



DB2 for z/OS DBaaS and the Advantages of Automated Objects

Daniel L Luksetich

DanL Database Consulting

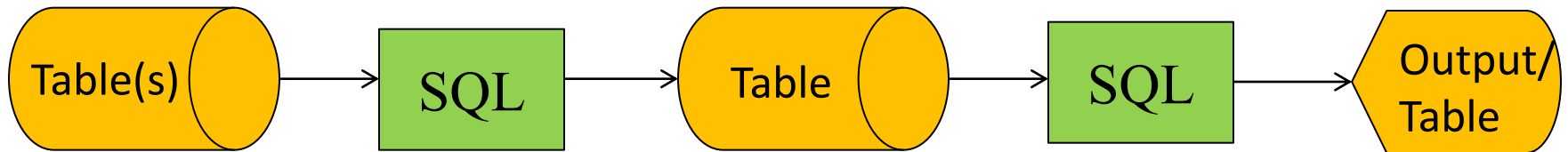
danl@db2expert.com

- Databases evolved out of the necessity to improve on information
 - Eliminate storage redundancy
 - Separate the programmer from the hardware
 - Improve time to delivery for applications
 - Reduce data inconsistency
 - Automate data integrity validation
 - Improve security
 - Improve/create meta-data
- Once implemented and refined database evolved further
 - Increased security options
 - User-defined functionality extensions
 - Adaptation of new industry standards
 - Adaptation of new industry trends in programming

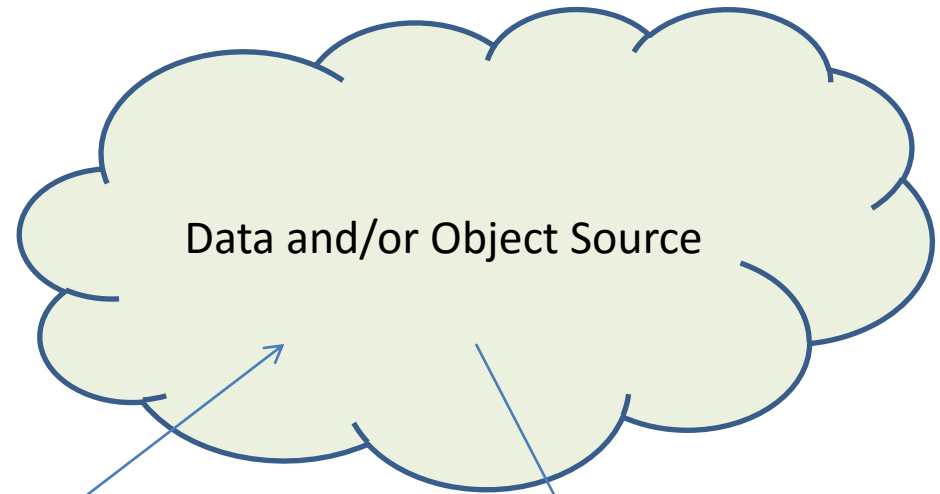
- Traditional DBA roles and responsibilities
 - Create and manage database objects
 - Tables and indexes
 - Storage groups
 - Security definitions
 - Manage and monitor database storage
 - Monitor and manage database performance
- Present and future DBA roles and responsibilities
 - Create and manage database objects
 - Tables and indexes
 - Security definitions (sometimes)
 - Data types
 - Database programming objects
 - Referential integrity
 - Types, functions, procedures, triggers, clones, materialized tables, versioning, etc.
 - Monitor and manage database performance

- Traditional developer roles and responsibilities
 - Write programs to manage inputs and outputs
 - Generate reports
 - Organize and develop complex systems
- Present and future developer roles and responsibilities
 - Ultra-fast deployment of new applications
 - Ultra-fast integration of applications
 - Flexible adaptable systems and technologies
 - Creation and management of technology abstraction
 - Application assembly
 - Object creation and management
 - Meta-data management (usually outside of database)
 - Common user interface
 - Web
 - Mobile

- Programs managed data
- Developers wrote programs that accessed data in tables
- SQL was a language used to access tables
- Application development, processing of data, happened within the application



- Programs manage content
- Developers assemble programs
- Frameworks used to access database objects
- Application development can happen anywhere
 - Objects
 - Database
 - Programs
 - Frameworks
 - Web services

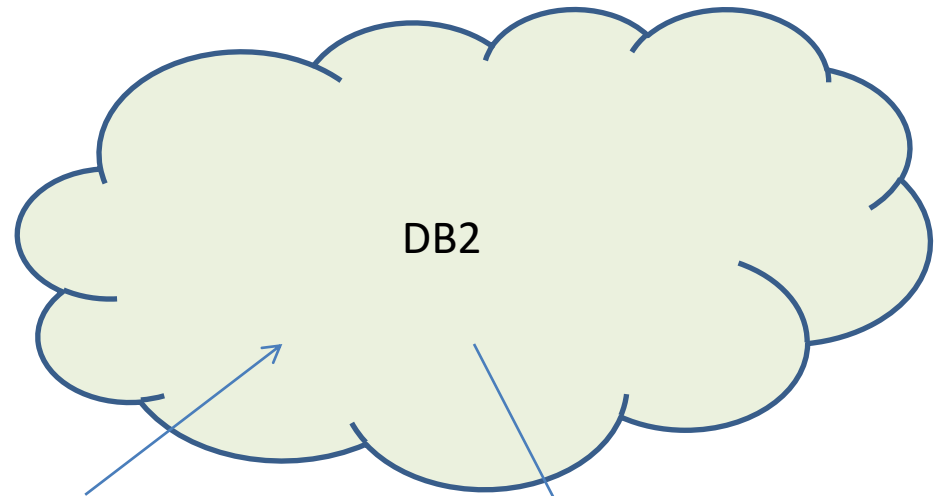


URI

```
http://10.25.14.17:259/service/db1/getcust?32537
```

```
{"customer":  
  {"ID": "32537",  
   "Name": "Bob Rady Enterprises"}  
}
```

- Stay relevant
- Stay adaptable to the changing application development world
- Use new and developing technologies
 - XML
 - JSON
 - Large objects
 - NoSQL
 - Procedures
 - Functions
 - DB2 RESTful API

