ANSI SQL Transparent Dynamic Multipath Hierarchical Structured Data Processing For Relational, XML, & Legacy Data

Michael M David and Lee Fesperman
Advanced Data Access Technologies, Inc.
www.adatinc.com

The example results shown in this presentation can be reproduced on our online interactive prototype at www.adatinc.com/demo.html

Previous Database History

- Hierarchical processing popular 40 years ago
  - Used user navigational processing
- Multipath hierarchical query processing used
  - Used nonprocedural navigationless processing
  - Made possible by automatic semantic processing
- Relational processing replaces hierarchical
  - Also uses nonprocedural navigationless processing
  - But more flexibility with data independence from join
- Hierarchical processing technology forgotten
  - No Internet to store hierarchical processing knowledge
- Hierarchical structures popular again with XML
  - User procedural navigation is back again
  - Where is navigationless access for structured access?

SQL/XML Industry Problems

- **Only proprietary vendor solutions**
  - All vendor solutions incompatible with each other
  - Integration solutions are XML centric and procedural
- **No satisfactory XML integration solution**
  - Hierarchical data value loss
  - XML not fully integrated into SQL
  - XML structured data processed as semistructured
- **No hierarchical processing standard**
  - Invalid hierarchical processing result is possible
  - Invalid hierarchical structure result is possible
- **Markup and structured data processed the same**
  - Structured data needs to be processed differently!

SQL/XML Structured Data Processing Problems

- **Limited to linear single path processing**
  - Query data selection limitations
  - Relational join single path mindset
- **Requires user database navigation**
  - Loss of dynamic hierarchical processing
  - Limits complex hierarchical processing
  - Prevents seamless & transparent processing
- **Requires specific vendor user training**
  - Requires XML and vendor trained users

A Working ANSI SQL XML Solution

- Limits processing to only structured data
  - Allows unambiguous nonprocedural querying
  - Enables navigationless schema-free operation
- Only performs hierarchical operations
  - Only uses hierarchical Left Outer Join operation
  - Naturally supports full hierarchical structures
  - Allows hierarchical structure-aware operation
- Supports inherent hierarchical processing
  - This solves relational/XML data integration
  - This also solves SQL to XML system mapping
  - Transparent multipath hierarchical processing

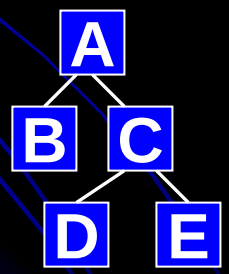
Structured Data Automatic Processing Benefits

- SQL transparent XML integration
  - ANSI SQL-92 syntax and semantics, not XML centric
  - Nonprocedural and navigationless, Interactive
  - No knowledge of structure necessary, schema-free
- Multipath nonlinear hierarchical processing
  - Hierarchically accurate results automatically
  - Dynamic hierarchical processing optimization
  - Dynamic output uses hierarchical result structure
- SQL hierarchical views fully functional
  - Global views with no overhead and maximum reuse
  - Global hierarchical queries now possible
  - Hierarchical data filtering & XML keyword search

Current XML Hierarchical Data

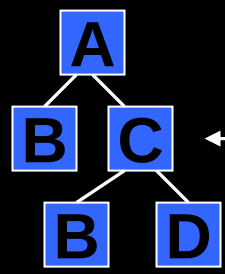
- XML Was Created as a Markup Language
- Unstructured Mapped to Semistructured
- Structured Vs. Semistructured Data
- Same as Fixed Vs. Fuzzy Meaning
- Why Semistructured Requires Navigation

Structured Data



Unambiguous Query Structure

Semistructured Data



Ambiguous Query Structure

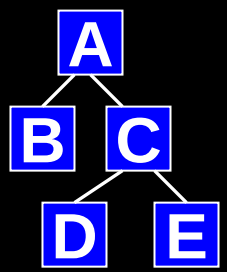
The multiple B nodes in this structure make it ambiguous for querying without user navigation.

