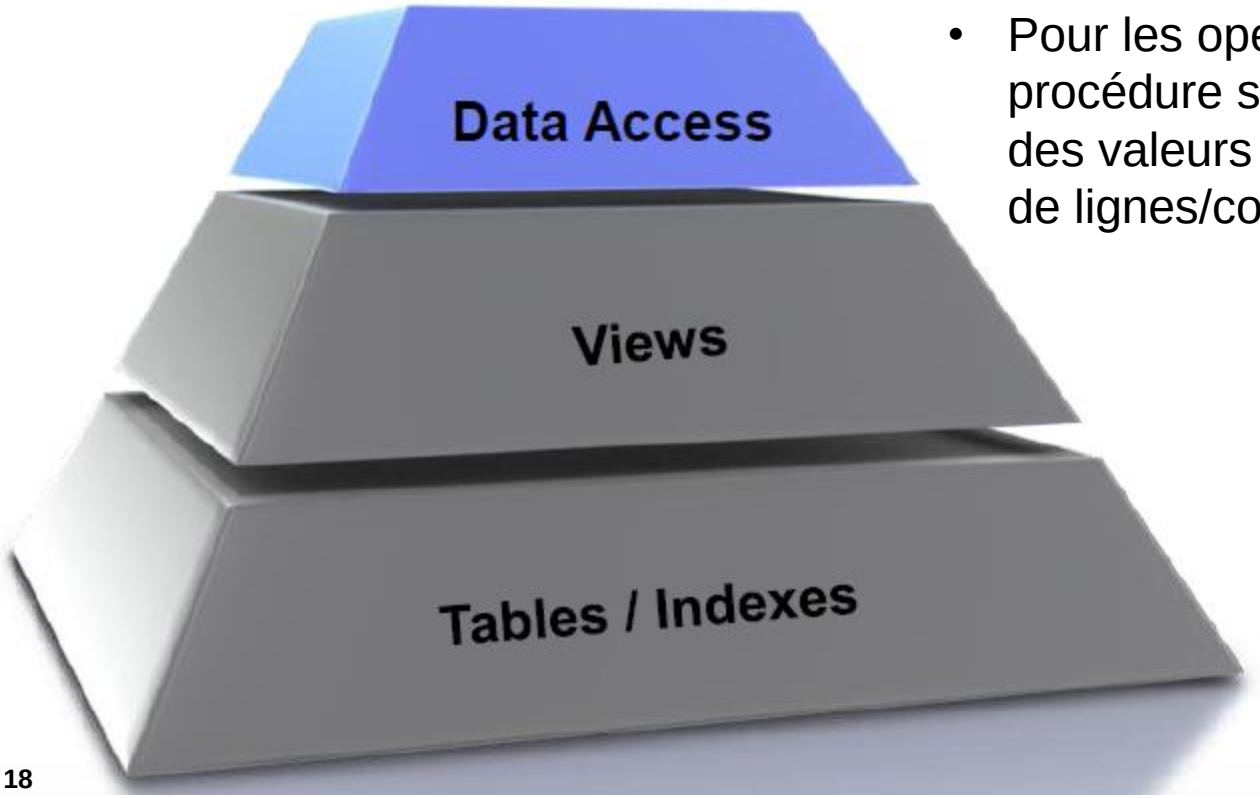
2. Couche logique de données
(vues, triggers INSTEAD OF)

→ masquage de la complexité

- Jointures
- Dénormalisation
- Zones calculées, groupage, OLAP…
- Masquage de données



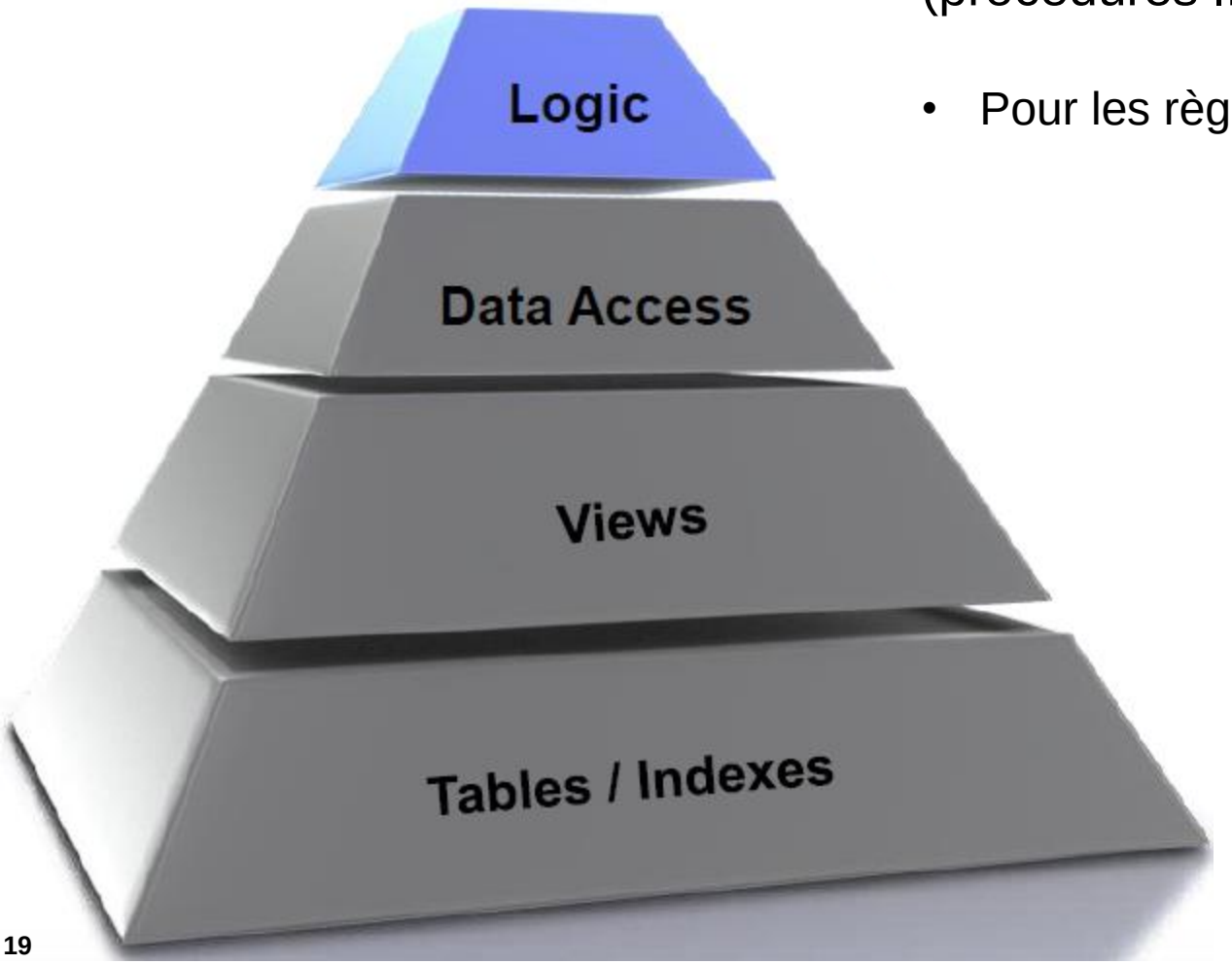
1. Méthodologie – Architecture Data Centric



3. Couche d'accès aux données
(procédures stockées)

- Pour les opérations INSERT, UPDATE et DELETE
- Pour les opérations SELECT (une procédure stockée peut retourner des valeurs simples ou un ensemble de lignes/colonnes (Result Sets))