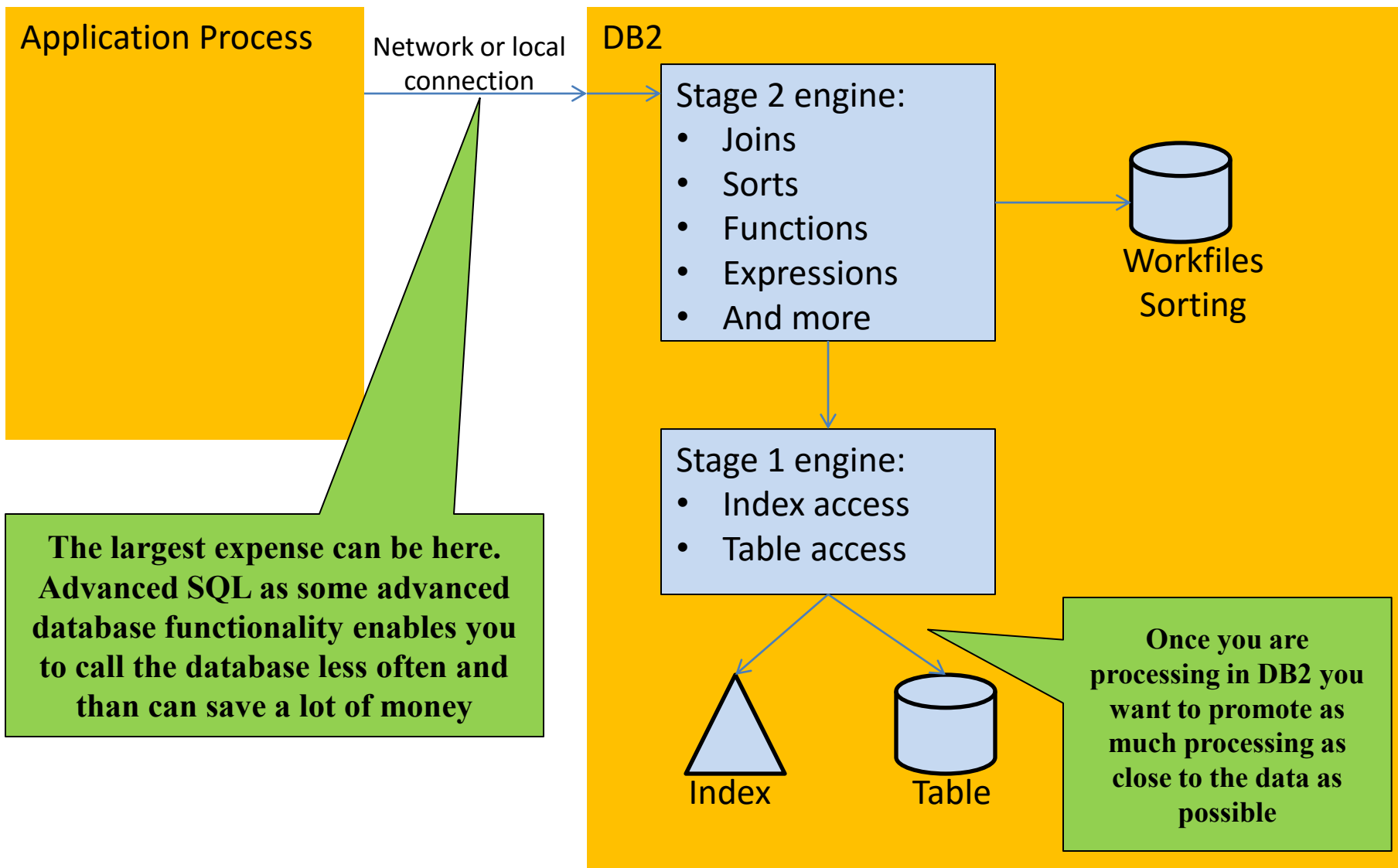


URI

```
http://10.25.14.17:259/service/db1/getcust?action=invoke
```

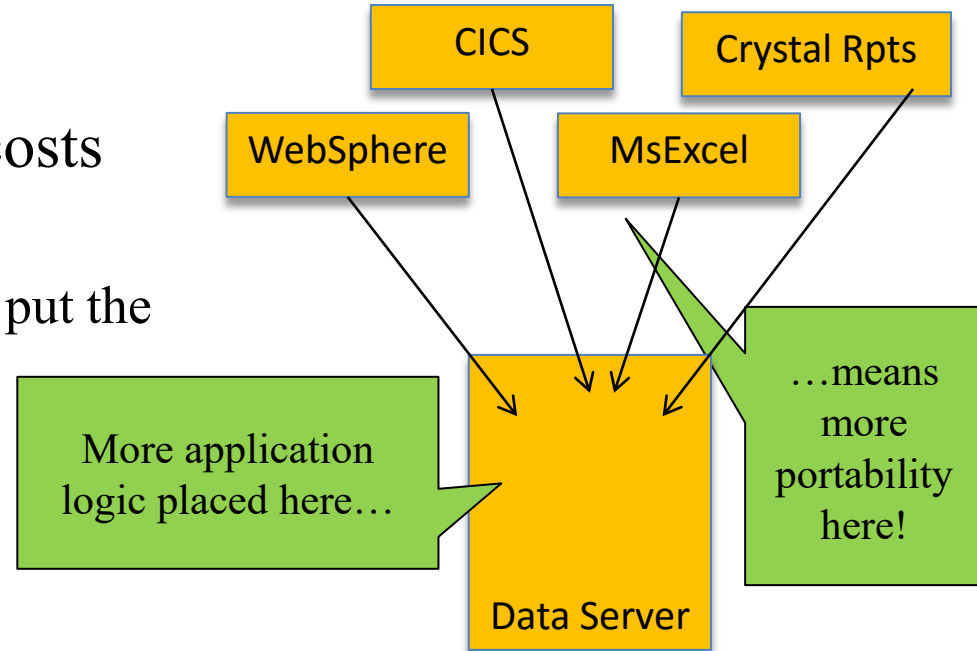
```
{"customer":  
  {"ID": "32537",  
   "Name": "Bob Rady Enterprises"}  
}
```

- Services can be built into DB2
 - Several ways this can be done
 - For automated services this can mean one action per call
- It can be helpful to integrate as much work into one call as possible
 - Make the service as fruitful as it can be
 - Do multiple actions inside DB2 to support a call
- DB2's automated features can be important here
 - Take multiple actions per call inside the database
 - Reduce the number of trips to the data server (DB2)
- DBAs and/or DB2 specialized developers become the DB2 object programmer



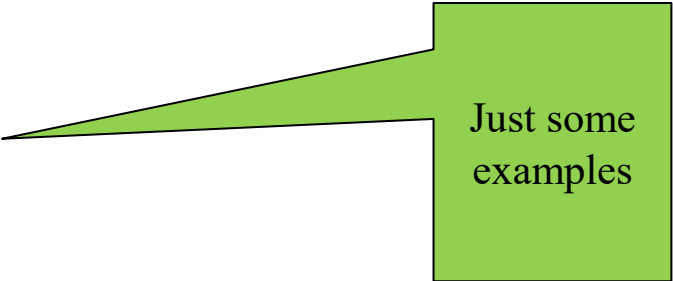
Why Program in the Database?!

- Fast application development time
- Centralization of application components
- Reusability of application components
- Improved application performance
 - In some cases
- Lower application operational costs
 - Sometimes
 - Sometimes it's more expensive to put the code in the database



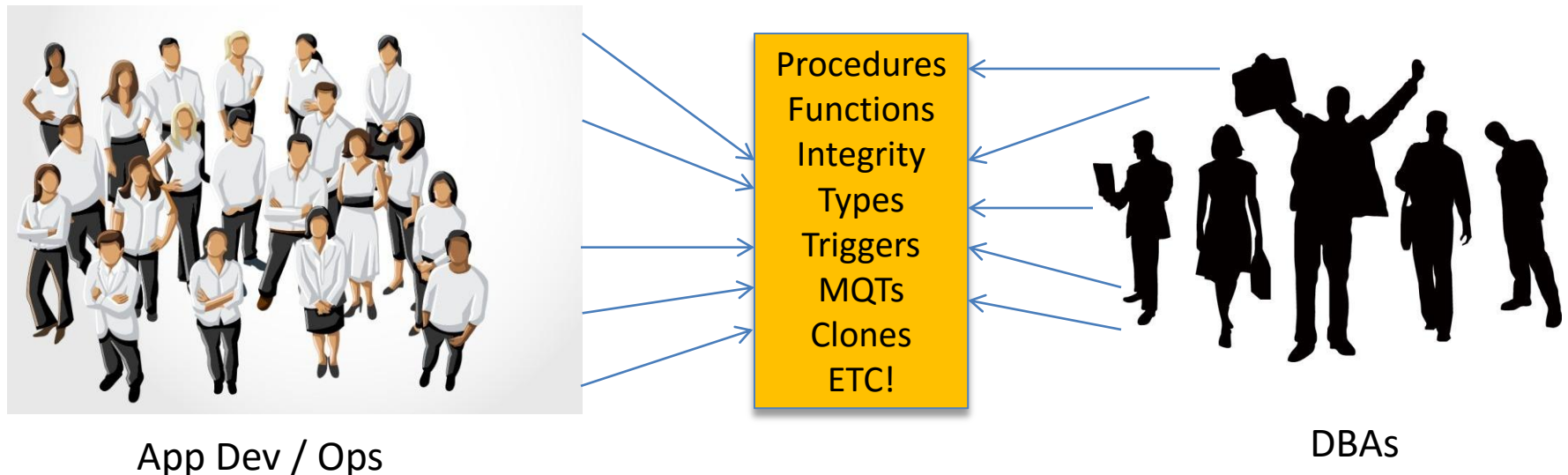
Complexity can be an Issue

- Database objects can contain programming logic
 - Application developers are responsible for programming logic
 - Database administrators are responsible for database objects
-
- Database objects can contain programming logic
 - Who is responsible?
 - Database objects in DB2 are not very easily traceable
 - Information in multiple places in system catalog
 - Logic in text (LOBs)
 - Information can be outside of application developer's realm
 - Information can be in language an application developer does not understand
 - Database application activity can be abstracted and unknown to anyone
 - Database enforced RI
 - Temporal and archive table data movement
 - Clone table existence and switching