Why Hierarchical Data structures are Powerful

- Automatic Data and Path Reuse
- Extending Path Increases Data Value
- Adding Paths Increase Data Value Further
- Hierarchical XML Becoming Ubiquitous
- Data Structures are Unambiguous

Data/Path

1) A/B
2) A/C
3) A/C/D
4) A/C/E

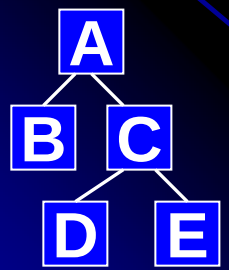


Creating more value than is collected, automatic nonlinear data value increase.

Why Hierarchical Structured Data Processing is Powerful

- Dynamic Multipath Query Combinations
- Multipath Data Value Grows Continually
- Every Node is Related to Every Other Node
- No. of Possible Queries Becomes Unlimited
- Automatic Processing = Unlimited Complexity

Mview



Increasingly Complex Multipath Queries

1) SELECT B,C FROM Mview
2) SELECT B,D FROM Mview WHERE A=2
3) SELECT A,B FROM Mview WHERE E=5
4) SELECT B,C FROM Mview WHERE D=1 AND E=5
5) SELECT D,E FROM Mview WHERE B=3 OR C=4

Naturally utilizes the inherent hierarchical semantics in multipath queries.

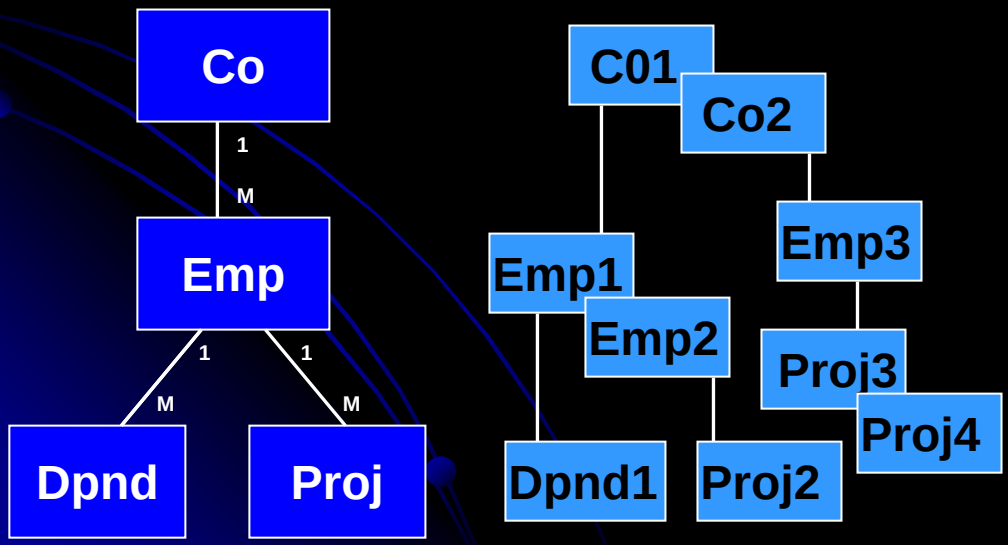
Hierarchical Data Structure Types

- XML is self defining contiguous and nested
  - No foreign key relation needed to make structure
- IBM's IMS database is discontinuous
  - Internally linked
- Structured VSAM is contiguous
  - With hierarchical level & data occurrence counts
- Flat tabular objects hierarchically modeled
  - Relational tables, flat files, spread sheets

All support same hierarchical operation and principles

Hierarchical Structured Data Makeup

- Multiple Node Types
- Nodes Support Multiple Data Occurrences
- Naturally Built With 1 to M Relationships
- Hierarchical Data Preservation Operation- Naturally Support Variable Length Paths

