IBM

Db2 for i
Advanced SQL DDL

Rob Bestgen
Db2 for i Consultant
bestgen@us.ibm.com

Why SQL Data Definition Language (DDL)?

- Data-Centric programming
  - Let the Database do more for you!
- Take advantage of the latest Db2 technology
- Drive work into the database and lessen work for the application
  - Improve consistency and efficiency
- Leverage new tools technology
- Open up new ways to access data
  - PHP, JDBC, ODBC, .NET, CLI

DDL related things you can do only with SQL

- Long names for files, fields, even libraries
- Write time data validation
- LOB columns
- Identity columns
- Row changed timestamp
- NCHAR
- Database managed (audit) columns with GENERATED ALWAYS
- Check constraints
- Row and column level security (RCAC)
- Temporal - History based data
- XML columns
- JSON store

SQL Table Practices

CREATE TABLE (DDL) vs CRTPF (DDS)

```
CREATE TABLE EMP_MAST (
    EMP_MAST_PK FOR COLUMN EM_PK
    BIGINT GENERATED BY DEFAULT AS IDENTITY  IMPLICITLY
    HIDDEN PRIMARY KEY,
    EMPNO CHAR(6) UNIQUE,
    FIRSTNME VARCHAR(12),
    MIDINIT CHAR(1),
    LASTNAME VARCHAR(15),
    EMP_PICTURE BLOB(102400) ,
    EMP_ROWID ROWID GENERATED ALWAYS /* illustration only*/,
    EM_ROW_CHANGE_TS FOR COLUMN EMROWCHGTS
    TIMESTAMP NOT NULL FOR EACH ROW ON UPDATE AS ROW
    CHANGE TIMESTAMP IMPLICITLY HIDDEN)
```

```
CRTPF FILE(EMPLOYEE) SRCFILE(QDDSSRC)
    SRCMBR(EMPLOYEE)
ADDPFM FILE(QDDSSRC) MBR(EMPLOYEE)
--Source Data
    A                          UNIQUE
    A          R EMPLOYEE
    A            EMPNO          6
    A            FIRSTNME      12   VARLEN
    A            MIDINIT        1
    A            LASTNAME      15  VARLEN
    A          K EMPNO
ADDPFCST FILE(EMPLOYEE) TYPE(*PRIKEY) KEY(EMPNO)
```

Many new data types and functions

Long names

Multiple constraints defined within statement

Self contained source statement

- store as IBM i source member or PC file

No new data types

Only 1 key per definition. Constraints must be manually added

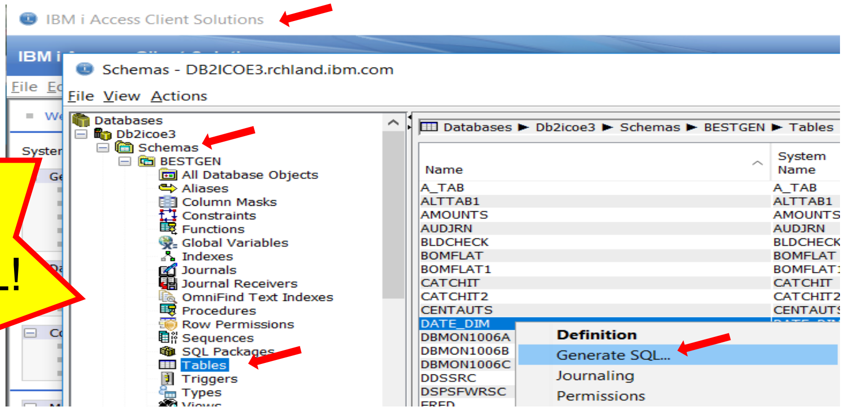
Requires separate source member

Source member must exist on IBM i to be compiled

SQL - Identify and Exploit DDL Enhancements

- Adding new columns takes advantage of data centric capabilities
  - Auto-generation fields
    - Identity Columns (Primary key)
    - Row change TIMESTAMP (optimistic locking, LCFO)
    - Sequence objects (Unique keys)
  - Large Object (LOB) Columns
  - …
- Numerous additional table options
  - NOT LOGGED, VOLATILE, LIKE, RCAC, temporal, partition tables, field procedures, …
- Future enhancements

Start with Generate SQL!