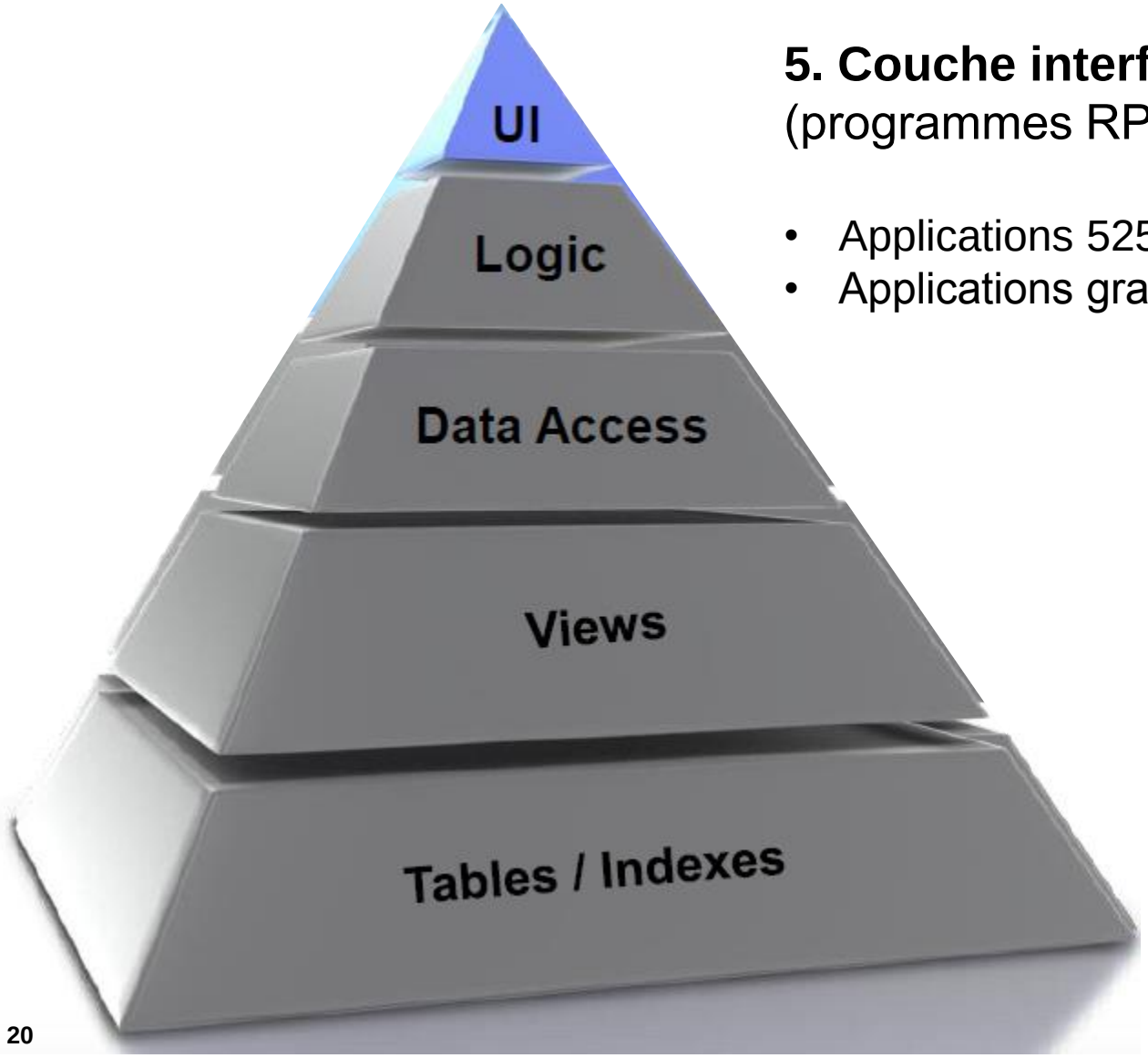
1. Méthodologie – Architecture Data Centric



4. Couche logique métier
(procédures ILE)

- Pour les règles métier spécifiques

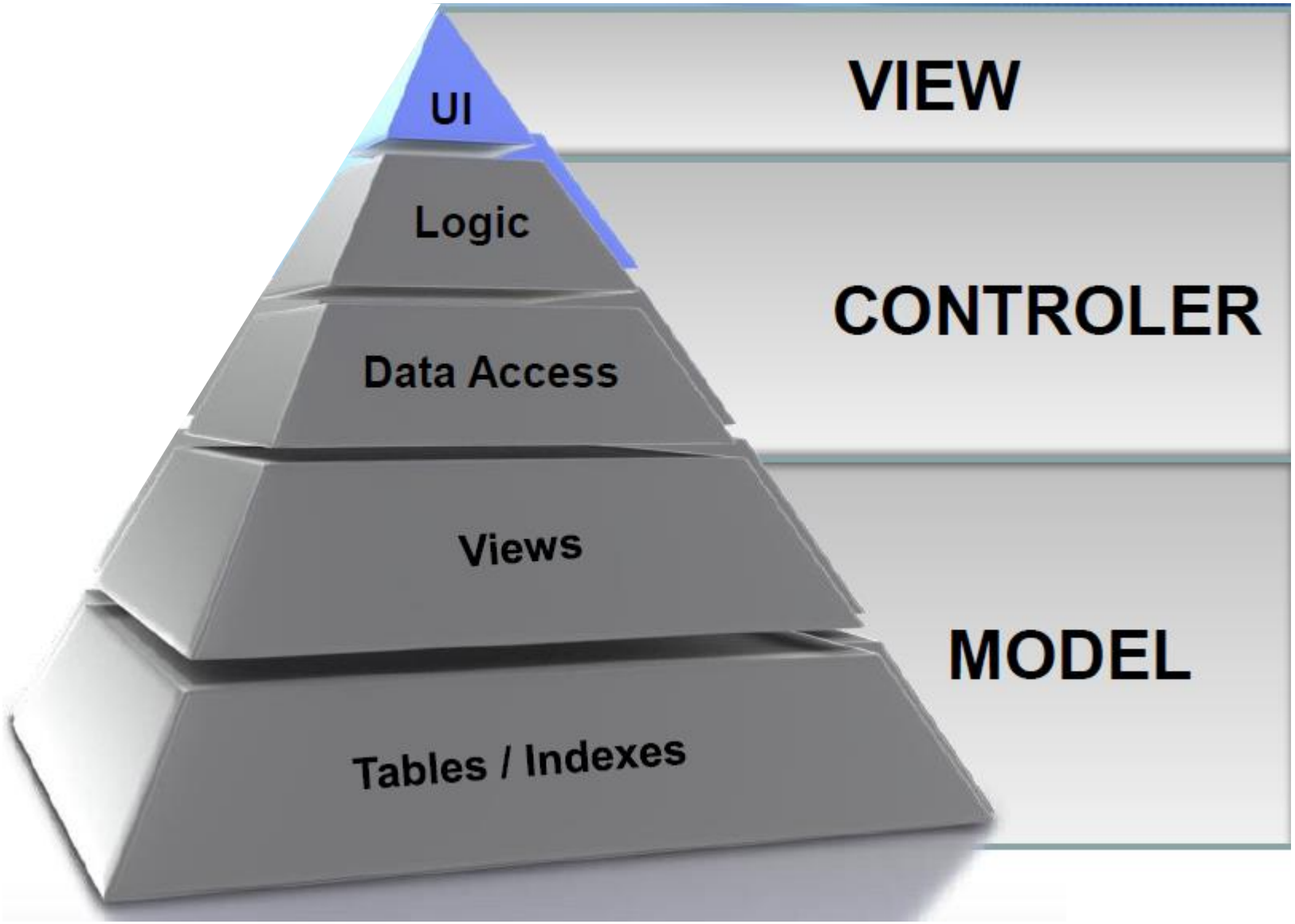
1. Méthodologie – Architecture Data Centric