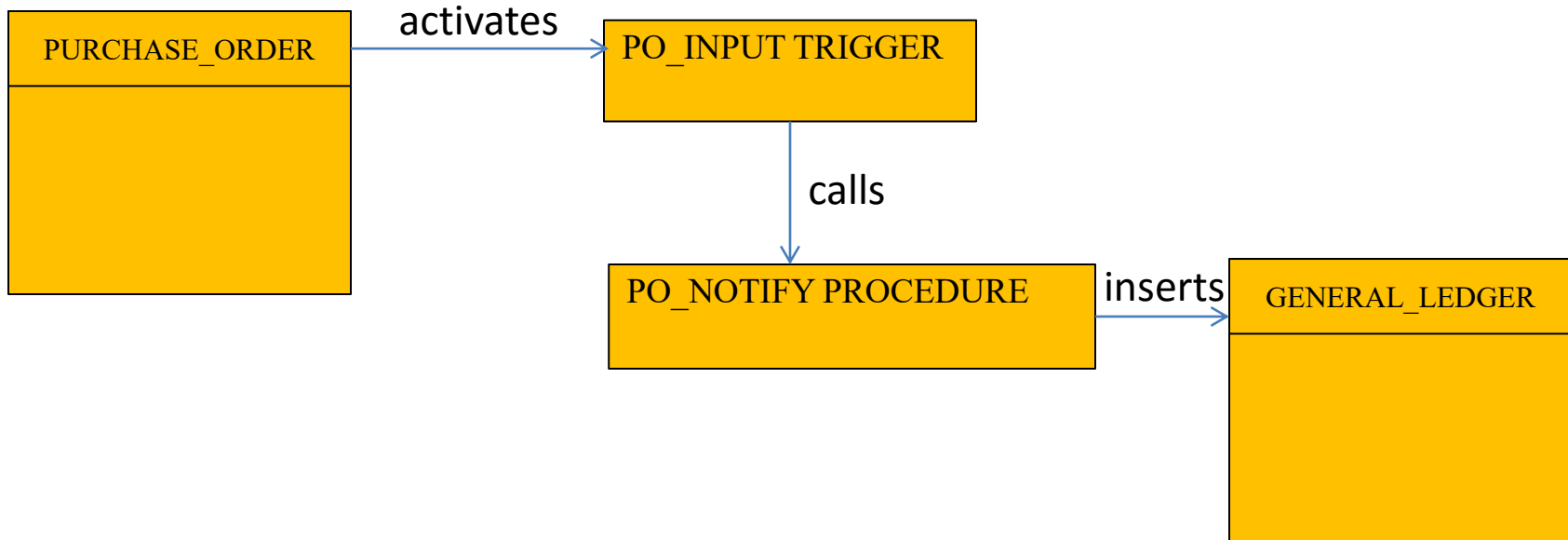


Just some examples

- The existence of program logic outside of the application executable requires additional responsibilities
- This spans DBA and developer areas of control
- Possible solutions
 - Create new development team – Procedural DBA Team
 - Expand role of the DBA team to include database application objects
 - Expand authority of development team to include database application objects



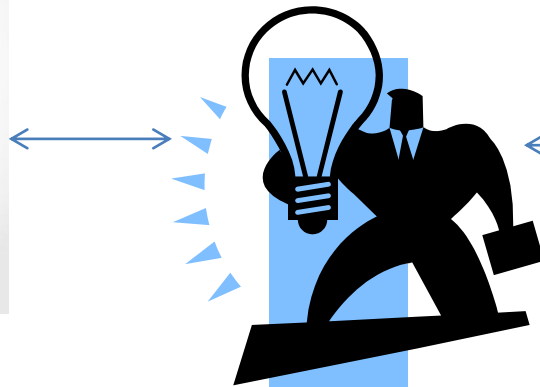
- The DB2 system catalog is NOT enough documentation
- Entity relationship diagrams are critical
- Database process models are needed
 - To avoid confusion, redundancy, and errors!



- The dedicated team that spans DBA and Developer is best for communication and control
- Some challenges
 - Keeping documentation / storyboards up to date
 - Coordination of changes
 - Coordination of testing
 - Performance monitoring and benchmarking



App Dev / Ops



DB Process Control
Specialist



DBAs

- Database enforced referential integrity
- Table check constraints
- Sequences and generated values
- Statement include columns
- Stored procedures and user-defined functions
- Triggers
- XML
- JSON

- There are many more but we are limited by time!
 - LOBS
 - Temporal and archive tables
 - Clone tables
 - Materialized query tables
 - And more!

- Data integrity is extremely important, especially in transaction environments
- DB2 can be programmed to automatically maintain data integrity
 - This includes the automatic enforcement of table relationships
- What are the benefits of this?
 - No additional application programming needed to maintain table relationships
 - Database enforcement means global or enterprise integrity enforcement
- Responsibility for integrity shifts
 - From application development/operation to database administration/operation
 - There is an abstraction of rules for the application development team
 - Relationship rules not in application programs
 - Relationship rules are stored in the database
 - Information about the rules in the DB2 system catalog

- **INSERT rule**
 - A nonnull insert value of the foreign key must match some value of the parent key of the parent table. The value of a composite foreign key is null if any component of the value is null.
- **UPDATE rule**
 - A nonnull update value of the foreign key must match some value of the parent key of the parent table. The value of a composite foreign key is treated as null if any component of the value is null.
 - An update of a parent key value can only happen if no matching foreign key values exist.
- **DELETE rule**
 - Controls what happens when a row of the parent table is deleted. The choices of action, made when the referential constraint is defined, are
 - RESTRICT, NO ACTION, CASCADE, or SET NULL.
 - SET NULL can be specified only if some column of the foreign key allows null values.

- **CASCADE**
 - Applicable to DELETE operations on parent tables
 - Child table row are deleted. That is, delete to a parent is propagated to all child tables for related foreign keys
- **RESTRICT**
 - Applicable to DELETE operations
 - Applicable to UPDATE operations (DB2 for LUW)
 - Delete or update to parent not allowed if child rows exist
- **NO ACTION**
 - Applicable to DELETE operations
 - Applicable to UPDATE operations (DB2 for LUW)
 - Delete or update to parent row not allowed if child rows exist
- **SET NULL**
 - Applicable to DELETE operations
 - If parent row deleted then optional child key values set to null

Difference between RESTRICT and NO ACTION is in the timing of the rule enforcement. NO ACTION is last to be enforced.

- Application enforcement of rules
 - No limits to rule complexity
 - All rule enforcement must be written by developers
 - Potential for non-centralized rule enforcement
 - Data must be cleansed before LOAD or IMPORT operations
 - Rule enforcement MAY be less expensive in application layer
 - But MAYBE NOT!!!!
- Database enforcement of rules
 - Centralizes the rule enforcement
 - May cause an increase in database resource consumptions
 - May result in multiple trips to data server to validate or rules
 - Enforcement during LOAD and IMPORT operations
 - Visible to users and tools via the DB2 system catalog
 - There are limits to rule complexity
 - One example is limited UPDATE rules
 - Limits can be overcome using triggers
 - Rules can be used by query optimization

- Data intensive operations are best performed within the database engine
 - Enforcing RI is data intensive
 - Cascading or SET NULL deletes can be much more efficient versus multiple application calls to the data server
 - Massive insert operations better as LOAD from a performance perspective
- High volume insert applications can be a performance detriment
 - Especially for repeated child table inserts
 - IBM has been working on optimizing this! Not much of an issue!
- Using database enforced RI on code tables is NOT a good idea from a performance perspective

Table Check Constraints

- A rule that specifies what values can be contained in a column of a table
- Part of the table definition
 - CREATE TABLE
 - ALTER TABLE
- A check constraint is specified via a restricted form of a search condition
- Multiple ways to define check constraints