Name	System Name
A_TAB	A_TAB
ALTTAB1	ALTTAB1
AMOUNTS	AMOUNTS
AUDJRN	AUDJRN
BLDCHECK	BLDCHECK
BOMFLAT	BOMFLAT
BOMFLAT1	BOMFLAT1
CATCHIT	CATCHIT
CATCHIT2	CATCHIT2
CENTAUTS	CENTAUTS
DATE_DIM	DATE_DIM
DBMON1006A	
DBMON1006B	
DBMON1006C	
DDSSRC	
DSPSFWRSC	
FRED	

Why SQL - Identity Column

- Identity Column Attribute
  - Attribute that can be added to any “whole” numeric column
  - Not guaranteed to be unique - primary key or unique index must be defined
  - Only available for SQL tables, BUT identity column value generated for non-SQL interfaces

```
CREATE TABLE employee( empno INTEGER GENERATED ALWAYS AS IDENTITY
                                                        (START WITH 10 , INCREMENT BY 10),
                             name CHAR(30),   dept# CHAR(4),
                              PRIMARY KEY(empno))
```

```
INSERT INTO employee(name,dept#) VALUES('MIKE','503A') or…
INSERT INTO employee VALUES(DEFAULT,'MIKE', '503A')
```

XML Data Type

- XML data type
  - Supports XML documents up to 2 GB
  - Type can be used for column, parameter, and host variable values

```
CREATE TABLE  Reservations
(  res_ID            INTEGER
                     GENERATED ALWAYS AS IDENTITY,
   res_Doc           XML,
   res_TimeStamp TIMESTAMP
                     NOT NULL
                     IMPLICITLY HIDDEN
                     FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
 )
```

ID	XML	Timestamp

Large Object (LOB) Data Types

- CLOB – up to 2GB of text
- BLOB – binary object
- DBCLOB – double byte and Unicode data

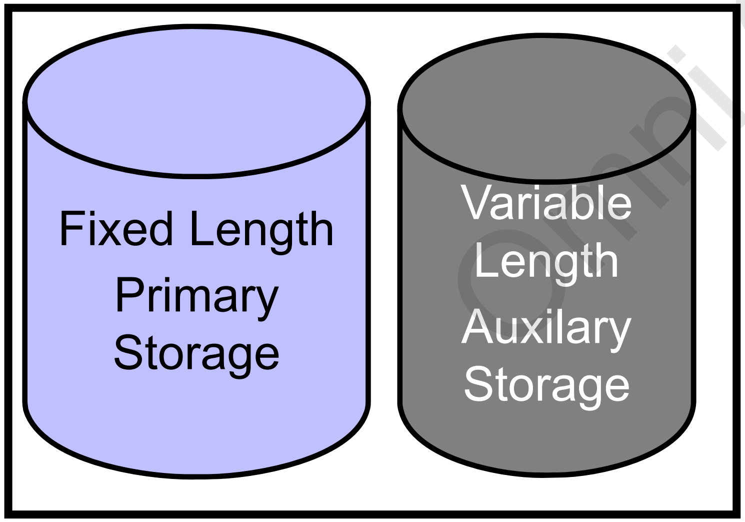
```
CREATE TABLE  Recruit
(  Id            INTEGER GENERATED ALWAYS AS IDENTITY,
   Name          VARCHAR(128),
   Resume        BLOB(2M),
   Picture       BLOB(10M),
   Received      TIMESTAMP NOT NULL
                 FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP)
```

Can populate blob using:

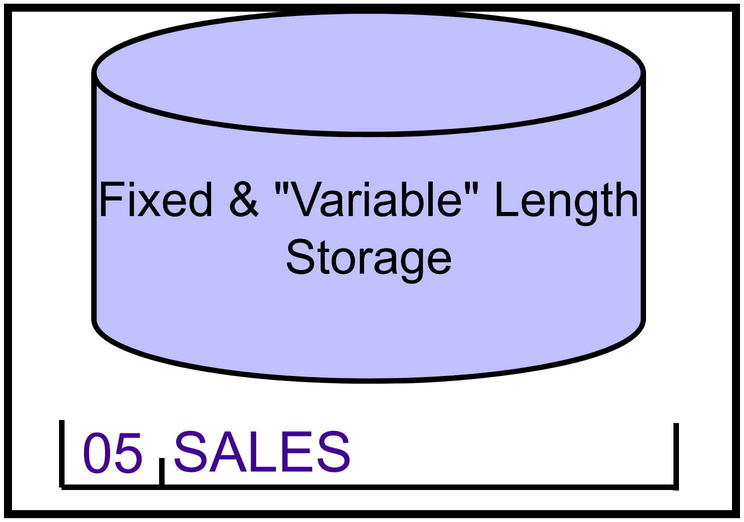
- SQL TYPE IS BLOB_FILE or
- GET_BLOB_FROM_FILE

VARCHAR considerations

```
CREATE TABLE dept
(
  id      CHAR(4),
  name VARCHAR(40),
  bldg_num INTEGER
 )
```



```
CREATE TABLE dept
(
  id      CHAR(4),
  name VARCHAR(40)
            ALLOCATE(20),
  bldg_num INTEGER
 )
```



CREATE TABLE (& SQL) Naming Considerations

- SQL Column & Object names have maximum lengths of 128
  - but the system only supports a 10-character length. How does that work?!
  - System automatically generates a short 10 character name
    - First 5 chars with unique 5 digit number