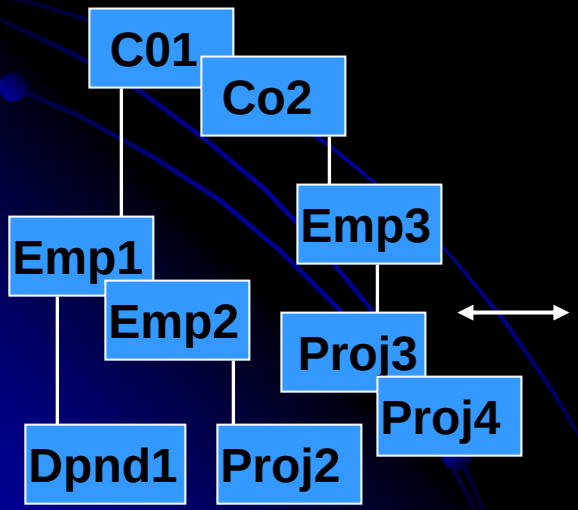


Not to be confused with function driven single node external hierarchical data structure processing.

Suitable for IMS, Structured VSAM, XML, COBOL FD and hierarchically modeled Relational and flat data.

SQL Hierarchical Structured Data Can Be Stored in Relational Rowsets

- Multiple Node Types in Relational Rowset
- Node Multiple Data Occurrences in Rowset
- Multiple Pathways in Rowset
- Variable Length Pathways in Rowset

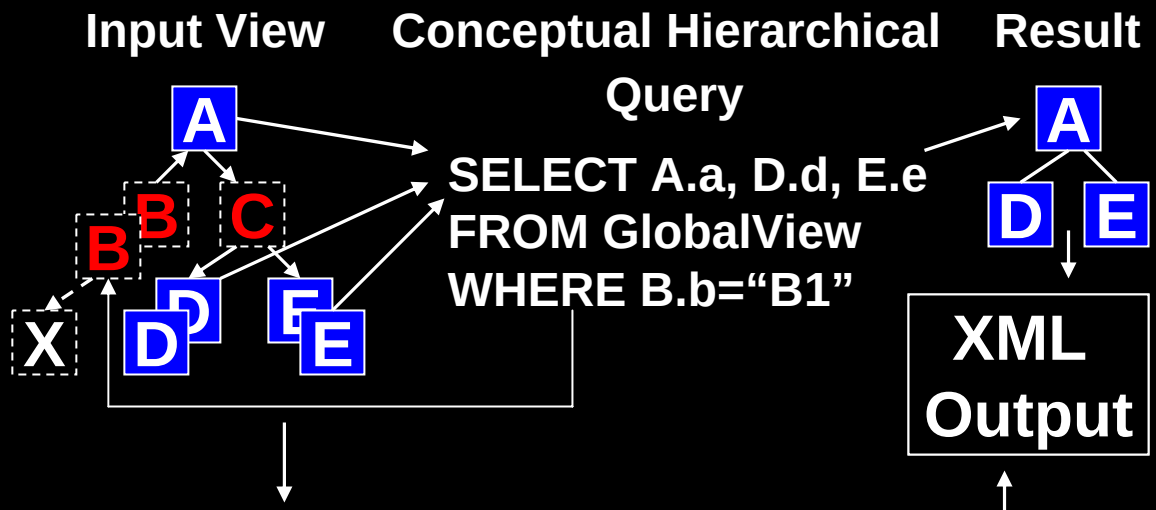


Relational Rowset

Co	Emp	Dpnd	Proj
Co1	Emp1	Dpnd1	
Co1	Emp2		Proj2
Co2	Emp3		Proj3
Co2	Emp3		Proj4

Hierarchical structure preserved in rowset, in and back out again.