In a column definition

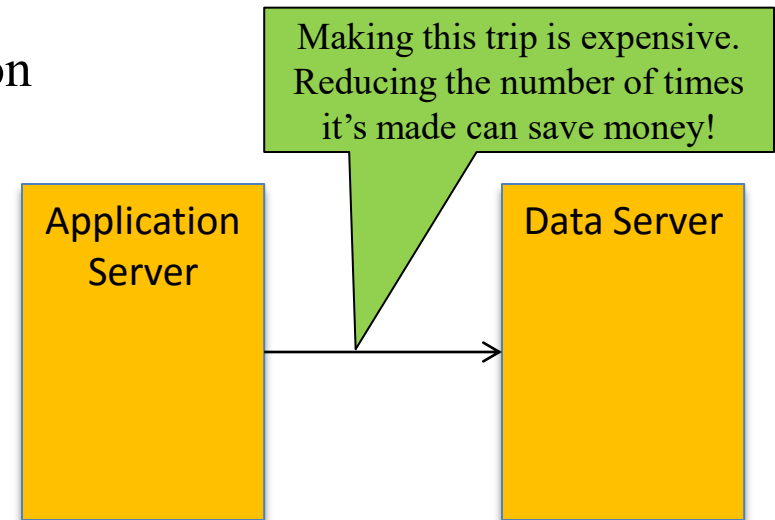
```
CREATE TABLE STOCK_TB
(
  STOCKNO      CHAR(6)  NOT NULL,
  NAME         VARCHAR(12) NOT NULL,
  ABVREVIAT   CHAR(3)
  EXCHANGE     CHAR(3) (EXCHANGE IN ('BEL', 'NEL', 'USA', 'JAP')) ,
  BUYDATE     DATE,
  CURRENCY     CHAR(3),
  BUYLEVEL    DECIMAL(9,2),
  SELL_LEVEL   DECIMAL(9,2),
  AMOUNT      SMALLINT,
  CHECK (CURRENCY IN ('EUR', 'DOL', 'YEN'),
  CONSTRAINT MAX_AMOUNT CHECK ((AMOUNT < 5 AND BUYLEVEL = >10000)
  OR (AMOUNT < 10 AND BUYLEVEL BETWEEN 1000 and 9999))
```

As a separate constraint, including complex multi-column rules

- Advantages
 - Centralized logic
 - Can be controlled/administered by a procedural DBA team
 - Easily documented logic in DB2 system catalog
 - Easy to code and implement
- Disadvantages
 - Not the best for performance, especially simple rules or multiple rules
 - More on next slide
 - Rule changes or mass data operations can negatively impact availability
 - Sets pending status
 - Limits table availability until integrity reestablished

Check Constraint Performance

- Care more about performance?
 - If simple rules required for data edits use application logic and NOT check constraints or RI constraints
 - All data edits can be done in application before database call
 - One call to database to insert or update the data
 - If multiple edits per table row application is the better performance choice
 - All data edits can be done in application before database call
 - One call to database to insert or update the data
- If enterprise control is desired
 - Check constraints in database
 - Make sure no redundant code in application
 - Be aware
 - Each constraint violation is an error
 - Multiple trips to database possible



Sequences

- Sequences are values stored in the DB2 system catalog
 - Numeric data type
 - Adjustable
 - Increment value
 - Start value
 - Maximum value
 - Ordering
 - Cycling
- Two types of sequences in use in DB2
 - Identity columns
 - Sequence Objects
- Why do sequences exist?
 - Improved database performance by reducing calls to the database
 - Automated application processes for generating key values
 - Centralize the generation of these values
 - Guaranteed unique values (assuming no reset of start value or cycling)
 - Avoids value collisions that could happen in an application process
 - Recoverable as part of crash recovery

SMALLINT
INTEGER
BIGINT
DECIMAL/NUMERIC

- A sequence that is attached to a column of a table
 - Part of the table definition
 - Can be generated always or by default
- New data entered into table will receive assigned sequences for the identity columns
 - If generated by default then the user can override the generated value by providing a value
- Only one identity column allowed per table
- Note: any past problems with identity columns are no longer an issue
 - Very flexible to manage, reset, alter, and override

```
CREATE TABLE INVOICE
(NUMBER INTEGER GENERATED ALWAYS AS IDENTITY
    START WITH 0
    INCREMENT BY 1,
CUSTID CHAR(30) ...
```

- Identity columns exist to help programmers generate key values
 - A significant performance and concurrency improvement over max tables or max value algorithms!
- Retrieval of last generated identity value in a program
 - SELECT from a data-change-statement
 - Use the IDENTITY_VAL_LOCAL() function
 - There are many rules and warnings on the use of this function
 - Read the manual thoroughly!

EMPNO generated as an identity column and returned in the result

```
SELECT EMPNO, FIRSTNME, LASTNAME
FROM FINAL TABLE (
INSERT INTO EMP2 (FIRSTNME, LASTNAME)
SELECT FIRSTNME, LASTNAME
FROM EMP
WHERE WORKDEPT = 'C01')
```

```
CREATE TABLE EMP2 (
EMPNO INT NOT NULL
GENERATED ALWAYS AS IDENTITY
(START WITH 1
INCREMENT BY 1),
FIRSTNME VARCHAR(12) NOT NULL,
LASTNAME VARCHAR(15) NOT NULL);
```

1	SALLY	KWAN
2	DELORES	QUINTANA
3	HEATHER	NICHOLLS
4	KIM	NATZ

- Sequence objects are sequences that are completely independent of tables
 - Can be used to generate values for multiple tables
 - Or for other purposes!
- Sequence values are retrieved via a sequence-reference

Identified by a two part identifier that includes a schema and a name

```
CREATE SEQUENCE BILLING.INVOICE_SEQ  
AS INTEGER  
START WITH 1  
INCREMENT BY 1
```

- These are expressions that can be placed almost anywhere in a SQL statement
- NEXT VALUE FOR sequence-name
 - expression generates and returns the next value for a specified sequence
 - One value per row of the result set
- PREVIOUS VALUE FOR sequence-name
 - returns the most recently generated value for the specified sequence for a previous statement within the current application process
 - can be used only if a NEXT VALUE expression specifying the same sequence name has already been referenced in the current application process
 - One value statement

```
SELECT EMPNO, FIRSTNAME, LASTNAME
FROM FINAL TABLE (
INSERT INTO EMP2 (EMPNO, FIRSTNAME, LASTNAME)
VALUES (NEXT VALUE FOR DANL.SEQ1, 'Bob', 'Rady');
```

Identity and Sequence Object Performance

- Identity values may outperform sequence objects for mass data movement operations
 - LOAD or IMPORT
 - Identity built into table definition
 - Sequence values have to be acquired separately
- Sequence values can, however, be acquired in bulk in a cursor

Single recursive SQL to deliver 100 sequence values

```
WITH GEN_KEYS (N) AS
(VVALUES (1)
 UNION ALL
 SELECT N+1 FROM GEN_KEYS
 WHERE N < 100)
SELECT NEXT VALUE FOR SEQ1
FROM GEN_KEYS
```

**Generates the number 1, use a
SELECT 1 FROM
SYSIBM.SYSDUMMY1 if on DB2 for
z/OS**

Recursive reference

Stop predicate to end recursion at 100

**CTE used to get next value from the
sequence object 100 times in one call**

Additional Generated Values

- GENERATE_UNIQUE() Function
- ROWID data type
- ROW CHANGE TIMESTAMP
- ROW BEGIN
- ROW END
- TRANSACTION START ID

Table independent

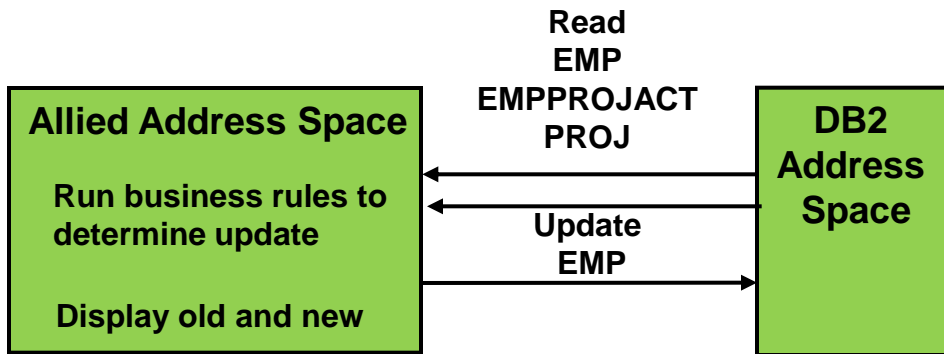
Part of a temporal table definition

Attached to columns of a table

- These are extremely powerful constructs for performance
 - They eliminate calls to the database
 - They shorten units of work
 - They shorten the length of time locks are held
- They are also powerful programming tools
 - INCLUDE and SET clauses can be specified
 - SELECT FROM
 - UPDATE
 - DELETE
 - INSERT
 - MERGE
 - Generated columns used just like other columns in the SELECT
 - Eliminates database calls
 - Eliminates programming
 - You can modify and report in a single statement