5. Couche interface utilisateur
(programmes RPG/COBOL, Java…)

- Applications 5250
- Applications graphiques, Web, mobile…

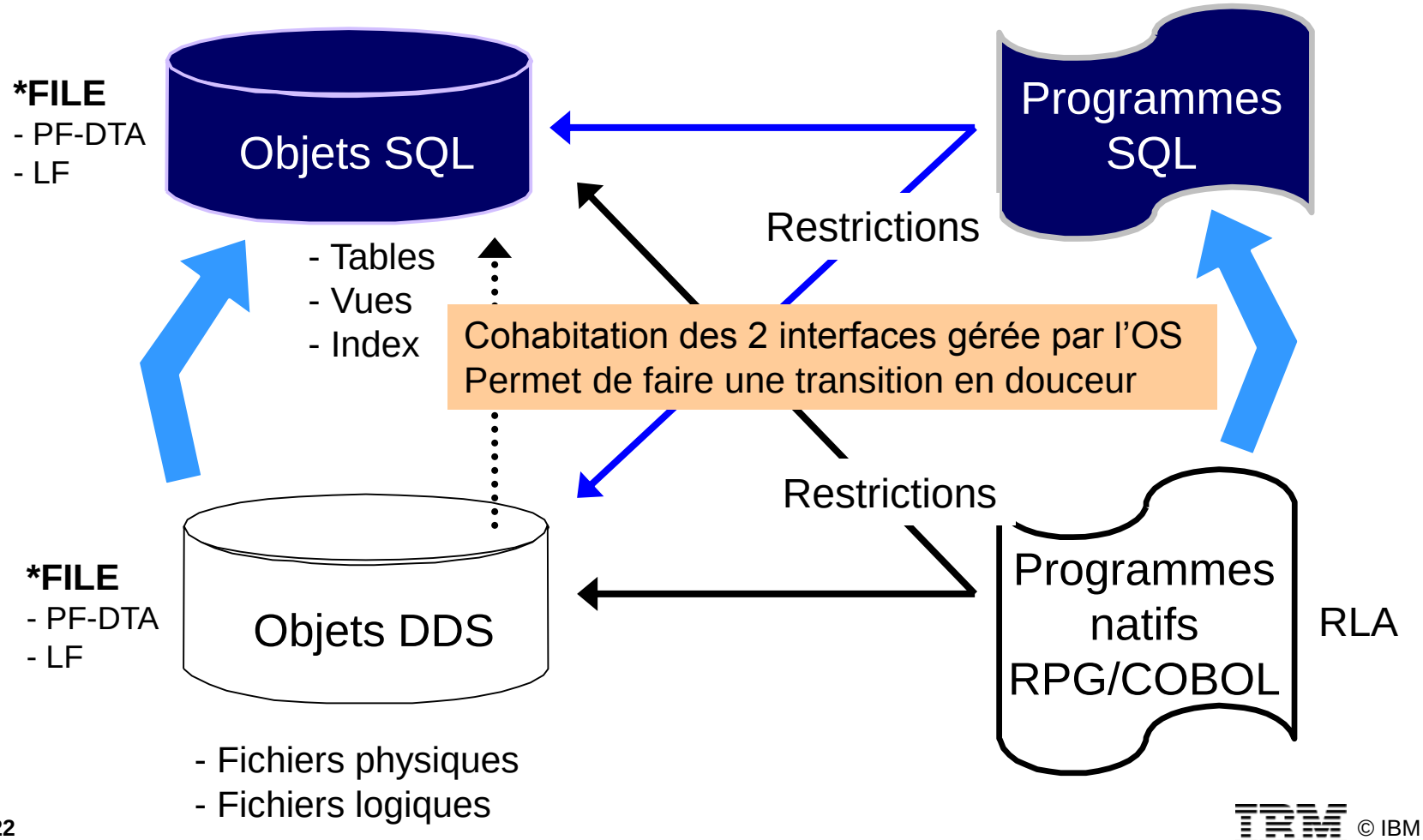
1. Méthodologie – Architecture Data Centric – Modèle MVC



1. Méthodologie – Les 2 étapes de modernisation

1. Modernisation des objets DB2

2. Modernisation des accès à DB2



2. Formalisation du besoin de modernisation

- a. Quelle est votre situation actuelle ?
  - a. Base de données DB2 for i
    - DDS :
      - Fichier(s) de référence ?
      - Clés dans les PFs ?
      - Clés dans les LFs ?
      - LFs avec partage de format ?
      - LFs sans partage de format ?
      - LFs avec jointures ?
    - SQL :
      - Tables ?
      - Vues ?
      - Index ?
      - Routines SQL ?
    - Contraintes ?
    - Triggers ?
    - Modification de la structure par CHGPF ?

2. Formalisation du besoin de modernisation

- a. Quelle est votre situation actuelle ?
  - b. Fichiers écran (DSPF) et imprimante (PRTF)
    - REF ou REFFLD à partir de tables ou du fichier de référence ?
  - c. Programmes
    - RPG ?
      - SQL dans le RPG ?
    - COBOL ?
      - SQL dans le COBOL ?
    - CLP ?
      - RUNSQL ?
    - Autre ? (Java, PHP, Windev / Webdev, .Net ?)
      - Méthode d'accès à DB2 for i ?
  - d. Autres ?
    - QRYDFN ?
    - Autres ?

2. Formalisation du besoin de modernisation

- b. Quels sont les objectifs attendus par la modernisation de DB2
  - a. Revoir le modèle de données ?
    - Eliminer la redondance des données
      - Données redondantes au sein d'une table ?
      - Données redondantes inter tables ?
    - Regrouper les informations
      - Informations d'une entité scindées en plusieurs tables ?
      - Noms de colonnes différents ?
      - Même table dans des bibliothèques différentes ?
        - Même nom de table mais données différentes
  - b. Transformer certaines colonnes :
    - Dates alphanumériques/numériques en zones date ?
    - Noms supérieurs à 10 caractères ?
    - Longueur insuffisante ?
    - Nouveau nom ?
    - Nouveau type ? (Unicode ?)
    - Colonnes auto-incrémentées ?
    - Autre ?

2. Formalisation du besoin de modernisation

- b. Quels sont les objectifs attendus par la modernisation de DB2 :
  - c. Eliminer les données invalides :
    - Dates invalides, zones numériques invalides ?
    - Orphelins ?
  - d. Eliminer la duplication des règles d'intégrité au niveau des applications :
    - Mettre en place des relations entre les tables ?
    - Mettre en place des règles de validation des données ?
  - e. Ajouter de nouvelles colonnes :
    - Types n'existant qu'en SQL ?
    - Auto-renseignées ?
    - Noms supérieurs à 10 caractères ?
  - f. Documenter la base de données ?
    - Par un modèle graphique
  - g. Sécuriser les accès à la base de données :
    - Restreindre les accès au niveau ligne et colonne ?

2. Formalisation du besoin de modernisation

- Quels sont les objectifs attendus par la modernisation de DB2 for i :
  - h. Protéger les données sensibles ?
    - Par des fonctions de cryptage
  - i. Eliminer la duplication des règles métier au niveau des applications :
    - En implémentation de la logique métier partagée ?
  - j. Assurer une indépendance entre la couche physique de données, les accès base de données et les applicatifs ?
  - k. Améliorer les performances d'accès aux données ?
  - l. Moderniser les compétences des équipes ?
  - m. Assurer la portabilité de la base de données ?
  - n. Autre ?

3. Définition du périmètre et éléments de volumétrie

- A. Sur quel périmètre devra s'appliquer la modernisation ?
  - Certaines tables bien précises ? Bibliothèques ? Applications ?

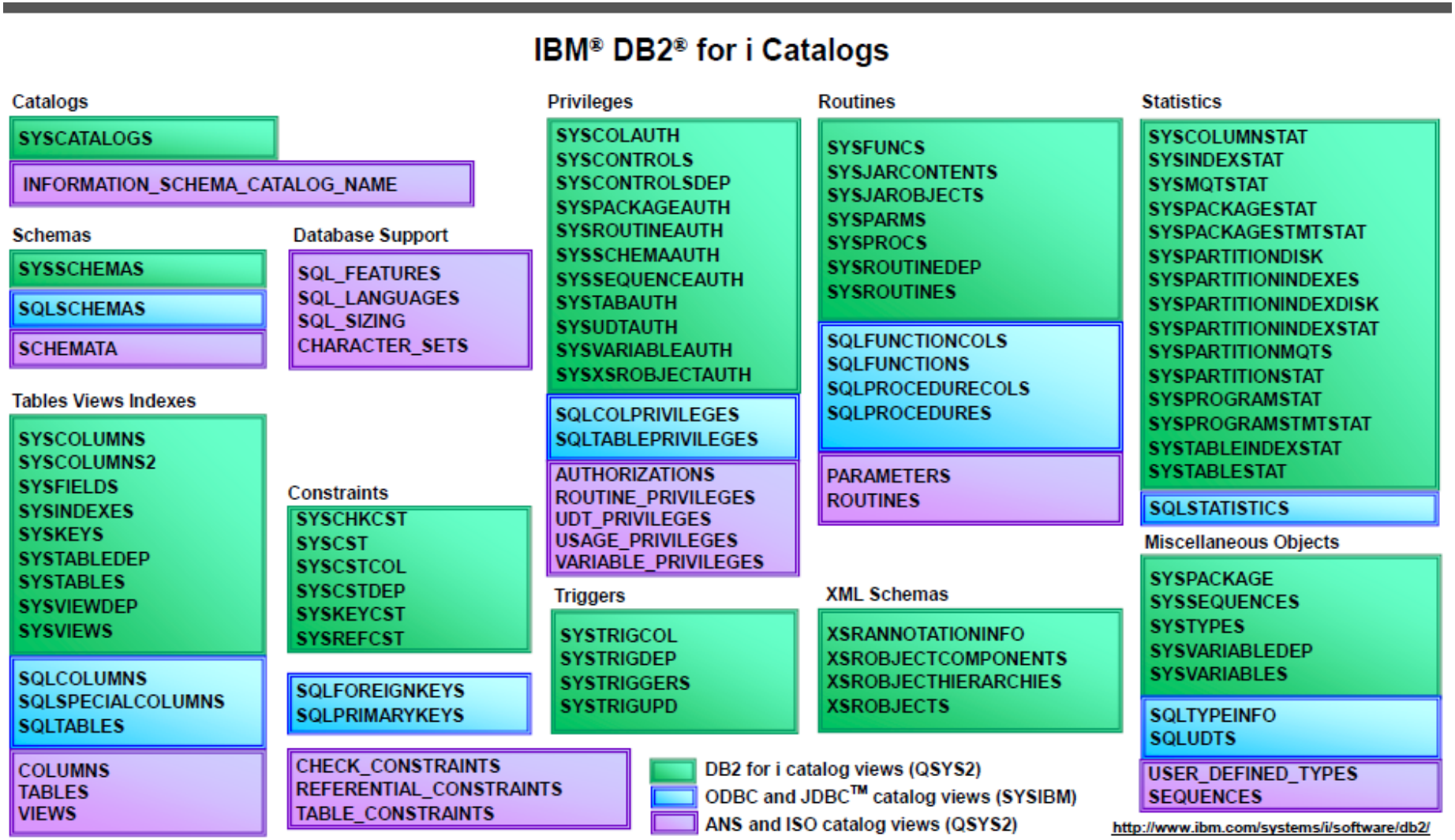
- B. Volumétrie
  - a. Nombre de PFs concernés par la migration ?
    - Eliminer les fichiers de travail
  - b. Puis éventuellement, pour chacun des PFs :
    - Nombre de LFs dépendants ?
    - Nombre de DSPF / PRTF faisant référence au PF ?
    - Nombre de programmes faisant référence au PF ou au(x) LFs ?
  - Comment ?
    - Utilisation du catalogue système et de commandes CL ou d'un outil

```
-- a3. Combien y-a-t-il de PFs dans la bib POTDB2 ?
 SELECT COUNT(*) AS "Nombre de PFs dans POTDB2" FROM qsys2.systables
 WHERE table_schema = 'POTDB2' AND file_type = 'D' AND table_type = 'P'
 AND table_name NOT LIKE 'Q%';

-- a4. Quels sont les PFs de la bib POTDB2 ?
 SELECT * FROM qsys2.systables WHERE table_schema ='POTDB2' AND file_type = 'D' AND table_type = 'P'
 AND table_name NOT LIKE 'Q%' ORDER BY table_name;
```