      CUSTOMER_MASTER >> CUSTO00001

- Short name might be different each time a table is created
  - depending on creation order and other objects

- Can use IBM i SQL syntax to specify short name
  - FOR SYSTEM NAME for tables, views, and indexes
  - FOR COLUMN clause for columns
  - FOR SCHEMA clause for libraries (schemas)
  - SPECIFIC clause for procedures, functions

But what about existing programs

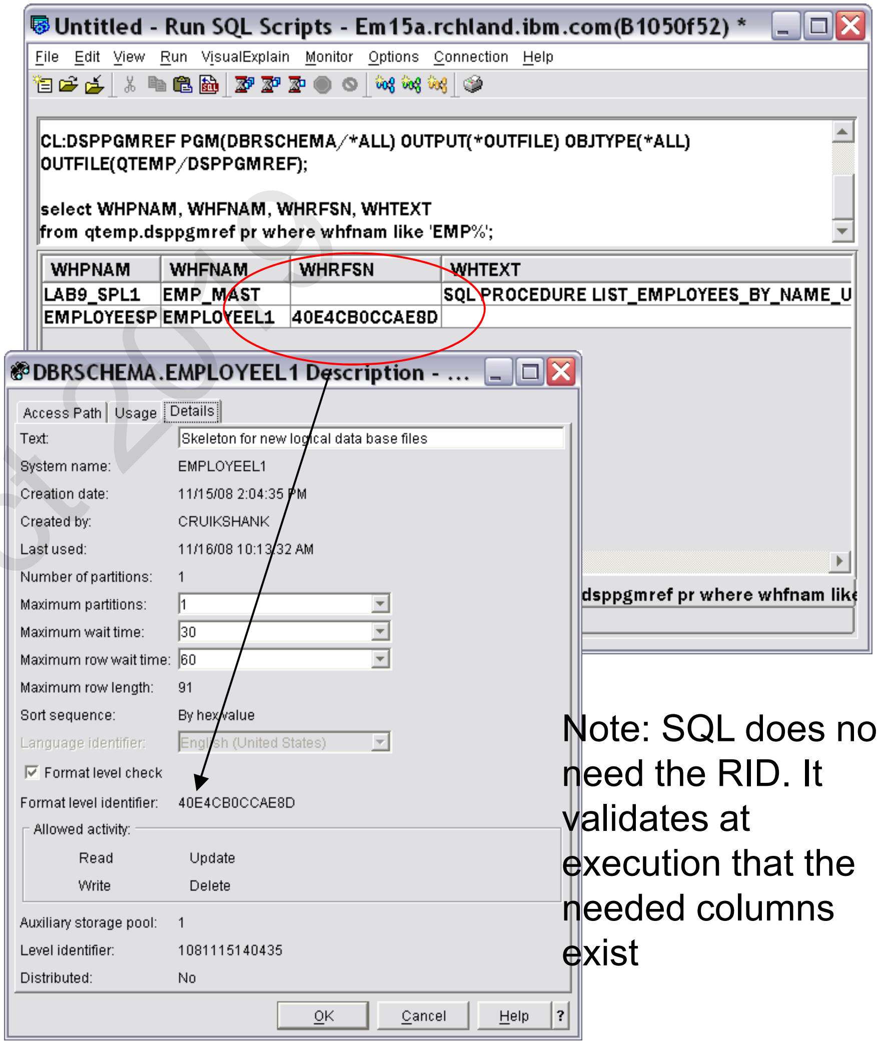
for all these new columns?

Change/recompile

OR

Beware the Format Level ID!

- A database file contains a: Record Format Level Identifier (RID)
  - The RID is captured in a program object when using Record Level Access (native)
  - Note: SQL does not care about RID
- The RID establishes integrity between the file and programs using native access
  - When the RID changes (i.e. column added or dropped) the program will break unless:
    - The program is created with Level Check = *NO (Not recommended)
    - The program is recreated



Untitled - Run SQL Scripts - Em15a.rchland.ibm.com(B1050f52) *
File Edit View Run VisualExplain Monitor Options Connection Help

```
CL:DSPPGMREF PGM(DBRSCHEMA/*ALL) OUTPUT(*OUTFILE) OBJTYPE(*ALL)
OUTFILE(QTEMP/DSPPGMREF);

select WHPNAM, WHFNAM, WHRFSN, WHTEXT
from qtemp.dsppgmref pr where whfnam like 'EMP%';
```

WHPNAM	WHFNAM	WHRFSN	WHTEXT
LAB9_SPL1	EMP_MAST		SQL PROCEDURE LIST_EMPLOYEES_BY_NAME_U
EMPLOYEESP	EMPLOYEEL1	40E4CB0CCAE8D	

dsppgmref pr where whfnam like
DBRSCHEMA.EMPLOYEEL1 Description - ...
Access Path
Usage
Details