SELECT FROM - Intent and Example

- Reduce CPU
- Reduce calls to the data server
- Shrink the size of locks taken



```

SELECT EMPNO, SALARY, BONUS
FROM EMP E
WHERE EXISTS
  (SELECT 1
   FROM EMPPROJECT EPA
   INNER JOIN
     PROJ P
   ON P.PROJNO = EPA.PROJNO
   WHERE P.DEPTNO = 'C01'
   AND E.EMPNO = EPA.EMPNO)
FOR UPDATE OF SALARY, BONUS;
  
```

```

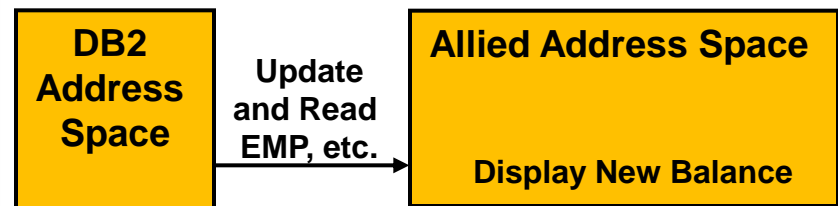
SELECT MAX(M.SALARY), MAX(M.BONUS)
FROM EMP M
  
```

```

UPDATE EMP E
SET (E.SALARY, E.BONUS) = ?, ?
WHERE CURRENT OF C1
  
```

```

SELECT EMPNO, SALARY, BONUS, OLDSALARY, OLDBONUS
FROM FINAL TABLE
  (UPDATE EMP E
   INCLUDE (OLDSALARY DEC(9,2), OLDBONUS DEC(9,2))
   SET (E.SALARY, E.BONUS) =
     (SELECT MAX(M.SALARY), MAX(M.BONUS)
      FROM EMP M), OLDSALARY = E.SALARY
     , OLDBONUS = E.BONUS
   WHERE EXISTS
     (SELECT 1
      FROM EMPPROJECT EPA
      INNER JOIN PROJ P
      ON P.PROJNO = EPA.PROJNO
      WHERE P.DEPTNO = 'C01'
      AND E.EMPNO = EPA.EMPNO))
  
```



Stored Procedures

- Stored procedures are an absolute key to multiple actions in support of a service call!
- Stored procedures can promote
 - Stronger security
 - Performance (replace multiple calls to service with one call)
 - Centralized logic

```

CREATE PROCEDURE PROB6A ( IN  IN_SET_MSG INT, IN  IN_DEPTNO
CHAR(03), OUT OUT_V1 VARCHAR(13))
  DYNAMIC RESULT SETS 1
P1: BEGIN
  DECLARE WS_DEPTNO CHAR(03) DEFAULT '  ';
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE EMP_C1 CURSOR WITH RETURN FOR
    SELECT E.EMPNO, E.LASTNAME, E.SALARY
    FROM   EMP E
    WHERE  E.WORKDEPT = IN_DEPTNO;
  CASE IN_SET_MSG
    WHEN 1 THEN SET OUT_V1 = 'Hello World';
    WHEN 2 THEN SET OUT_V1 = 'Goodbye World';
    ELSE      SET OUT_V1 = 'Invalid Entry';
  END CASE;
BEGIN
  DECLARE EXIT HANDLER FOR NOT FOUND
  BEGIN
    SIGNAL SQLSTATE '38001'
    SET MESSAGE_TEXT = 'Department does not exist';
  END;

```

```

SELECT D.DEPTNO INTO WS_DEPTNO
FROM   DEPT D
WHERE  D.DEPTNO = IN_DEPTNO;
END;

FOR VLOOP AS
  SELECT P.RESPEMP AS RESPEMP
  FROM   PROJ P
  WHERE  P.DEPTNO = IN_DEPTNO
DO
  UPDATE EMP
  SET SALARY = SALARY * 1.1
  WHERE EMPNO = VLOOP.RESPEMP;
END FOR  ;

OPEN EMP_C1;

END P1

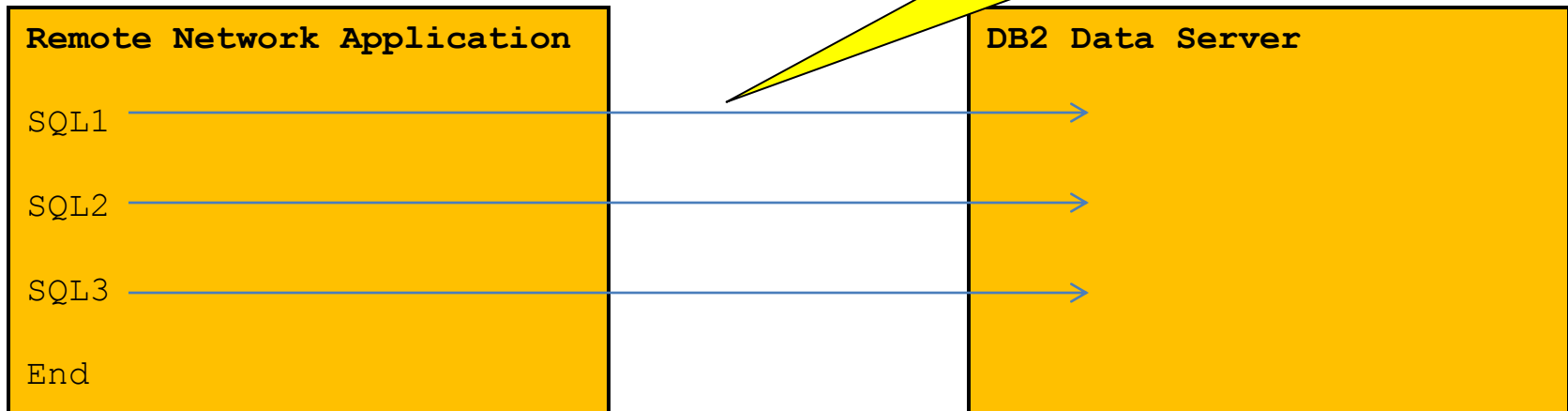
```

- Stored procedures represent an extension to the logic of a process
 - Run under thread of the caller
 - With in the unit of work of the caller
 - BUT, can be autonomous (native SQL procedures only)
 - If non-autonomous COMMIT and ROLLBACK apply across the thread
 - This includes work performed by the caller
 - Can accept input parameters, and return output parameters and result sets
- Languages available
 - REXX
 - COBOL
 - Java
 - C, C++
 - SQL procedure language (SQLPL)
 - PL/I
 - Assembler

Stored Procedure Performance

- The primary stored procedure performance advantage
 - Reduce the number of messages across a network
 - Only if many calls are replaced with a single call
 - A single statement stored procedure is a performance disadvantage!
- Native (SQLPL) stored procedures can represent a cost savings
 - They are zIIP eligible on z/OS (other types of stored procedures are not!)
 - They can run internal to the DB2 engine

Multiple network messages to process a transaction can be replace with a single call to a stored procedure



- SQL control statements (aka SQL procedure language) allow SQL statements to behave like programs
 - Logic control
 - Declare and set variables
 - Handle warnings and exceptions
- SQL control statements can be placed in
 - SQL Stored procedures
 - SQL User-defined scalar functions
 - SQL table functions (LUW only)
 - Triggers (LUW only)
 - Compound statements (LUW only)

DB2 SQL Procedural Language (SQL PL) is

- an extension to traditional SQL.
- a subset of the SQL Persistent Stored Modules (SQL/PSM) language standard. Supported by most major RDBMS
- It combines database language and procedural programming language

```
CREATE PROCEDURE TEST1
  (IN/OUT PARAMETERS
  DECLARATION)
-- Behavioral/definition options
-- Bind options
-- Begin of procedural logic
```