Mapping Relational SQL to Hierarchical XML



Working Set

A	B	C	D	E
A1	B1	C1	D1	E1
A1	B1	C1	D2	E1
A1	B1	C1	D1	E2
A1	B1	C1	D2	D2
A1	B2	C1	D1	E1

Result Set

A	D	E
A1	D1	E1
A1	D2	E1
A1	D1	E2
A1	D2	D2

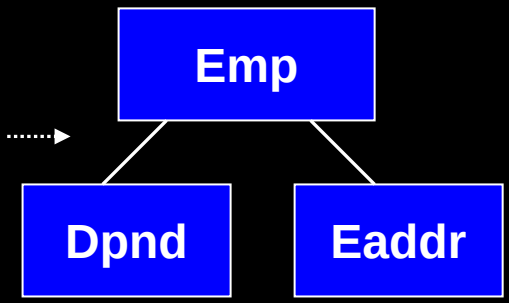
Shown on this slide:

1. Conceptual Level
2. Multipath Processing
3. Hierarchical Filtering
4. Dynamic Data Select
5. Access Optimization
6. Global View Support
7. Schema-free Query
8. Node Promotion
9. Auto Output Format

Relational Logical Hierarchical View

```
CREATE VIEW EmpView AS
SELECT * FROM Emp
LEFT JOIN Dpnd
ON EmpID=DpndEmpID
     AND DpndCode='D'
LEFT JOIN Eaddr ON EmpCustID=EaddrCustID;
```

Hierarchical SQL data modeling and processing semantics defined.




```
SELECT EmpID, DpndID,
          EaddrID
FROM EmpView
```

```
<root>
 <emp empid="Emp01">
  <dpnd dpndid="Dpnd01"/>
  <eaddr eaddrid="Addr01"/>
 </emp>
 <emp empid="Emp02">
  <eaddr eaddrid="Addr03"/>
 </emp>
</root>
```

Logical hierarchical SQL view and semantics processed directly by relational engine making it operate fully hierarchically.

XML Physical Hierarchical View

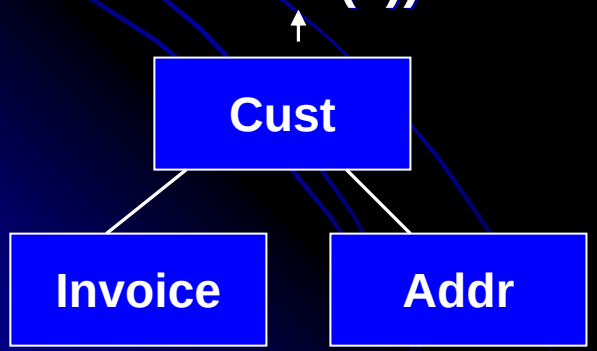
```
CREATE XML CustView
Cust(
  CustID Char(8),
  CustStoreID Char(8)),
Invoice(
  InvID Char(8),
  InvCustID Char(8),
  InvStatus Char(8)) Parent Cust,
Addr(
  AddrID Char(8),
  AddrCustID Char(8),
  AddrState Char(8)) Parent Cust
```

Converted to SQL CustView

```
CREATE VIEW CustView AS
SELECT * FROM Cust
LEFT JOIN Invoice
ON CustID=InvID
LEFT JOIN ADDR
ON CustID=AddrID;
```

```
SELECT Cust, Invoice, Addr
FROM CustView
```