Text:	Skeleton for new logical data base files
System name:	EMPLOYEEL1
Creation date:	11/15/08 2:04:35 PM
Created by:	CRUIKSHANK
Last used:	11/16/08 10:13:32 AM
Number of partitions:	1
Maximum partitions:	1
Maximum wait time:	30
Maximum row wait time:	60
Maximum row length:	91
Sort sequence:	By hex value
Language identifier:	English (United States)
☑ Format level check	
Format level identifier:	40E4CB0CCAE8D
Allowed activity:	Read, Update, Write, Delete
Auxiliary storage pool:	1
Level identifier:	1081115140435
Distributed:	No

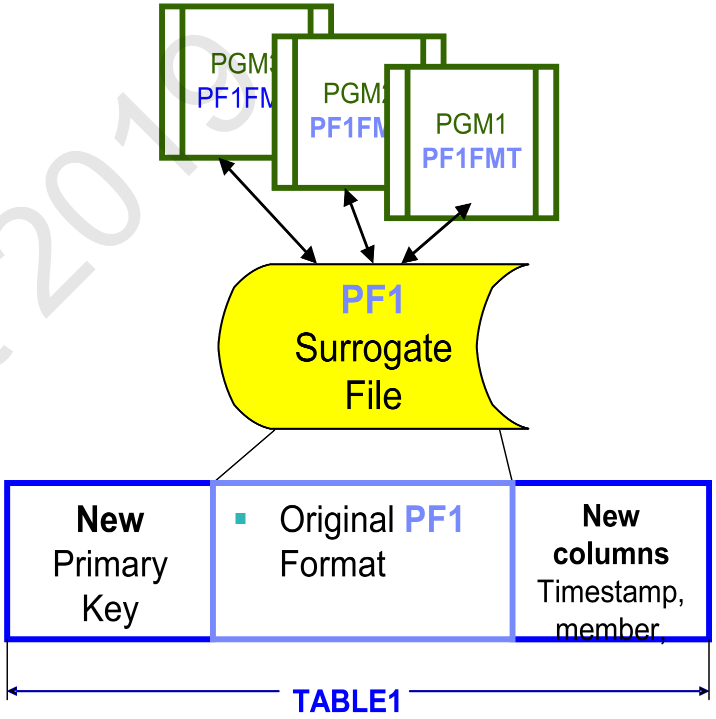
Note: SQL does not need the RID. It validates at execution that the needed columns exist

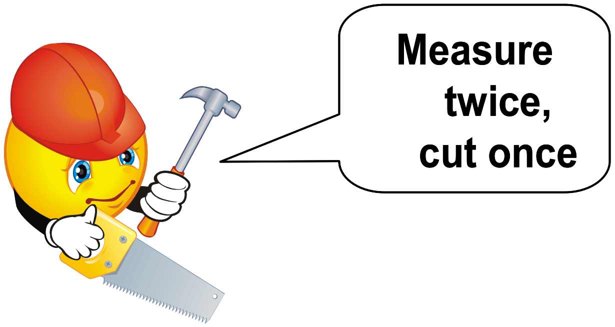
How can we handle RID **AND** leverage new DDL support?

Adding New Columns to Re-engineered Table

- Surrogate LF methodology enables converted SQL table to be enhanced with new features…