- A function is an operation defined by a function and zero or more parameters
 - A function is an expression
 - Built-in functions delivered with DB2
 - User-defined functions we use to extend the data server capabilities
- Several types of functions
 - Scalar
 - Zero or more inputs return a single value
 - Aggregate
 - Zero or more inputs
 - Operates on a set of data
 - Returns a single value
 - Row
 - Zero or more inputs
 - Returns a single row of one or more columns
 - DB2 for LUW only
 - Table
 - Zero or more inputs
 - Returns a table

- User-defined functions can extend the capabilities of the database
 - Now there are no limits to what the database can do
- UDFs can be internal or external
 - Sourced from an existing built-in or user-defined function
 - SQL based
 - SQLPL based
 - External programming language
 - Can access external resources
- UDFs can be
 - Scalar functions
 - Row functions (LUW only)
 - Table functions
 - Aggregate functions (only for sourced functions)

- This function accepts calculates age
 - Date as input
 - Small integer as output

```
CREATE FUNCTION DANL.AGE (BDATE DATE)
RETURNS SMALLINT
LANGUAGE SQL
DETERMINISTIC
CONTAINS SQL
RETURN (YEAR(CURRENT_DATE) - YEAR(BDATE))
```

```
SELECT E.EMPNO, E.LASTNAME, DANL.AGE(E.BIRTHDATE)
FROM EMP E
WHERE E.EMPNO = '000010'
```


- Concatenate two strings regardless of nulls

```
CREATE FUNCTION DANL.CCAT (STRING1 VARCHAR(4000),  
STRING2 VARCHAR(4000))  
RETURNS VARCHAR(4000)  
LANGUAGE SQL  
DETERMINISTIC  
CONTAINS SQL  
CALLED ON NULL INPUT  
BEGIN  
    IF STRING1 IS NULL THEN RETURN STRING2; END IF;  
    IF STRING2 IS NULL THEN RETURN STRING1; END IF;  
    RETURN STRING1 CONCAT STRING2;  
END!
```

**Virtually any SQLPL, SQL,
SQL/XML statement can go
into the routine body**

```
select danl.ccat ('BobRady',NULL) from sysibm.sysdummy1
```

SQL Table Function Example

- A table functions returns a table to a statement
 - Can be external
 - Can be SQL based (DB2 for z/OS and DB2 for LUW)
 - Can contain SQLPL (DB2 for LUW)

```
CREATE FUNCTION DANL.DEPTDTLS (DEPTNO CHAR(06))
RETURNS TABLE (AVGSAL DECIMAL(9,2), EMPCNT INT)
LANGUAGE SQL
NOT DETERMINISTIC
READS SQL DATA
RETURN
SELECT AVG(SALARY), COUNT(*)
FROM EMP
WHERE WORKDEPT = DEPTNO;
```

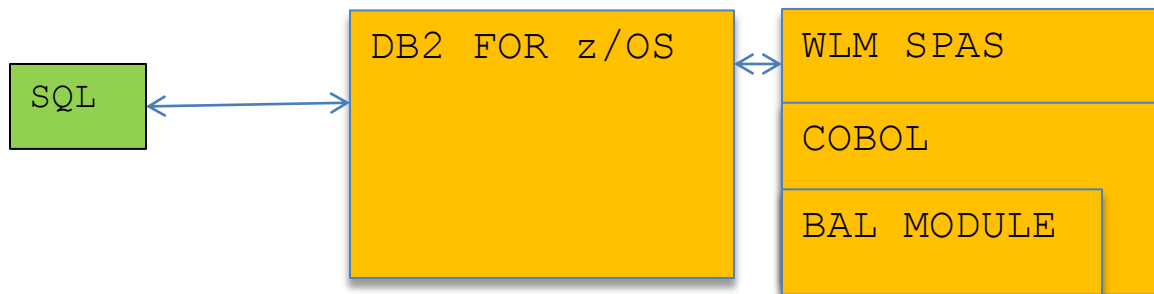
```
SELECT E.EMPNO, E.LASTNAME, DT.AVGSAL
FROM EMP E
INNER JOIN
      TABLE (DANL.DEPTDTLS (E.WORKDEPT)) AS DT
ON 1=1
```

External UDF Example

- Proprietary assembler routine in DB2 for z/OS delivers a soundex of a name
 - There is a DB2 soundex function
 - However, this is a custom version
- Source code not available
 - COBOL program written with parameters to match a DB2 scalar UDF
 - COBOL program calls assembler routine
 - COBOL program returns soundex value to DB2

```
SELECT LASTNAME, DANL.SOUNDEX(LASTNAME)
FROM EMP
WHERE EMPNO = '000010';
```

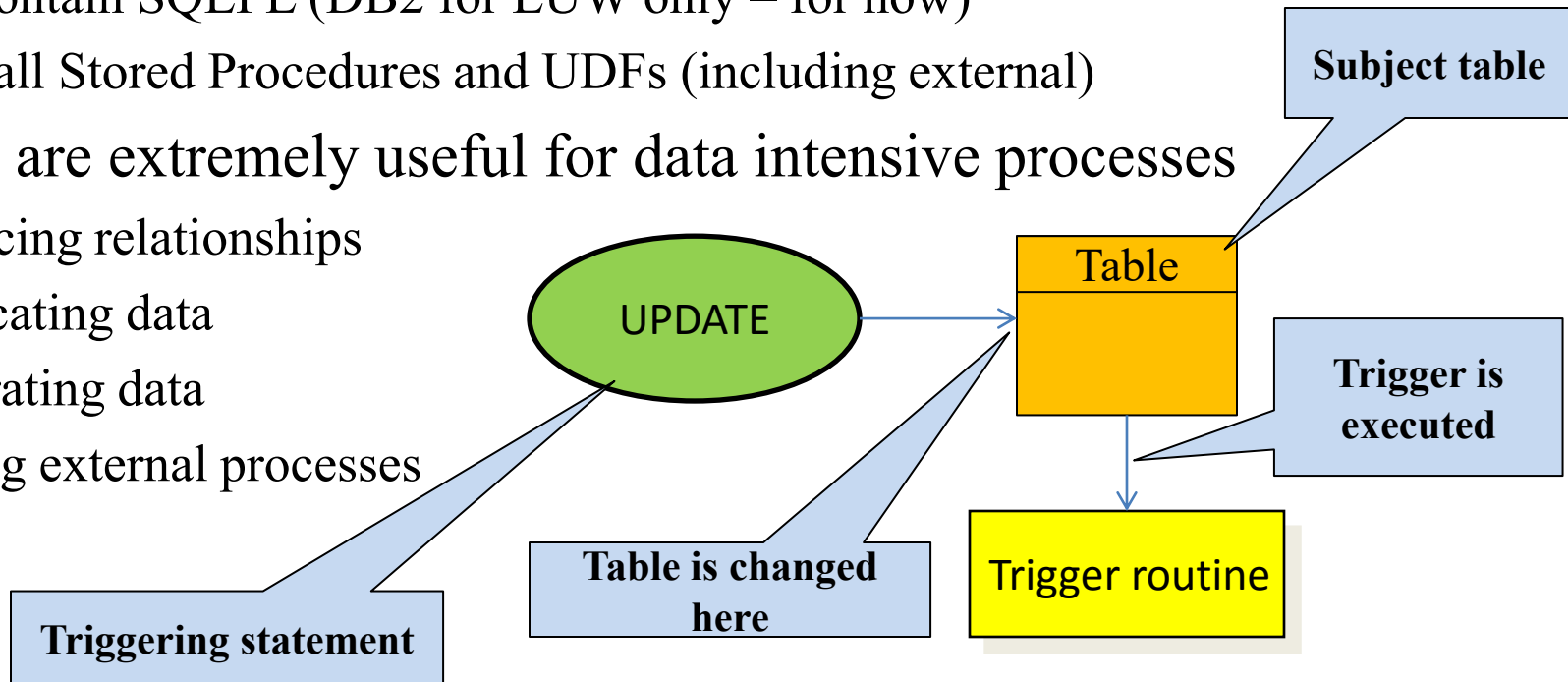
Super cool and super powerful!