3. Définition du périmètre et éléments de volumétrie

- http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_73/db2/rbafzcatalog.htm



4. Etapes et impacts d'une migration DDS→SQL manuelle

- a. Audit des problèmes potentiels de conversion des PFs
  - Les PFs avec des données corrompues
  - Les PFs avec clé non unique
  - Les PFs multi-membres
  - Les mots-clés DDS non supportés

- b. Génération des sources DDL à partir des objets PFs
  - Analyser les paramètres de création/compilation (CRTPF) qui seront différents en SQL
  - Récupération des caractéristiques des PFs
    - Propriétaire, droits, journalisation, contraintes, audit, triggers

- c. Choix de la méthode de migration
  - Méthode avec Surrogate (LF intermédiaire)
  - Méthode sans Surrogate (pas de LF intermédiaire)

4. Etapes et impacts d'une migration DDS→SQL manuelle

- d. Création des objets SQL
  - Création des tables à la place des PFs
    - Eventuellement modifier les attributs des tables
  - Copier les données
  - Si méthode Surrogate, création du LF intermédiaire
  - Optionnellement création des vues et index à la place des LFs
    - Après audit des problèmes potentiels de conversion des LFs et génération des sources DDL
- e. Modification éventuelle des DSPF, PRTF et programmes
  - Si méthode non Surrogate : DSPF, PRTF et programmes impactés
  - Si méthode Surrogate : éventuellement certains programmes CL
- f. Analyse et modification éventuelle des objets connexes
  - Analyses QUERY/400 (objets de type *QRYDFN)
  - Programmes DFU
  - Routines SQL (procédures, fonctions, triggers)

4a. Audit des problèmes potentiels de conversion des PFs

- Les PFs avec des données corrompues
  - Zones numériques invalides, dates invalides, orphelins…
  - Identification des fichiers et des données, correction
- Les PFs avec clé non unique
  - Choix ? (conservation, conversion (doublons ?), index/LF...)
- Les PFs multi-membres
  - Choix ? (consolider, nouvelle zone…)
- Les mots-clés DDS non supportés
  - RANGE, VALUES, COMP, EDTCDE, DATFMT…
  - Choix ? (conservation, report DSPF/PRTF…)

4b. Génération des sources DDL à partir des objets PFs

- Outils :
  - Procédure GENERATE_SQL
  - System i Navigator / IBM Navigator for i
  - API QSQGNDDL
  - Outil tiers

- Gérer la récupération des **attributs** des objets
  - Mots-clés de compilation (REUSEDLT, FRCRATIO, WAITFILE…)
  - Propriétaire, droits, journalisation, contraintes, audit, triggers
  - Des produits tiers peuvent automatiser ce travail

4b. Génération des sources DDL à partir des objets PFs

- Procédure GENERATE_SQL

```
CALL qsys2.generate_sql('ACTEURP', 'DBDCOR', 'TABLE', 'QSQLSRC', 'DBDCOR', 'ACTEURP',
STATEMENT_FORMATTING_OPTION=>'0',
QUALIFIED_NAME_OPTION=>'1',
HEADER_OPTION=>'0',
CREATE_OR_REPLACE_OPTION=>'1');
```

```
CREATE OR REPLACE TABLE ACTEURP (
--  SQL150B   10   REUSEDLT(*NO) de la table ACTEURP de DBDCOR ignoré.
  CODACT DECIMAL(3, 0) NOT NULL DEFAULT 0 ,
  NOMACT CHAR(20) CCSID 297 NOT NULL DEFAULT '' ,
  PREACT CHAR(15) CCSID 297 DEFAULT NULL ,
  NATACT CHAR(5) CCSID 297 DEFAULT NULL ,
  CONSTRAINT QSYS_ACTEURP_00001 PRIMARY KEY( CODACT ) )

  RCDFMT ACTF       ;

LABEL ON TABLE ACTEURP
  IS 'Fichier des acteurs' ;

LABEL ON COLUMN ACTEURP
( CODACT IS 'Code                acteur' ,
  NOMACT IS 'Nom acteur' ,
  PREACT IS 'Prénom acteur' ,
  NATACT IS 'Nationalité' ) ;
```

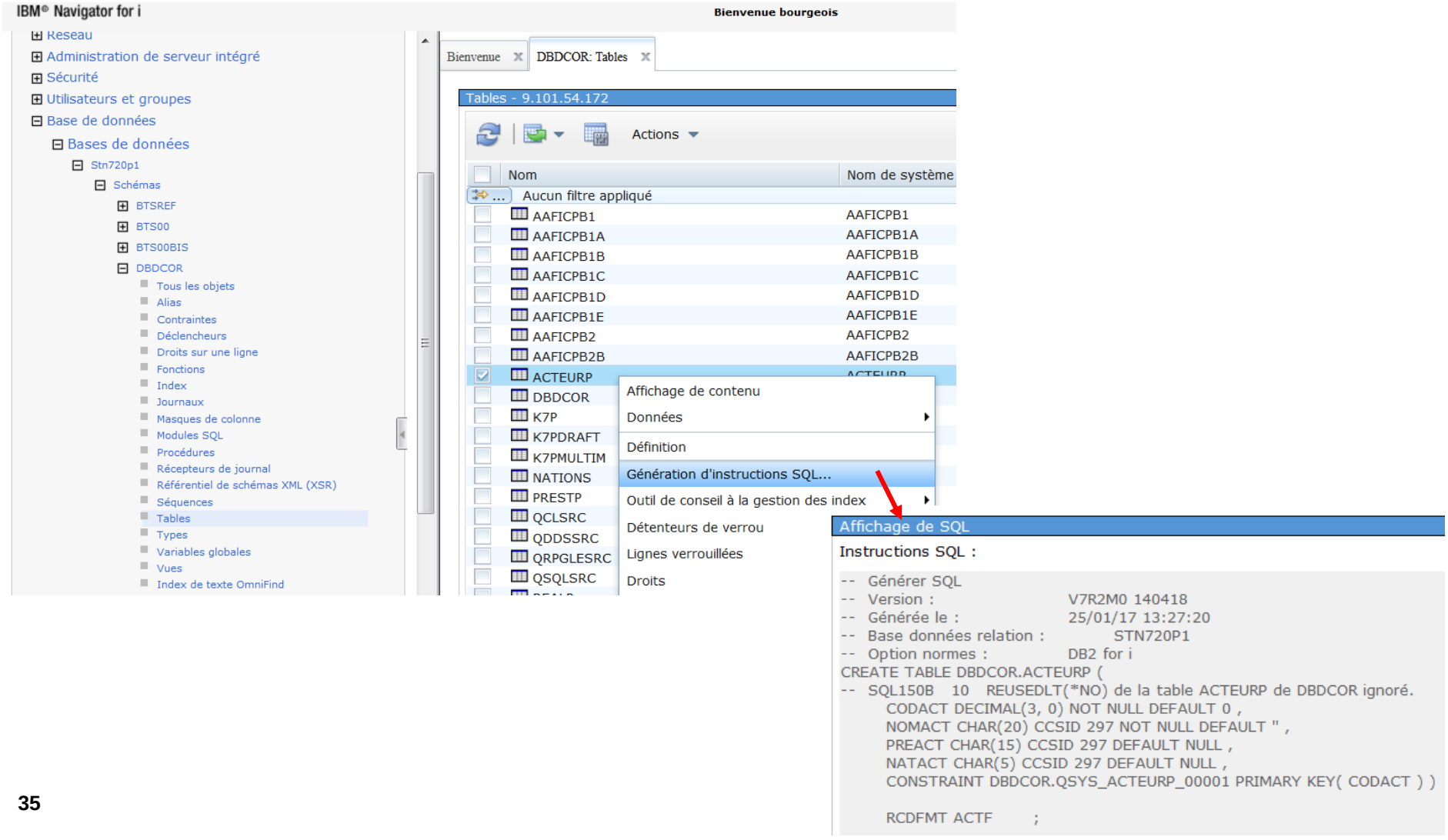
```
LABEL ON COLUMN ACTEURP
( CODACT TEXT IS 'Code acteur' ,
  NOMACT TEXT IS 'Nom acteur' ,
  PREACT TEXT IS 'Prénom acteur' ,
  NATACT TEXT IS 'Nationalité' ) ;

GRANT DELETE , INSERT , SELECT , UPDATE
ON ACTEURP TO PUBLIC ;

GRANT ALTER , DELETE , INDEX , INSERT , REFERENCES ,
SELECT , UPDATE
ON ACTEURP TO BOURGEOIS WITH GRANT OPTION ;
```