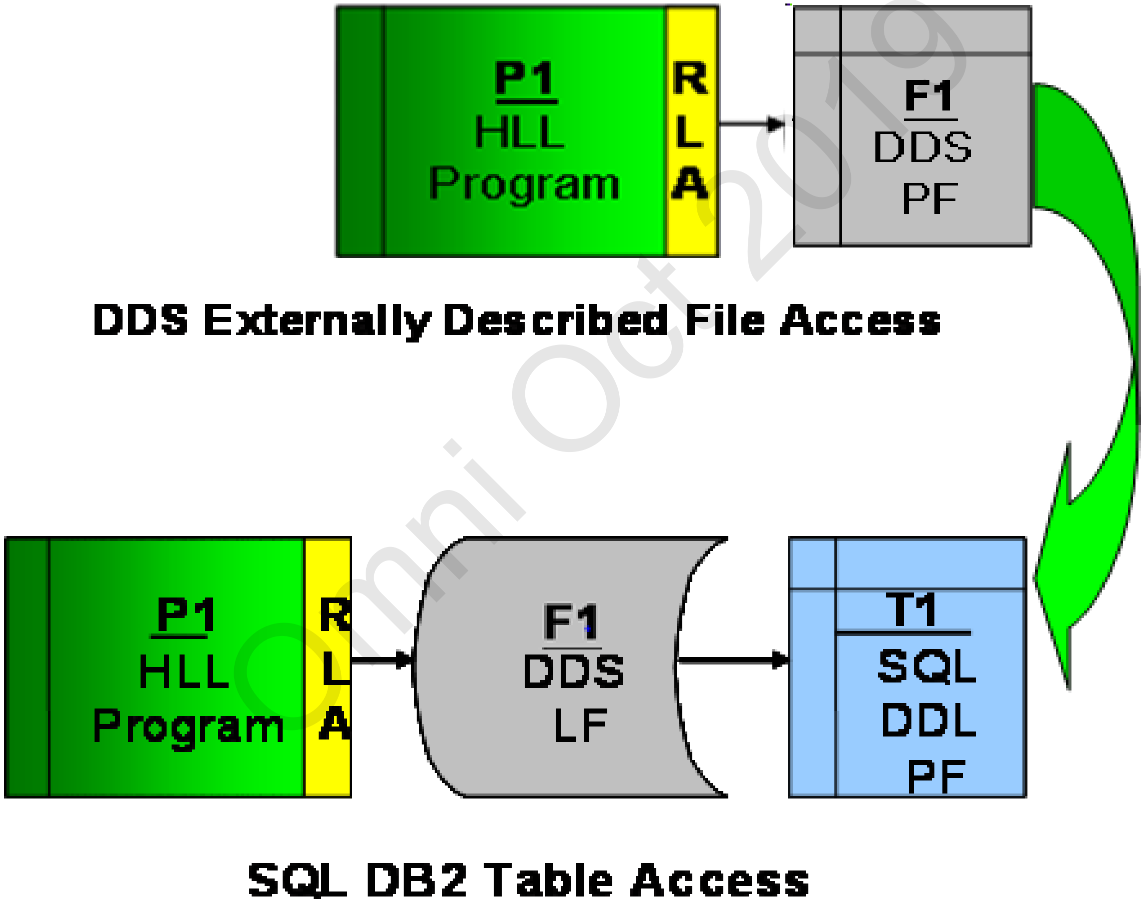
  WITHOUT changing ID of Surrogate!
  - New columns can be added before or after the original columns
    - Add Identity columns
    - Add Implicitly Hidden columns
  - Original column definitions can be altered