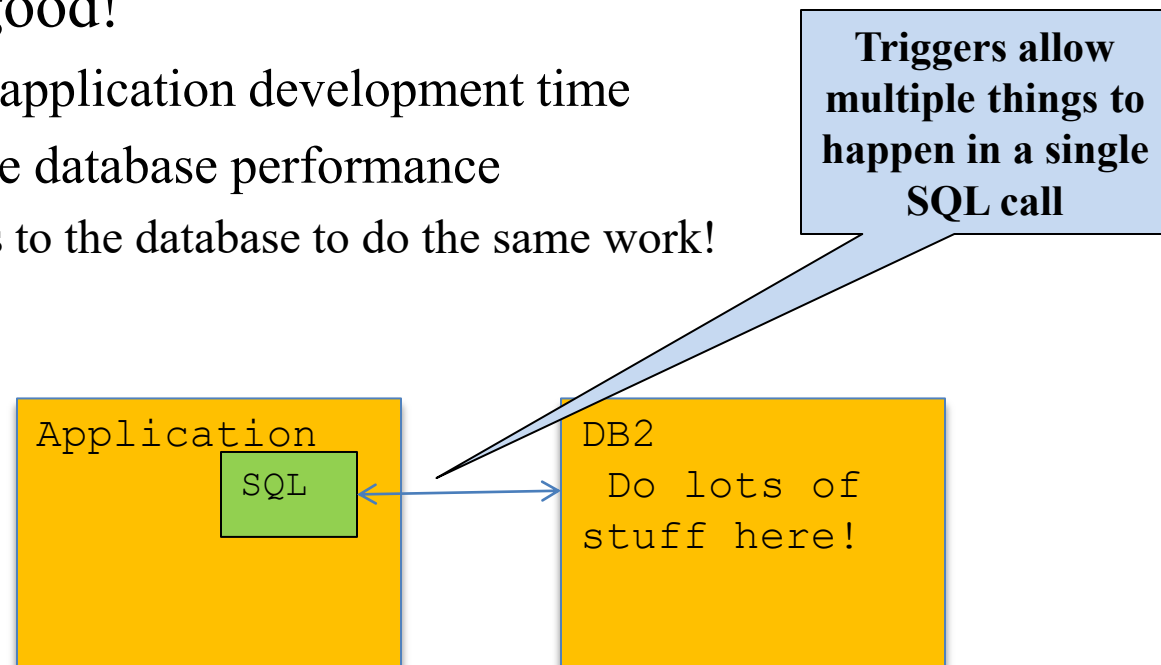
- There are virtually no performance advantage to UDFs
 - They exist for functionality
 - Same performance implications as built-in functions
 - DETERMINISTIC setting can affect materialization of nested table expressions
 - External functions execute as external processes
 - Can be very expensive to call these
- User-defined table functions can be a performance option
 - Can be used in some cases to force nested loop join
 - Can be same as correlated nested table expression
 - BUT!
 - May be merged with referencing statement
 - Can be expensive to execute
 - Best for transactions processing a small number of rows

- A trigger is a database object that contains application logic
 - Attached to a table
 - Activated (triggered or fired) upon execution of a data change statement
 - INSERT
 - UPDATE
 - DELETE
 - MERGE
 - Can contain SQLPL (DB2 for LUW only – for now)
 - Can call Stored Procedures and UDFs (including external)
- Triggers are extremely useful for data intensive processes
 - Enforcing relationships
 - Replicating data
 - Generating data
 - Calling external processes

Triggers are not activated by LOAD command or utility in most situations



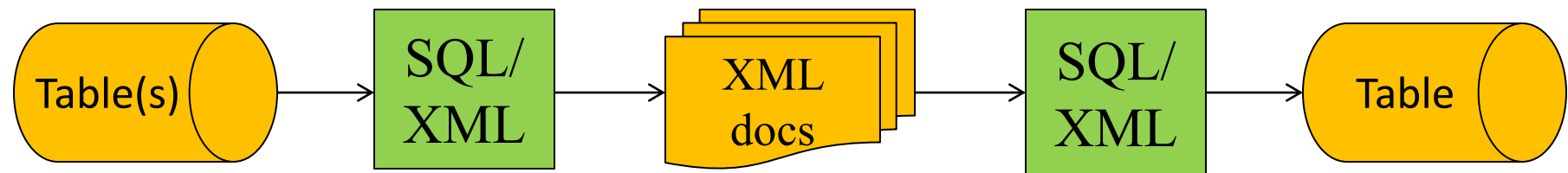
- Triggers reduce application development time
 - If an application needs to change multiple tables
 - If the action against one table requires an action against another table
 - If an external action is required based upon a table change
 - Incorporation of external functions into a process
 - Automatic data control
 - Centralized and reusable routines
- Triggers are all good!
 - Triggers reduce application development time
 - Triggers improve database performance
 - Reduced calls to the database to do the same work!



Trigger Components

- Subject table name
- Designation of action on table – triggering event
 - INSERT
 - UPDATE
 - DELETE
 - INSTEAD OF
- When the trigger is fired – trigger activation time
 - Before the object is changed
 - After the object is changed
- Trigger granularity
 - Statement – activated once per data change statement
 - Row – activated once per row changed
- Table referencing clause – transition variables and tables
 - Allows rows or table changing to be referenced
 - Before and after the data change
- Triggered-action
 - One or more SQL statements
 - The actions or processes that the trigger takes or invokes

- DB2 provides for the ability to deal with XML in several different ways
 - Read relational and output XML
 - Read XML and output relational
 - Read XML and output XML
- Powerful XML functions allow for relatively easy manipulation of the data
- Xquery and XPATH provide a new language of SQL/XML programming
- Built in validation of XML documents is available



- Built-in XML functions give full flexibility

XMLATTRIBUTES

Assigns one or more attributes to an element

XMLAGG

Aggregate function of elements and includes an ORDER BY

XMLELEMENT

Creates an element

XMLFOREST

Creates multiple elements

```
SELECT XMLELEMENT(NAME "Department",
XMLATTRIBUTES(D.DEPTNO AS "DID"),
D.DEPTNAME,E.EMPINFO) FROM DEPT D
INNER JOIN
(SELECT WORKDEPT,
XMLAGG(
XMLELEMENT(NAME "Employee",
XMLATTRIBUTES(EMPNO AS "EID"),
FIRSTNME || ' ' || LASTNAME,
XMLFOREST(BIRTHDATE as "BirthDate"
, HIREDATE AS "HireDate",
SALARY as "Salary"))
ORDER BY EMPNO)
AS EMPINFO FROM EMP
GROUP BY WORKDEPT) E
ON E.WORKDEPT = D.DEPTNO
WHERE D.DEPTNO IN ('D11', 'D21')
```

```
<Department DID="D11">MANUFACTURING SYSTEMS<Employee EID="000060">IRVING
STERN<BirthDate>1975-07-07</BirthDate><HireDate>2003-09-
14</HireDate><Salary>72250.00</Salary></Employee><Employee EID="000150">BRUCE
ADAMSON<BirthDate>1977-05-17</BirthDate><HireDate>2002-02-
12</HireDate><Salary>55280.00</Salary></Employee><Employee EID="000160">ELIZABETH
PIANKA<BirthDate>1980-04-12</BirthDate><HireDate>2006-10-
11</HireDate><Salary>62250.00</Salary></Employee><Employee EID="000170">MASATOSHI
YOSHIMURA<BirthDate>1981-01-05</BirthDate><HireDate>1999-09-
15</HireDate><Salary>44680.00</Salary></Employee></Department>
```

```
<Department DID="D21">ADMINISTRATION SYSTEMS<Employee EID="000070">EVA
PULASKI<BirthDate>2003-05-26</BirthDate><HireDate>2005-09-
30</HireDate><Salary>96170.00</Salary></Employee><Employee EID="000230">JAMES
JEFFERSON<BirthDate>1980-05-30</BirthDate><HireDate>1996-11-
21</HireDate><Salary>42180.00</Salary></Employee><Employee EID="000240">SALVATORE
MARINO<BirthDate>2002-03-31</BirthDate><HireDate>2004-12-
05</HireDate><Salary>48760.00</Salary></Employee></Department>
```

Product DESCRIPTION Column for PID="100-100-01"

```
<product pid="100-100-01"><description><name>Snow Shovel, Basic 22  
inch</name><details>Basic Snow Shovel, 22 inches wide, straight handle with D-  
Grip</details><price>9.99</price><weight>1 kg</weight></description></product>
```