4b. Génération des sources DDL à partir des objets PFs

- IBM Navigator for i



4b. Génération des sources DDL à partir des objets PFs

- Ce qui est récupéré :
  - Le nom du format
  - La valeur par défaut (mot-clé DFT dans les DDS)
  - La valeur nulle (ALWNULL)
  - L'entête de colonne (COLHDG)
  - Le texte descriptif de colonne (TEXT)
  - Le texte descriptif du PF (paramètre TEXT de la commande CRTPF)
  - Les droits
  - Les contraintes
  - Les triggers SQL

- Ce qui n'est pas récupéré
  - Les attributs de compilation (warning)
  - Les triggers système (warning)
  - La structure de journalisation
    - Les tables SQL sont automatiquement journalisées si la bibliothèque est journalisée ou si elle contient un journal QSQJRN (ou un autre nom indiqué dans une DTAAREA QDFTJRN)

4c. Choix de la méthode de migration DDS->SQL

- Les questions à se poser
  - **Q1**. Est-ce que la table SQL aura la même structure que le PF ?
  - **Q2**. Est-ce que le PF est utilisé directement dans les programmes ?
  - **Q3**. Voulez-vous une conversion sans impact sur l'existant ?

- Les méthodes possibles

Q1	Q2	Q3	Avec Surrogate	Sans Surrogate
OUI	OUI/NON	OUI/NON		X
OUI/NON	**NON**	OUI/NON		X
NON	**OUI**	**OUI**	X	
NON	OUI	NON		X

4c. Choix de la méthode de migration DDS→SQL

- 1. Méthode **avec** Surrogate
  - Table SQL à la place du PF, avec un *nom différent*
    - Avec possibilité d'une structure différente, de nouveaux champs et attributs
    - Les nouveaux programmes peuvent bénéficier des ces nouveautés
  - Nouveau LF (dit surrogate) pointant sur la table
    - Avec le même nom et le même level ID que le PF
  - Modification des LFs pour qu'ils pointent sur la table
  - Vues et index à la place des LFs (optionnel)
  - Pas de modification / recompilation des DSPF, PRTF et programmes existants (sauf éventuellement les programmes CL)

  → + *: conversion rapide, pas d'impact sur l'existant*

  → - *: introduit une couche supplémentaire*

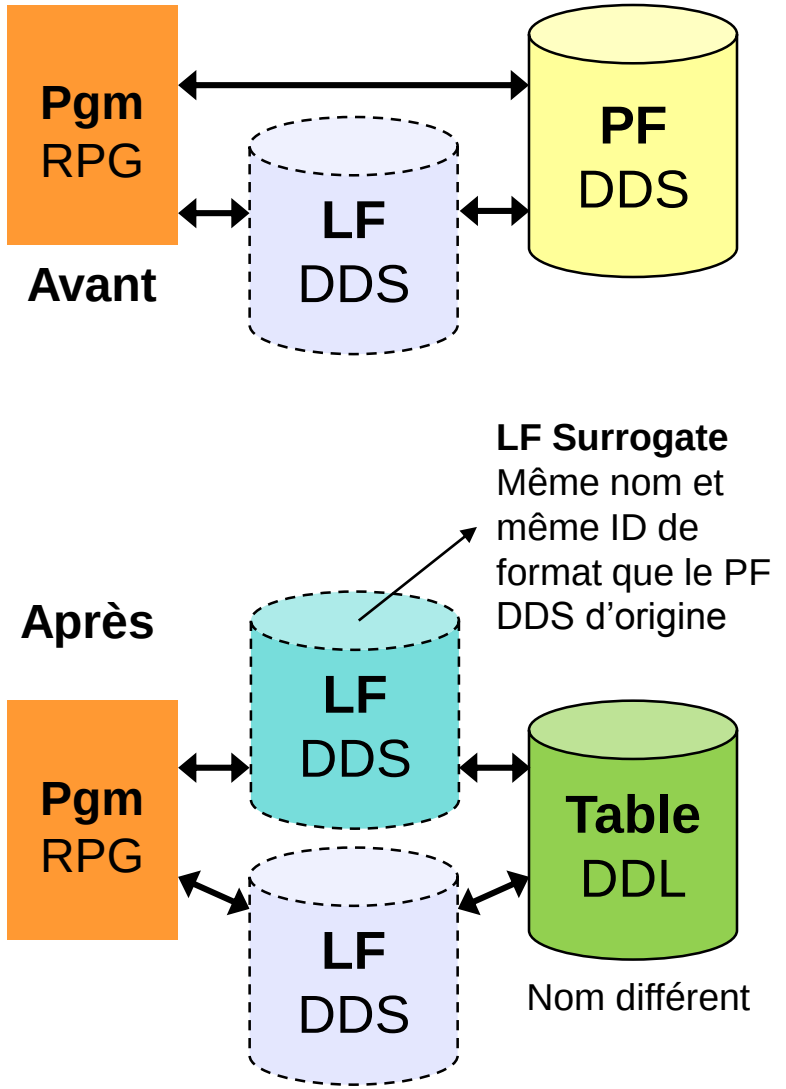
- 2. Méthode **sans** Surrogate
  - Table à la place du PF avec le *même nom*
    - Possibilité d'une structure différente, de nouveaux champs et attributs
  - Vues et index à la place des LFs (optionnel)
  - Si modification de la structure de la table : éventuellement modification / recompilation des DSPF, PRTF et programmes existants

  → + *: pas de couche supplémentaire*

  → - *: nécessite éventuellement la modification des objets dépendants*

4c. Méthode avec Surrogate

- 1. Convertir le source du PF en TABLE en lui donnant un nouveau nom
- 2. Créer la table et migrer les données
- 3. Créer un LF surrogate avec le même nom que le PF d'origine
  - A partir du PF, modifier le type en LF et ajouter le mot-clé PFILE
- 4. Modifier les LFs :
  - Afin qu'ils référencent la table SQL à la place du PF
    - Mot-clé PFILE
  - Afin qu'ils continuent à partager le format (pour ceux qui partageaient le format du PF)
    - Mot-clé FORMAT
- 5. Transformer les LFs en objets SQL (optionnel)
- 6. Modifier si besoin les programmes qui utilisent des commandes CL de gestion des PFs