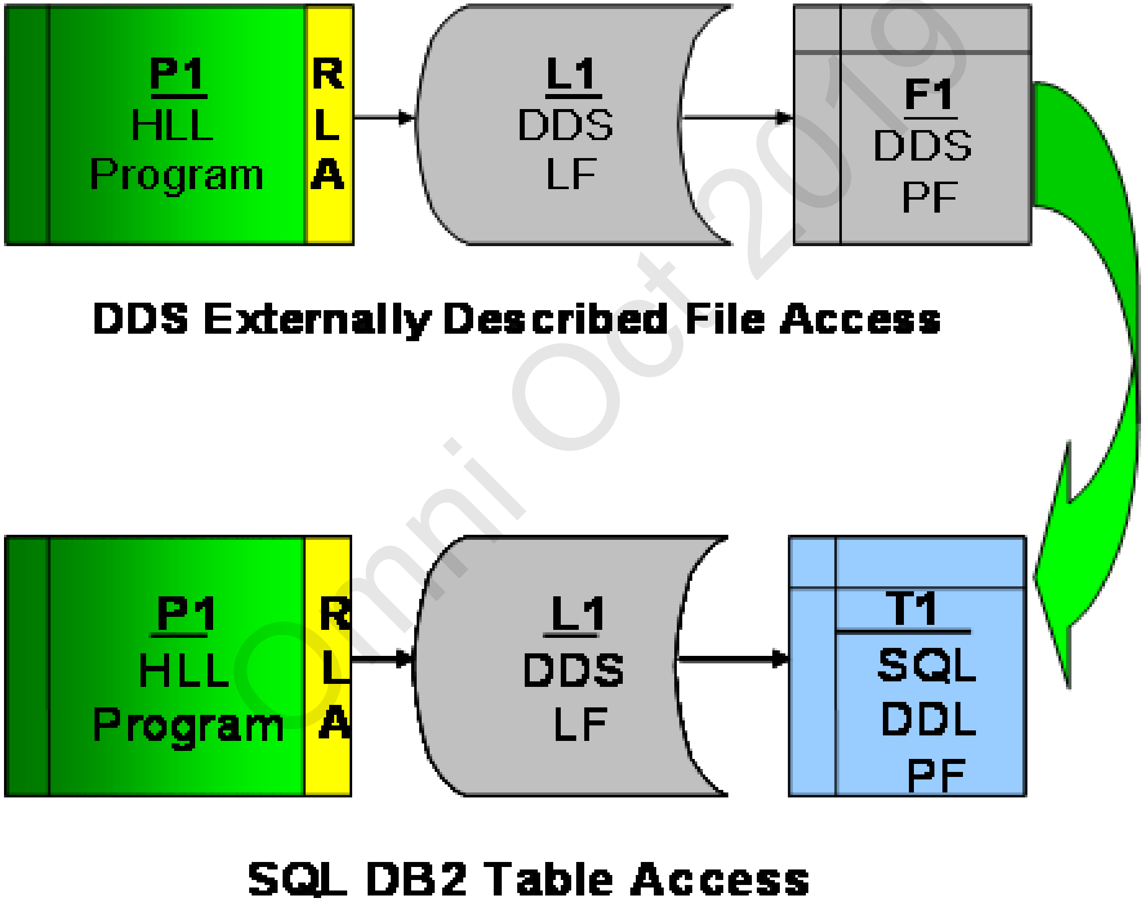


Transparent Migration to SQL – Surrogate and Logical files!

Shield (stable) existing pgms from the changes in the table



Transparent Migration to SQL – Options…



Transparent SQL Migration - Example

- Existing PF - INVENTORY

```
A R INVFMTR
A   ITEM   15A
A   ORDER  10A
A   SUPPLY 15A
A   QTY     5P
A   QTYDUE  5P
A K ITEM
```

- Existing LF - INVLF

```
A R INVFMTR PFILE(INVENTORY)
A K ORDER
A K ITEM
```

- Converted SQL Table –

```
CREATE TABLE sq_invent (
 item CHAR(15),
 order CHAR(10),
 supply CHAR(15),
 qty DECIMAL(5,0),
 qtydue DECIMAL (5,0))
```

- Surrogate LF - INVENTORY

```
A R INVFMTR PFILE(SQ_INVENT)
A   ITEM
A   ORDER
A   SUPPLY
A   QTY
A   QTYDUE
A K ITEM
```

- Modified Existing LF - INVLF

```
A R INVFMTR PFILE(SQ_INVENT)
A           FORMAT(INVENTORY)
A K ORDER
A K ITEM
```

Enhanced DDL TABLE and Surrogate DDS **LF**

```
CREATE TABLE CUST_MAST [1] (
   CUST_MAST_ID FOR COLUMN [2]
   CUSTMASTID BIGINT GENERATED BY
   DEFAULT AS IDENTITY PRIMARY KEY,
   CUSTKEY INTEGER NOT NULL UNIQUE [3],
   CUSTOMER CHAR(25)  NOT NULL   ,
   ADDRESS CHAR(40)  NOT NULL   ,
   CITY CHAR(30)  NOT NULL   ,
   STATE CHAR(2)  NOT NULL   ,
   ZIPCODE NUMERIC(10, 0) NOT NULL ,
   PHONE CHAR(15)  NOT NULL   ,
   CM_LAST_CHANGED FOR COLUMN
   CMLASTCHG TIMESTAMP  NOT NULL
             FOR EACH ROW ON UPDATE
             AS ROW CHANGE TIMESTAMP);
```

```
CRTLF CUSTMAST
A        R CUSTMASTR    PFILE(CUST_MAST [1])
A          CUSTKEY       R
A          CUSTOMER      R
A          ADDRESS       R
A          CITY          R
A          STATE         R
A          ZIPCODE       R
A          PHONE         R
A         K CUSTKEY [3]
```

Notes

1. **Original PF is now LF and references new SQL table CUST_MAST**
2. **New SQL only columns are not part of surrogate file**
3. **CUSTKEY is now unique key constraint (if appropriate)**

Reengineering Considerations

- Not all files need to be converted to SQL DDL – especially work files!!!

- Use Logical files to insulate Non-SQL access from underlying SQL table changes

- You should have good business reasons for migrating
  - New or changing requirements
  - Need for enhanced features and functions
  - New applications accessing legacy data

- Start small, get some experience
  - Identify a pilot application which would benefit from modernization
  - Get educated on SQL and Db2 for i

Indexes

Indexes

Indexes are used to **improve performance**

- Permanent object
- Not query-able from SQL
- Used proactively and reactively for improving performance
- Also used (under the covers) for constraint enforcement
- Come in two flavors for SQL
  - Regular (radix)
  - Encoded Vector Index (EVI)

CREATE INDEX vs CRTLF (Keyed)

```
CREATE INDEX EMP_LASTNAME_DEPT
  ON EMP_MAST(WORKDEPT, LASTNAME)
  ADD COLUMNS EMPNO,FIRSTNME,MIDINIT
```

Expressions can be used in the definition of the key columns (derived key index)

Sparse Indexes with WHERE clause
i.e. Select/Omit

- Use sparingly (preferably not at all!)

```
CRTLF FILE(EMPLOYEEL1)
   SRCFILE(QDDSSRC) SRCMBR(EMPLOYEEL1)
--Source Data
A          R EMPLOYEER1    PFILE(EMPLOYEE)
A            WORKDEPT
A            LASTNAME
A            EMPNO
A            FIRSTNME
A            MIDINIT
A          K WORKDEPT
A          K LASTNAME
```