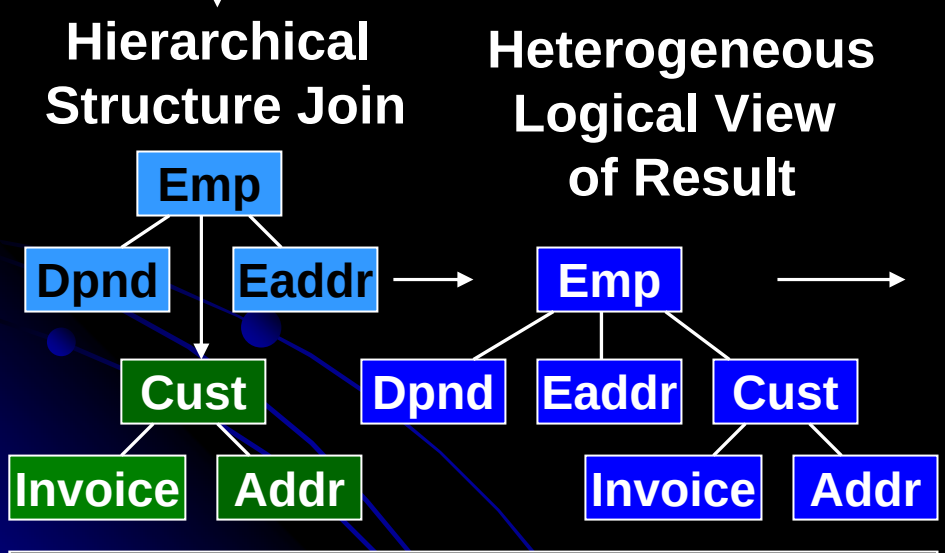
```
<root>
  <cust custid="Cust01">
    <invoice invid="Inv01"/>
    <invoice invid="Inv02"/>
    <addr addrid="Addr01"/>
```



This SQL created outer join view syntax is used as a hierarchical map of the physical IMS CustView for seamless operation.

Joining Heterogeneous Views

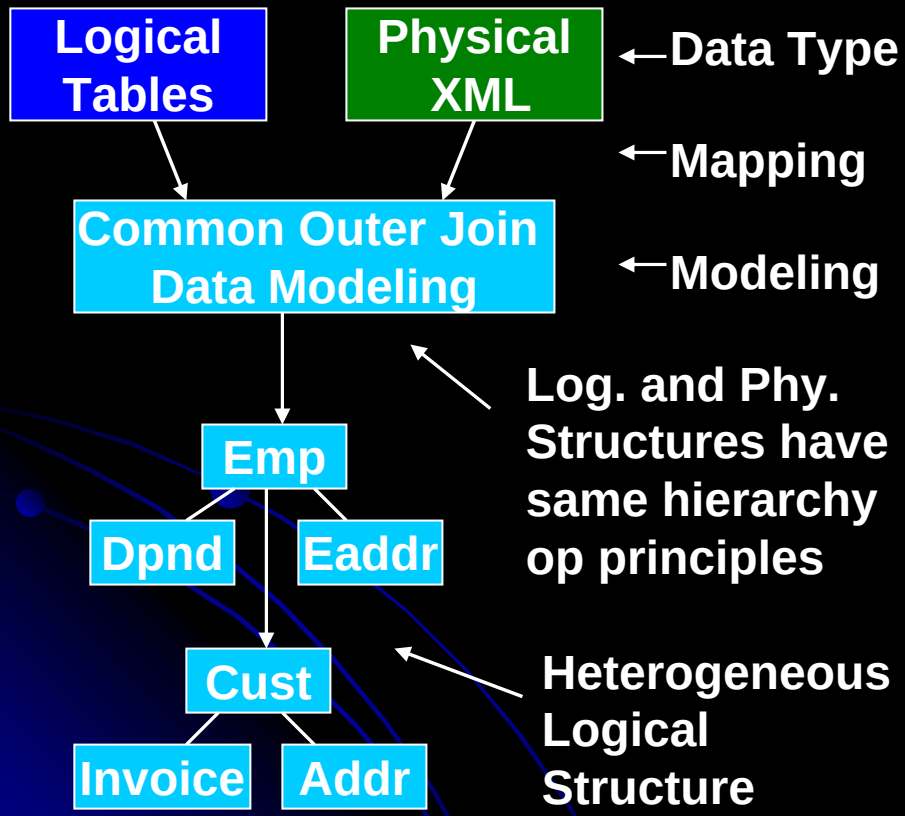
```
SELECT EmpID, DpndID, EaddrID,
       CustID InvID, AddrID
FROM EmpView
LEFT JOIN CustView
ON EmpCustID=CustID
```