LF transformé pour s'appuyer sur la table DDL

4c. Méthode avec Surrogate

```
=> _____                                                    CLIENTS
PF .....A..........T.Name++++++RLen++TDpB......Functions+++++++++++++++++
   ***************  Début des données  ***********************************
00       A                                     UNIQUE
00       A           R FCLIENTS
00       A             CLI_NUM         6  0    COLHDG('Numéro' 'client')
00       A             CLI_NOM        25       COLHDG('Nom client')
00       A             CLI_PRENOM     15       COLHDG('Prénom client')
00       A             CLI_ADR        50       COLHDG('Adresse client')
00       A           K CLI_NUM
```

```
SEU==> _____                                                CLIENTS
FMT LF .....A..........T.Name++++++.Len++TDpB......Functions+++++++++++++++++
   ***************  Début des données  ***********************************
0001.00      A                                     UNIQUE
0002.00      A           R FCLIENTS                PFILE(CLIENTSSQL)
0003.00      A             CLI_NUM         6  0    COLHDG('Numéro' 'client')
0004.00      A             CLI_NOM        25       COLHDG('Nom client')
0005.00      A             CLI_PRENOM     15       COLHDG('Prénom client')
0006.00      A             CLI_ADR        50       COLHDG('Adresse client')
0007.00      A           K CLI_NUM
```

```
=> _____                                                    CLINOM
LF .....A..........T.Name++++++.Len++TDpB......Functions+++++++++++++++++
   ***************  Début des données  ***********************************
00       A           R FCLIENTS                PFILE(CLIENTS)
00       A           K CLI_NOM
```

```
=> _____                                                    CLINOM
LF .....A..........T.Name++++++.Len++TDpB......Functions+++++++++++++++++
   ***************  Début des données  ***********************************
00       A           R FCLIENTS                PFILE(CLIENTSSQL)
00       A                                     FORMAT(CLIENTS)
00       A           K CLI_NOM
```

4c. Méthode sans Surrogate

- 1. Convertir le PF en TABLE en gardant le même nom
- 2. Créer la table et migrer les données
- 3. Transformer les LFs en objets SQL (optionnel)
- 4. Modifier si nécessaire les objets dépendants

4c. Transformation des LFs en objets SQL

- Les LFs **sans** clé peuvent être transformés en vues
  - Pas de support des LFs multi-formats
  - Support des LFs joints

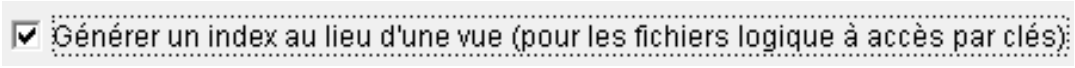
  → Audit nécessaire

- Les LFs **avec** clé peuvent être transformés en index
  - Pas de support des LFs multi-formats
  - Pas de support des LFs joints

  → Audit nécessaire
  - Support des LFs qui sont codés avec
    - Reprise de certaines zones du PF
    - Sélections / omissions
  - Attention : les index ne peuvent pas être utilisés comme source de données SQL

- Outils : procédure GENERATE_SQL, System i Navigator / IBM Navigator for i, API QSQGNDDL, outil tiers (Xcase…)

4c. Transformation des LFs en objets SQL

- Conversion des LFs **avec** clé
  - Logique **avec** clé → **index**
  - Dans System i Navigator : 


```
R FCLIENTS                      PFILE(CLIENTS)
K CLI_NOM
```

```
CREATE INDEX CLINOM ON CLIENTS (CLI_NOM)
RCDFMT FCLIENTS ;
```

```
R FCLINOM2                      PFILE(CLIENTS)
  CLI_PRENOM
  CLI_NOM
K CLI_NOM
```

```
CREATE INDEX CLINOM2 ON CLIENTS (CLI_NOM)
RCDFMT FCLINOM2 ADD CLI_PRENOM;
```

```
R FCLINOM3                      PFILE(CLIENTS)
  CLI_PRENOM
  CLI_NOM
K CLI_NOM
S CLI_NOM                       COMP(LT 'N')
```

```
CREATE INDEX CLINOM3 ON CLIENTS (CLI_NOM)
WHERE CLI_NOM < 'N' RCDFMT FCLINOM3 ADD CLI_PRENOM;
```

Les index peuvent être utilisés dans les programmes RPG/COBOL à la place des LF mais pas comme source de données en SQL

4c. Transformation des LFs en objets SQL

- Conversion des LFs **sans** clé
  - Logique **sans** clé → **vue**
  - Dans System i Navigator :


```
R FSINCLI                       JFILE(CLIENTS CONTRATS SINISTRES)
J                               JOIN(1 2)
                                JFLD(CLI_NUM CLI_NUM)
J                               JOIN(2 3)
                                JFLD(CONT_NUM CONT_NUM)
  CLI_NUM                       JREF(1)
  CONT_NUM                      JREF(2)
  SIN_NUM                       JREF(3)
  MNT_REP
```

```
CREATE VIEW SIN_CLI2 (CLI_NUM , CONT_NUM , SIN_NUM , MNT_REP ) AS
 SELECT Q01.CLI_NUM , Q02.CONT_NUM , Q03.SIN_NUM , Q03.MNT_REP
 FROM PBSQL.CLIENTSOLD AS Q01 INNER JOIN
 CONTRATS AS Q02 ON ( Q01.CLI_NUM = Q02.CLI_NUM ) INNER JOIN
 SINISTRES AS Q03 ON ( Q02.CONT_NUM = Q03.CONT_NUM )
 RCDFMT FSINCLI;
```

```
CREATE INDEX SIN_CLI2_QSQGNDDL_00001
ON CONTRATS (CLI_NUM);

CREATE INDEX SIN_CLI2_QSQGNDDL_00002
ON SINISTRES (CONT_NUM);
```

4d. Création des objets SQL

- Commande RUNSQLSTM
  - A partir des sources SQL qui sont dans un membre source ou dans l'IFS
- Création de procédures stockées SQL
  - Puis appel par un CALL
- Copie des données par SQL, par CPYF, par programme

Amélioration des objets SQL après conversion

- Objectif : mettre en place une architecture Data-Centric (faire faire le maximum de choses par DB2) pour
  - Assurer l'intégrité des données quelle qu'en soit l'interface d'accès
  - Alléger les programmes
  - Eliminer la redondance des contrôles
  - Faire de DB2 une "vraie" base de données

- Comment ?
  - Mettre en place des contraintes d'intégrité
  - Définir des clés auto-incrémentées
  - Définir des colonnes auto-renseignées
  - Définir les tables comme tables temporelles
  - Mettre en place une sécurité niveau ligne / colonne

- Profitez-en pour
  - Donner des noms longs
  - Utiliser de nouveaux types de données
  - Utiliser la clause CREATE **OR REPLACE** TABLE / VIEW

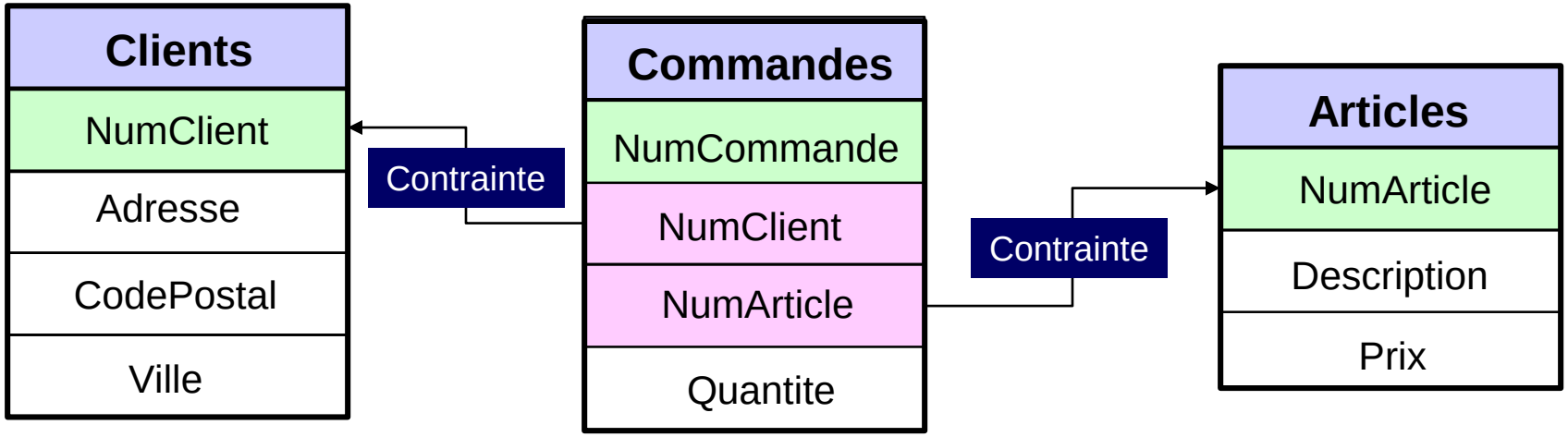
Intégrité des données – Définition de contraintes

- Une contrainte est une règle contrôlée par DB2 pour limiter les valeurs des données qui peuvent être insérées, modifiées ou supprimées dans une table