Only Binary Radix Tree structure support – no EVIs

Limited support for key derivations and expressions

Smaller default logical page size

CREATE INDEX – Encoded Vector Index (EVI)

- EVI - complementary indexing technology for boosting performance in analytical query & reporting environments (OLAP)
  - Patented technology that advances traditional bitmapped indexing
  - Best fit – columns with low cardinality (type, color, state, etc…)

  Example: `CREATE ENCODED VECTOR INDEX idx1 ON sales(region)`

- INCLUDE Aggregate

```
CREATE ENCODED VECTOR INDEX idx1 ON sales(region)
      INCLUDE ( SUM(saleamt),  COUNT(*) )
```

```
CREATE ENCODED VECTOR INDEX idx2
      ON sales(territory)
      INCLUDE (SUM(saleamt + promoamt))
```

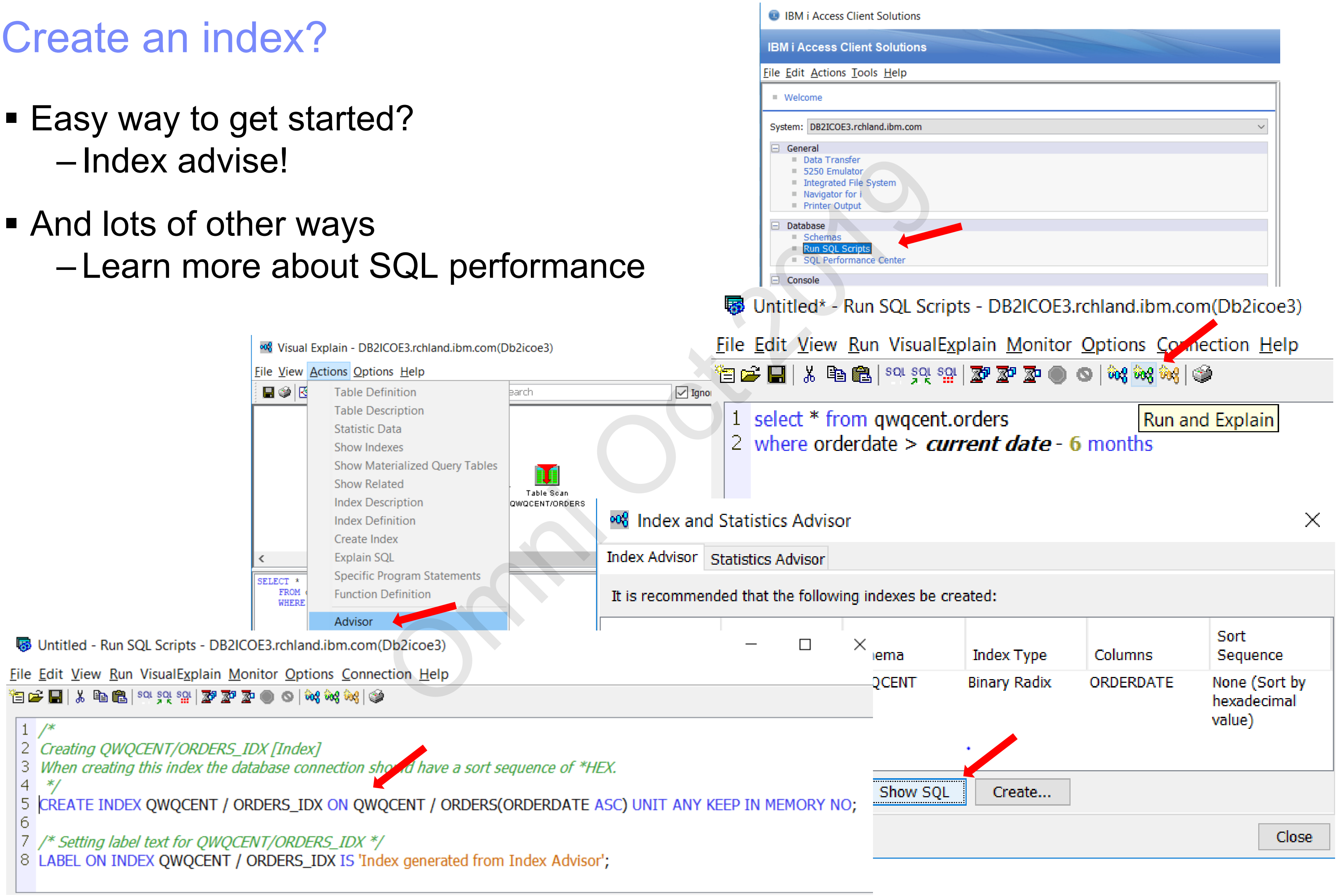
EVI aggregates maintained as underlying table changes

SELECT territory, SUM(saleamt+promoamt) FROM sales
 GROUP by territory

SELECT region, SUM(saleamt) FROM sales GROUP BY region

Create an index?