Emp logical and Cust physical views are both hierarchically modeled in the same way in SQL making for a seamless, unified and logical hierarchical join.

```
<root>
 <emp empid="Emp01">
  <dpnd dpndid="Dpnd01"/>
  <eaddr eaddrid="Addr01"/>
  <cust custid="Cust01">
   <invoice invid="Inv01"/>
   <invoice invid="Inv02"/>
   <addr addrid="Addr01"/>
  </cust>
 </emp>
 <emp empid="Emp02">
  <eaddr eaddrid="Addr03"/>
  <cust custid="Cust03">
   <addr addrid="Addr03"/>
  </cust>
 </emp>
</root>
```

Logical Hierarchical Structures Offer Flexible Relational/XML Integration



Data Model:

- Natural
- Common

Features:

- Abstraction
- Consistency
- Seamless

Capabilities:

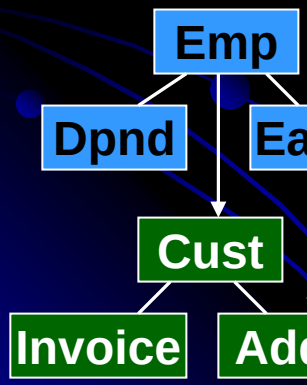
- Data Integration
- Solves Rel and Hier Problems
- Separates Structure from Data
- Hierarchical Structure Flexibility
- Offers Flexibility to Fixed Structures

Logical hierarchical structures solve hierarchical fixed structure problems.

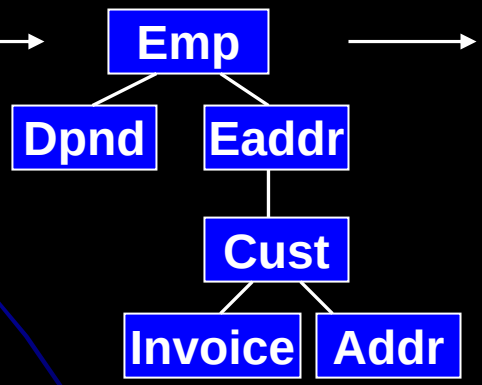
Heterogeneous Data Structure Mashup Uses Linking Below Root

```
SELECT EmpID, DpndID, InvID,
       AddrID, EaddrID, CustID
FROM EmpView LEFT JOIN
     CustView ON EaddrID=AddrID
WHERE CustID <> ”Cust02”
```