- Il existe 4 types de contraintes :
  - D'unicité (de clé unique)
    - Les valeurs de clé d'une table sont valides uniquement si elles sont uniques
    - Une ou plusieurs contraintes d'unicité 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 cohérentes entre 2 tables
  - De vérification
    - Limite les valeurs autorisées dans une ou plusieurs colonnes d'une table

Intégrité des données – Définition de contraintes

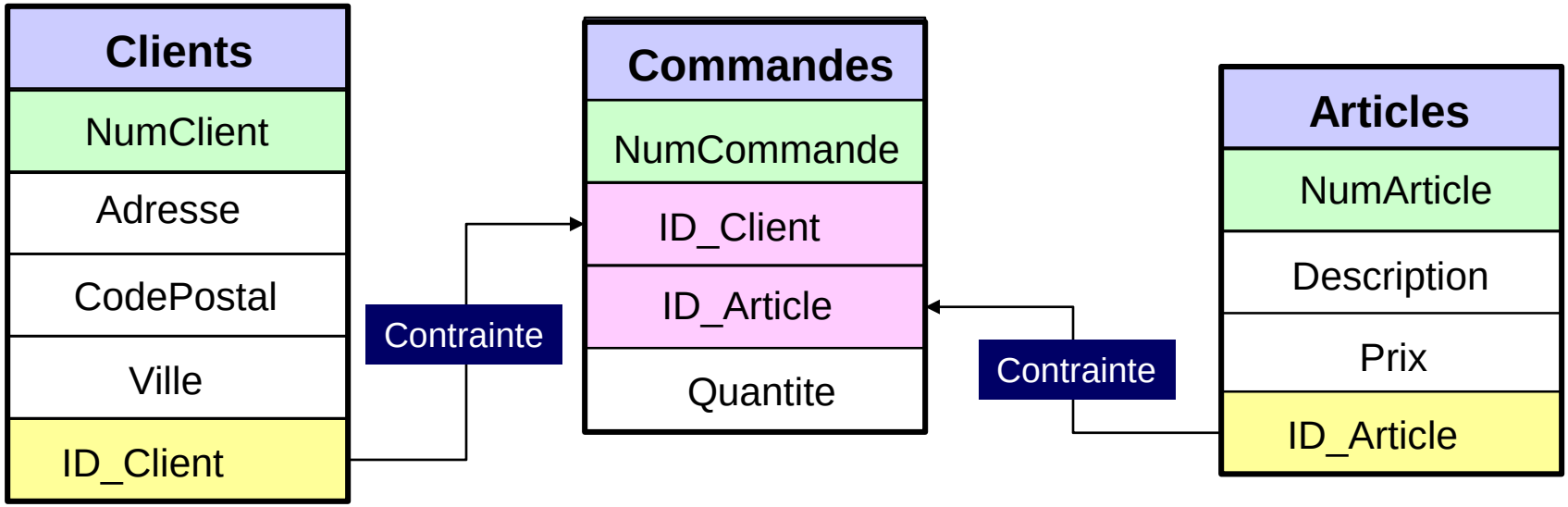
- **Cas n°1** – Mise en place de contraintes d'intégrité référentielle sur des colonnes existantes



Contrainte	Règle
PRIMARY KEY	Identifie la clé parente
FOREIGN KEY	Règle implicite : intégrité avec la clé parente Règles explicites : définition des actions en cas d'ajout, mise à jour et suppression dans la table parente

Intégrité des données – Définition de contraintes

- **Cas n°2** – Mise en place de contraintes d'intégrité référentielle par l'ajout de nouvelles colonnes



Contrainte	Règle
UNIQUE	Identifie la clé unique
PRIMARY KEY	Identifie la clé parente
FOREIGN KEY	Règle implicite : intégrité avec la clé parente Règles explicites : définition des actions en cas de mise à jour et suppression dans la table parente

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
  - Clauses
    - UNIQUE
    - PRIMARY KEY
    - FOREIGN KEY
    - CHECK
- On peut les visualiser/gérer par
  - IBM Navigator for i
  - System i Navigator
- Elles peuvent également être définies en CL
  - ADDPFCST, CHGPFCST, RMVPFCST, WRKPFCST

Intégrité des données – Définition de contraintes

```
47 ALTER TABLE employes FOREIGN KEY (numsrv) REFERENCES services (numsrv) ON DELETE RESTRICT;
48
49 INSERT INTO employes VALUES(128, 'LAROUSSE', 'D51');
50
```

```
[ Wed Jan 18 14:23:05 CET 2017 ]  Run Selected...
INSERT INTO employes VALUES(128, 'LAROUSSE', 'D51')
SQL State: 23503
Vendor Code: -530
Message: [SQL0530] Opération non admise par la contrainte référentielle Q_PBSQL_EMPLOYES_NUMSRV_00001
de PBSQL. Cause . . . . . : Dans le cas d'une instruction INSERT, UPDATE ou MERGE, la valeur n'est pas admise
pour la clé étrangère car il n'y a pas de valeur correspondante dans la clé parente. Dans le cas d'une instruction
DELETE ou MERGE affectée par une règle de suppression SET DEFAULT, la valeur par défaut est incorrecte pour la
même raison. Dans le cas d'une instruction ALTER TABLE, le résultat de l'opération serait incompatible avec la
contrainte Q_PBSQL_EMPLOYES_NUMSRV_00001. La contrainte Q_PBSQL_EMPLOYES_NUMSRV_00001 de
PBSQL pour la table EMPLOYES de PBSQL suppose que toute valeur définie de la clé associée ait une valeur
correspondante dans la clé parente. Que faire . . . : Pour être conforme à la règle de contraintes, effectuez l'une
des opérations suivantes : - Prenez pour INSERT, UPDATE ou MERGE une valeur existant dans la clé parente. -
Insérez dans le fichier parent une ligne qui corresponde aux valeurs de clé associée en cours d'insertion ou de
mise à jour. - Insérez dans le fichier parent une ligne qui corresponde aux valeurs par défaut de clé associée des
lignes dépendantes. Sinon, supprimez la contrainte référentielle.
```

Règles en cas de suppression dans la table parente (ON DELETE)

- RESTRICT / NOACTION : pas de suppression si clé étrangère associée
- 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 / NOACTION : pas de mise à jour si clé étrangère associée

Colonnes auto-incrémentées et auto-renseignées

- Colonnes auto-incrémentées
  - Dans la table : clause AS IDENTITY sur la colonne
  - En dehors de la table : objets de type SEQUENCE

- Colonnes auto-renseignées
  - Stockage automatique par DB2 de :
    - L'horodate de dernière mise à jour de l'enregistrement (7.1)
      - Clause FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
    - Le type d'opération (I, U, D) (7.3)
      - Clause GENERATED AS DATA CHANGE OPERATION
    - Des registres spéciaux (7.3)
      - Nom de l'utilisateur, nom du serveur… : clause GENERATED AS USER, AS CURRENT SERVER…
    - Des variables globales (7.3)
      - Le nom du job, l'adresse IP…: clause GENERATED AS QSYS2.JOB_NAME, AS SYSIBM.CLIENT_IPADDR…

Amélioration des objets SQL après conversion

- Exemple

Nom long

Nom court

Colonne auto-incrémentée

```
CREATE OR REPLACE TABLE employes (
  matriculeEmploye FOR matemp NUMERIC(6,0) AS IDENTITY (START WITH 128) PRIMARY KEY,
  nomEmploye FOR nomemp CHAR(25) NOT NULL,
  numeroService FOR numsrv CHAR(3),
  dateModification TIMESTAMP NOT NULL IMPLICITLY HIDDEN
                    FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP,
  utilisateur VARCHAR(128) GENERATED AS (SESSION_USER))
RCDFMT fmtemp;
```

Zone cachée

Stockage du nom de l'utilisateur

Stockage de l'horodate de dernière modification

MATRICULEEMPLOYE	NOMEMPLOYE	NUMEROSERVICE	UTILISATEUR	DATEMODIFICATION
115	DUPONT	D11	-	2017-01-18 12:40:54.029791
78	DURAND	D21	-	2017-01-18 12:40:54.029820
127	AZOULAY	D11	-	2017-01-18 12:40:54.029833
46	ROUX	D31	-	2017-01-18 12:40:54.029845
53	CIGNOT	D21	-	2017-01-18 12:40:54.029858
128	LAROUSSE	D31	BOURGEOIS	2017-01-18 12:58:12.713921

Tables temporelles (à partir de la version 7.3)