- Easy way to get started?
  - Index advise!
- And lots of other ways
  - Learn more about SQL performance


IBM i Access Client Solutions
IBM i Access Client Solutions
File Edit Actions Tools Help
Welcome
System: DB2ICOE3.rchland.ibm.com
General
Data Transfer
5250 Emulator
Integrated File System
Navigator for i
Printer Output
Database
Schemas
Run SQL Scripts
SQL Performance Center
Console
Untitled* - Run SQL Scripts - DB2ICOE3.rchland.ibm.com(Db2icoe3)
File Edit View Run VisualExplain Monitor Options Connection Help
Run and Explain
Visual Explain - DB2ICOE3.rchland.ibm.com(Db2icoe3)
File View Actions Options Help
Table Definition
Table Description
Statistic Data
Show Indexes
Show Materialized Query Tables
Show Related
Index Description
Index Definition
Create Index
Explain SQL
Specific Program Statements
Function Definition
Advisor
Table Scan
QWQCENT/ORDERS
Index and Statistics Advisor
Index Advisor
Statistics Advisor
Show SQL
Create...
Close
Untitled - Run SQL Scripts - DB2ICOE3.rchland.ibm.com(Db2icoe3)
File Edit View Run VisualExplain Monitor Options Connection Help

```
select * from qwqcent.orders
where orderdate > current date - 6 months
```

It is recommended that the following indexes be created:

ema	Index Type	Columns	Sort Sequence
QCENT	Binary Radix	ORDERDATE	None (Sort by hexadecimal value)

```
/*
Creating QWQCENT/ORDERS_IDX [Index]
When creating this index the database connection should have a sort sequence of *HEX.
 */
CREATE INDEX QWQCENT / ORDERS_IDX ON QWQCENT / ORDERS(ORDERDATE ASC) UNIT ANY KEEP IN MEMORY NO;

/* Setting label text for QWQCENT/ORDERS_IDX */
LABEL ON INDEX QWQCENT / ORDERS_IDX IS 'Index generated from Index Advisor';
```

Views and Alias

SQL Views

- Views provide a logical perspective of the data
- Permanent (file) object
- Use like you would a table (SELECT… FROM view…)
- Encapsulates (hides) complexity
  - Optimizer merges view definition with the query of the view at runtime
- Provide virtualization layer between application and physical table layout
  - Including improving readability
- Remove complexity from application
- Contains NO data!
- Can be used to create virtual columns
- Can use other SQL objects like global variables

View examples

- Simple view over a table

```
create or replace view orders as
 (select * FROM ordhdr where order_date > ‘2019-01-01’)
```

- Expand date information in a table

```
create or replace view orders as
  (select o.prdid as product_id,  o.quantity, o.linetotal as revenue, d.*
  from ordhdr o inner join date_conv d  on o.orderdate = d.dc_date)
```

- Completely virtual ‘table’, created on the fly

```
create or replace view year_of_dates as
  (with my_cte(d) as
   (select * from table(values(current date-1 year+1 day)) x
    union all
    select d + 1 day from my_cte
    where d < current date)
  select d as thedate, year(d) as theyear, month(d) as themonth, day(d) as theday
  from my_cte)
```

View examples…

- View with join and grouping

```
CREATE OR REPLACE VIEW
   EMPLOYEE_BONUSES_BY_DEPARTMENT_WITHIN_STATE AS
SELECT EA.STATE, DM.DEPTNAME, SUM(EM.BONUS) AS TOTAL_BONUS
FROM EMAST EM
  JOIN EADDR EA USING (EM_PK)
  JOIN DMAST DM ON WRKDPT = DPTNO
GROUP BY EA.STATE, DM.DEPTNAME
```

View perspectives

- Encapsulates (hides) complexity
  - But does not magically ‘fix’ performance issues!

- Contains NO data!
  - Data is processed when read

- No support for keying/ordering

Good practice: always access data through a view

(unless already part of logical separation layer – ETL or DAO)

Good practice #2: use **SQL** to access a view, not RLA ‘native’

ALIAS

Allows for simpler reference to database files

- Alias is itself a real object on the system
- Great way to reference a particular file member from SQL
- Hides other complexity like three part naming (remote system access)

```
CREATE OR REPLACE ALIAS CURMONTH FOR MAINLIB.SALES(MAY)
```

```
CREATE OR REPLACE ALIAS REMOTESALES FOR REMOTESYS.MAINLIB.SALES
```

Thank You!