XMLTABLE
A table function that
uses an XPath
expression to specify
a path to an element
and then column
definitions for
subelements

```
SELECT DSC.*  
FROM PRODUCT P  
INNER JOIN  
XMLTABLE ('$d/product/description'  
          PASSING P.DESCRPTION AS "d"  
          COLUMNS "NAME" CHAR(30) PATH 'name'  
                  , "DETAILS" VARCHAR(70) PATH 'details'  
                  , "PRICE" DEC(7,2) PATH 'price'  
          ) AS DSC  
ON 1 = 1 WHERE PID='100-100-01'
```

NAME	DETAILS	PRICE
Snow Shovel, Basic 22 inch	Basic Snow Shovel, 22 inches wide, straight handle with D-Grip	9.99

- XML functions use Xpath expressions to apply restriction and selection
- XML retrieval can also be written as Xquery instead of SQL/XML

Product DESCRIPTION Column for PID="100-100-01"

```
<product pid="100-100-01"><description><name>Snow Shovel, Basic 22  
inch</name><details>Basic Snow Shovel, 22 inches wide, straight handle with D-  
Grip</details><price>9.99</price><weight>1 kg</weight></description></product>
```

```
SELECT XMLQUERY('$d/product/description/name'  
    PASSING DESCRIPTION AS "d")  
FROM PRODUCT  
WHERE  
XMLEXISTS('$d/product/description[price<10]'  
    PASSING DESCRIPTION AS "d")
```

XMLQUERY
Pulls the product name
element out of the document

XMLEXISTS
Applies a predicate where
price is less than 10 dollars

```
<name>Snow Shovel, Basic 22 inch</name>  
<name>Ice Scraper, Windshield 4 inch</name>
```

- The XML functionality in DB2 is extremely powerful
- Once over the learning curve XML functions and XPath are easy
- Faster time to delivery
 - XML generation from relational data
 - XML storage and processing within DB2
 - Full functionality without an application program
- XML schema validation is also built into DB2
 - Important for backup and recovery coordination
 - Important for document control
 - Validation installed into the database
- DB2 query transformation for XML is very powerful and flexible

- XML is stored separate from the table data
 - Additional access time required
- XML processing in DB2 uses additional CPU resources
- XML indexes can be defined to improve search performance
- Performance choices
 - Need relational data and XML
 - Shred documents into tables
 - Rebuild documents coming out
 - Share relational data
 - Simply need XML and no XML processing
 - Stored data as a CLOB or VARCHAR
 - Want full XML processing
 - Use the XML data type and XPath or XQuery
- I have seem DB2 XML processing outperform external applications!

- JavaScript Object Notation
 - Lightweight data exchange format
 - Textual
 - Subset of JavaScript
- Why JSON over XML?
 - Minimal set of functionality
 - Small message footprint when compared to Java
 - Makes it better for portable app data support
 - JSON is all about working with objects
 - Can be used directly in JavaScript
- JSON can be considered a NoSQL technology
 - Based upon key-value pairs
 - Data is not stored relationally in the database
- Why JSON DB2 support?
 - leveraging the NoSQL JSON paradigm and flexible schemas
 - preserve the traditional DBMS capabilities, leveraging existing skills and tools

- JSON is a simple way to describe and store data
- Limited data types
 - Four primitives
 - Strings
 - Numbers
 - Booleans
 - Null
 - Two structures
 - Objects
 - Arrays

```
{
  "PO": {
    "@id": 123,
    "@orderDate": "2013-11-18",
    "customer": { "@cid": 999 },
    "items": {
      "item": [
        {
          "@partNum": "872-AA",
          "productName": "Lawnmower",
          "quantity": 1,
          "USPrice": 149.99,
          "shipDate": "2013-11-20"
        },
        {
          "@partNum": "945-ZG",
          "productName": "Sapphire Bracelet",
          "quantity": 2,
          "USPrice": 178.99,
          "comment": "Not shipped"
        }
      ]
    }
  }
}
```


- JSON wire listener
 - Not recommended
 - Not discussed here
- JSON NoSQL Java API
 - Not recommended
 - Not discussed here
- IBM supplied UDFs for JSON support
 - Recommended.
 - Documented for DB2 11 for z/OS
 - Not documented well for DB2 for LUW or DB2 10 for z/OS
 - Additional UDFs available
 - Must use NoSQL API to get this stuff created
- Other alternatives
 - Cloudant?
 - DIY – Do it yourself!
 - Strings in the database and JavaScript in the application
- The recommendation is to use SQL and not the NoSQL API

- Store JSON as text
 - VARCHAR if less than 32K
 - CLOB if less than 2G
 - Best if JSON simply stored and retrieved in DB2 and not manipulated
 - Same as with XML
 - All manipulation performed in the application
- Shred JSON and store as relational
 - This is all up to you! No automation here!
- Use DB2 JSON functionality
- JSON payloads are part of DB2 as a service in z/Connect

- JSON can be stored internally in DB2 as BSON
 - Binary JSON
- JSON_VAL built-in function
 - Extract and retrieve JSON data into SQL data types from BSON
- Two DB2 11 for z/OS user-defined functions
 - SYSTOOLS.JSON2BSON to convert JSON to BSON
 - SYSTOOLS.BSON2JSON to convert BSON to JSON
- SQL support for JSON is possible in DB2 for LUW
 - Not documented as far as I can tell
- Indexes on JSON data stores are also possible in DB2

Creating a Table to Store JSON

- In reality you are creating a table with a BLOB column
 - This is just like any other table with a BLOB column
 - Same BLOB recommendations apply
 - In this example we are hoping almost all the objects take less than 25K
- The DB2 functions are used to store and retrieve the JSON
 - Only in DB2 11 for z/OS

```
CREATE TABLE JSONCUSTOMER
          ( ID INTEGER NOT NULL,
            DATA BLOB(16M) INLINE LENGTH 25000,
            PRIMARY KEY(ID) CCSID UNICODE;
```

```
INSERT INTO JSONCUSTOMER VALUES (
  101,
  SYSTOOLS.JSON2BSON(
    '{"Customer":{"@cid": 999,
                  "name": "Michael",
                  "age": 31,
                  "telephone": "234-343-2343",
                  "country": "USA"
                }
    }' ));
```

Selecting JSON Data

- There are different ways to retrieve the JSON data

```
SELECT SYSTOOLS.BSON2JSON(DATA)
FROM   JSONCUSTOMER
WHERE  ID = 101;
```

```
SELECT JSON_VAL(DATA, 'Customer.name', 's:40')
FROM   JSONCUSTOMER
WHERE  JSON_VAL(DATA, 'Customer.@cid', 'i:na')=999
```

JSON_VAL can be used in SELECT, WHERE, ORDER BY

Data type for conversion to relational

Joins and many other SQL capabilities are also possible!

- It is possible to create an index on expression
 - Improve performance of query operations
 - Similar to any other index on expressions

```
CREATE INDEX IX1 ON JSONCUSTOMER (  
JSON_VAL(DATA, 'Customer.@cid','i:na'));
```

This index can now be
used for fast query
performance

Updating and Removing JSON Data

- Entire JSON BLOB value is updated or removed

```
UPDATE JSONCUSTOMER
SET DATA = SYSTOOLS.JSON2BSON(
    '{"Customer":{"@cid": 999,
        "name": "Bob",
        "age": 31,
        "telephone": "234-343-2343",
        "country": "USA"
    }'
)
WHERE JSON_VAL(DATA, 'Customer.@cid', 'i:na') = 999;
```

```
DELETE FROM JSONCUSTOMER
WHERE JSON_VAL(DATA, 'Customer.@cid', 'i:na')=999
```

- No limit to DB2 and SQL except your imagination
- Advanced features can be a performance improvement if you use it to reduce trips to the data server
- Advanced features allows for flexible features to be written and deployed quickly
- Advanced features will be a keystone to DB2 as a service performance
- Highly portable SQL liberates you from one specific platform
- It's also a lot of fun!

- DanL Database Consulting
 - DB2 consulting services
 - Training
 - Performance analysis and tuning
 - Database design
 - DB2 migrations



DB2 for z/OS DBaaS and the Advantages of Automated Objects

Daniel L Luksetich

DanL Database Consulting

danl@db2expert.com