- Table temporelle = historisation automatique des données
  - Quels étaient nos vendeurs il y a 2 ans ?
  - Quel était l'état de ma table articles au 1er trimestre 2014 ?
  - Je voudrais reproduire l'inventaire comme si nous étions le 10 mars 17h00
- DB2 garde automatiquement l'historique des données
- L'interrogation de l'historique se fait par SQL directement sur la table de production :
  - SELECT nom_vendeur FROM commandes **FOR SYSTEM_TIME** **AS OF** CURRENT TIMESTAMP – 2 YEARS
  - SELECT * FROM articles **FOR SYSTEM_TIME** **BETWEEN** '2014-01-01-00.00.00' **AND** '2014-03-31-23.59.59'
  - **SET CURRENT TEMPORAL SYSTEM_TIME** '2017-03-10-17:00:00'
    - CALL rapport_inventaire()

Amélioration des objets SQL après conversion

- Exemple

```
CREATE OR REPLACE TABLE employes (
  matriculeEmploye FOR matemp NUMERIC(6,0) AS IDENTITY (START WITH 128) PRIMARY KEY,
  nomEmploye FOR nomemp CHAR(25) NOT NULL,
  numeroService FOR numsrv CHAR(3),
  dateModification TIMESTAMP NOT NULL IMPLICITLY HIDDEN
                             FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP,
  utilisateur VARCHAR(128) GENERATED AS (SESSION_USER),
  operation CHAR (1) GENERATED AS (DATA CHANGE OPERATION),
  debut TIMESTAMP(12) NOT NULL GENERATED AS ROW BEGIN,
  fin TIMESTAMP(12) NOT NULL GENERATED AS ROW END,
  periode TIMESTAMP(12) GENERATED AS TRANSACTION START ID,
  PERIOD SYSTEM_TIME(debut, fin))
RCDFMT fmtemp;

CREATE OR REPLACE TABLE emplhisto LIKE employes;

ALTER TABLE employes ADD VERSIONING USE HISTORY TABLE emplhisto ON DELETE ADD EXTRA ROW;
```

MATRICULEEMPLOYE	NOMEMPLOYE	NUMEROSERVICE	UTILISATEUR	OPERATION	DEBUT	FIN	PERIODE	DATEMODIFICATIO
115	DUPONT ...	D11	-	-	0001-01-01 00:00:00.000000000000	9999-12-30 00:00:00.000000000000	-	2017-01-18 14:32:2
78	DURAND ...	D21	-	-	0001-01-01 00:00:00.000000000000	9999-12-30 00:00:00.000000000000	-	2017-01-18 14:32:2
127	AZOULAY ...	D11	-	-	0001-01-01 00:00:00.000000000000	9999-12-30 00:00:00.000000000000	-	2017-01-18 14:32:2
46	ROUX ...	D31	-	-	0001-01-01 00:00:00.000000000000	9999-12-30 00:00:00.000000000000	-	2017-01-18 14:32:2
53	CIGNOT ...	D21	-	-	0001-01-01 00:00:00.000000000000	9999-12-30 00:00:00.000000000000	-	2017-01-18 14:32:2
128	LAROUSSE ...	D31	BOURGEOIS	-	0001-01-01 00:00:00.000000000000	9999-12-30 00:00:00.000000000000	-	2017-01-18 14:32:4

Sécurité niveau ligne et colonne

- **RCAC** (Row and Column Access Control)
- A partir de la 7.2
- Permet de limiter l'accès à
  - Certaines colonnes
    - En créant un masque : CREATE MASK
  - Certaines lignes
    - En créant des permissions : CREATE PERMISSION
- S'applique quelque soit l'interface d'accès à la table
- Ne nécessite pas la modification des applications
- Personne n'y échappe

Sécurité niveau ligne et colonne

- 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 'Non autorisé'
  END
  ENABLE;

ALTER TABLE EMPLOYES ACTIVATE COLUMN ACCESS CONTROL;
```

- La fonction VERIFY_GROUP_FOR_USER permet de vérifier qu'un utilisateur est bien membre d'un profil de groupe

Sécurité niveau ligne et colonne

- 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;
```

SQL – Pour en savoir plus

- Workshop "DB2 for i – SQL avancé" – 3 jours
  - DDL avancé, DML avancé, programmation SQL
  - Dans vos locaux
  - Prix à la journée quel que soit le nombre de personnes

- Mise-à-jour des connaissances DB2 for i
  - Nouveautés 6.1, 7.1, 7.2, 7.3 (à la carte)
  - Dans vos locaux
  - Prix à la journée quel que soit le nombre de personnes

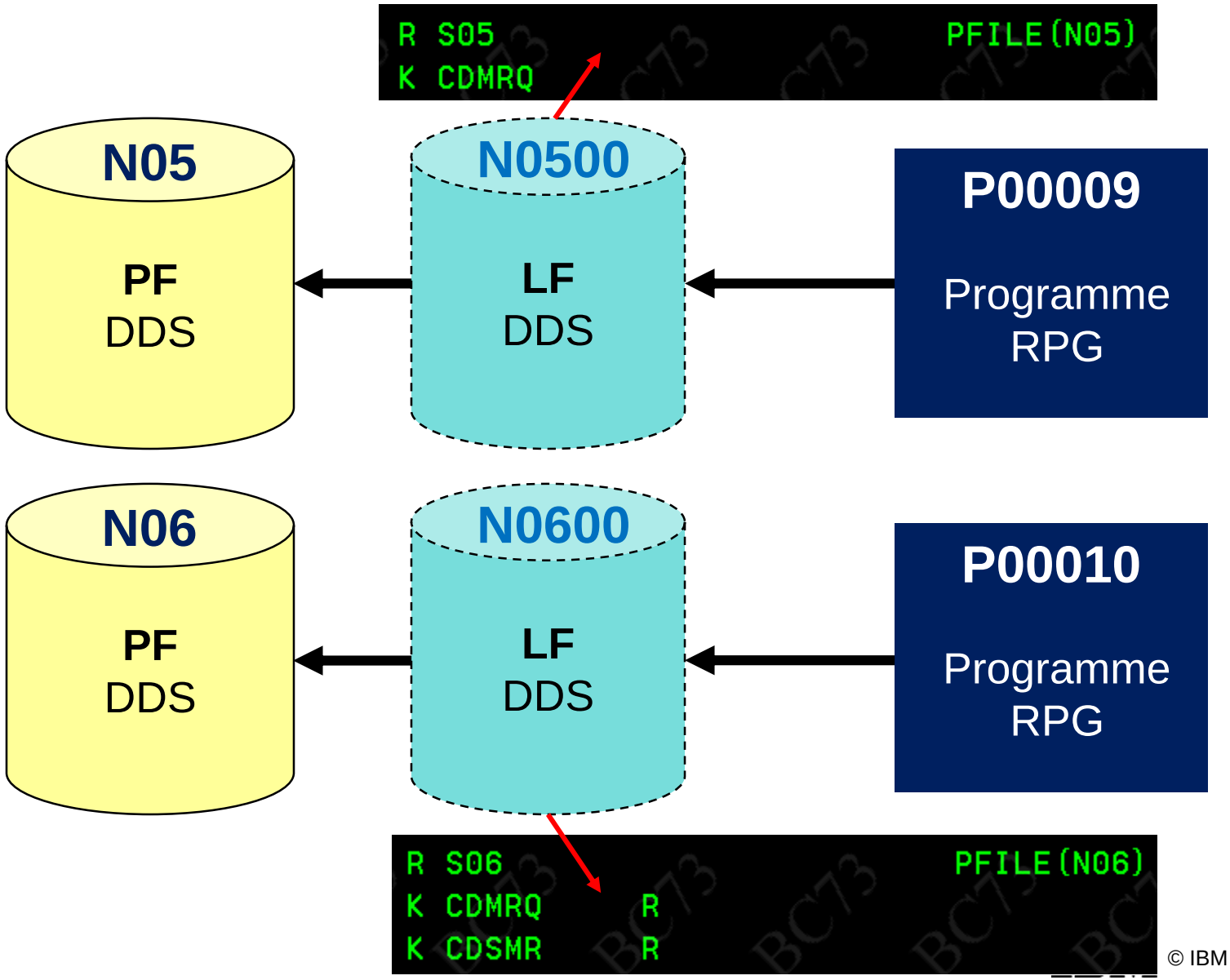
- Me contacter : pbourgeois@fr.ibm.com

Les outils de migration automatique

- a. La solution **X-Case** for i
  - De Resolution Software, distribué en France par ITHEIS
- b. La solution **Transformer-DB**
  - D'ARCAD Software
- Avantages :
  - Automatisation du processus de migration
    - Réduction des risques d'erreur
    - Audit des problèmes potentiels de conversion
  - Découverte du modèle relationnel d'une base existante et documentation graphique des relations
  - Audit de la faisabilité de la mise en place de l'intégrité référentielle et implémentation active / non active

5. Démonstration

Contexte AVANT modernisation



Contexte APRES modernisation