Link Below Root



Result Structure



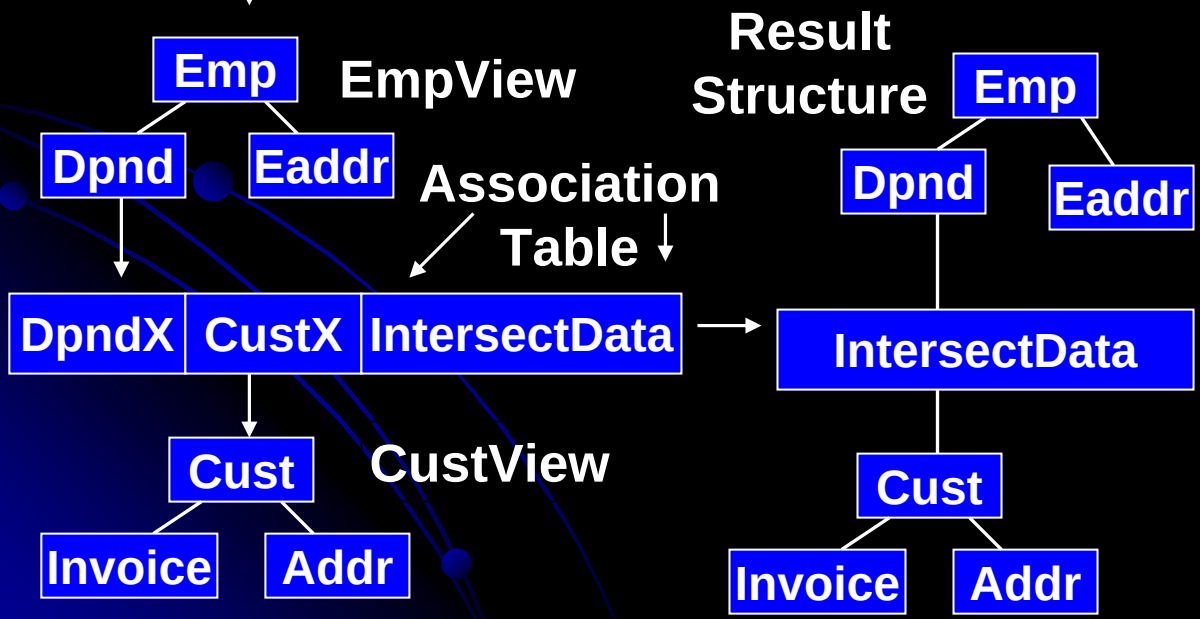

```
<root>
 <emp empid="Emp01">
  <dpnd dpndid="Dpnd01"/>
  <eaddr eaddrid="Addr01">
   <cust custid="Cust01">
    <invoice invid="Inv01"/>
    <invoice invid="Inv02"/>
    <addr addrid="Addr01"/>
   </cust>
  </eaddr>
 </emp> ...</root>
```

Dashed arrow is linkage.
Solid arrow is new structure.

Mashups allow linking anywhere into the lower level structure. We determined this was valid and what the new semantic structure is.

Association Table Use

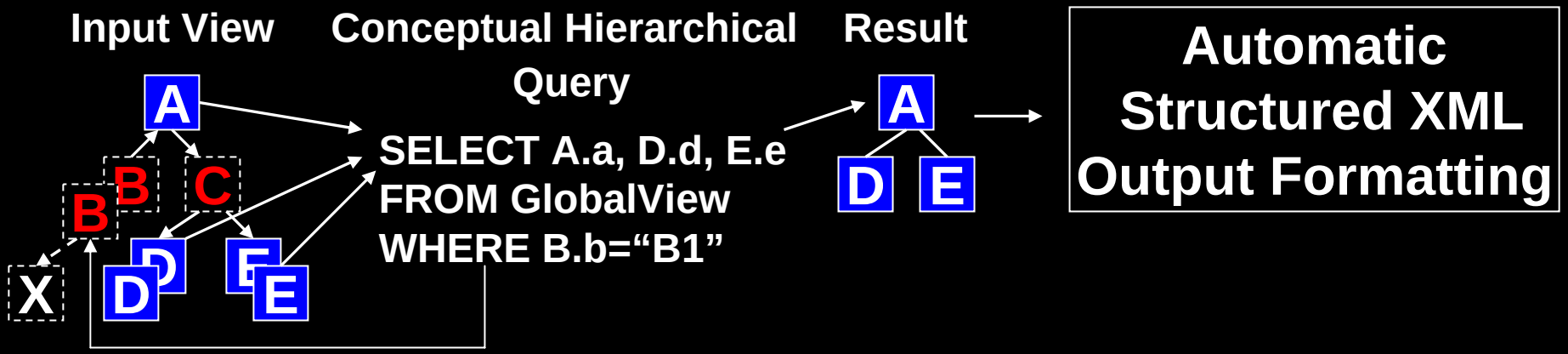
```
SELECT EmpID, DpndID, EaddrID, CustID, InvoiceID
       AddrID, IntersectData
FROM EmpView
LEFT JOIN AssociationTable On DpndID=DpndX
LEFT JOIN CustView  ON CustID=CustX
```



Association Table Capabilities added:

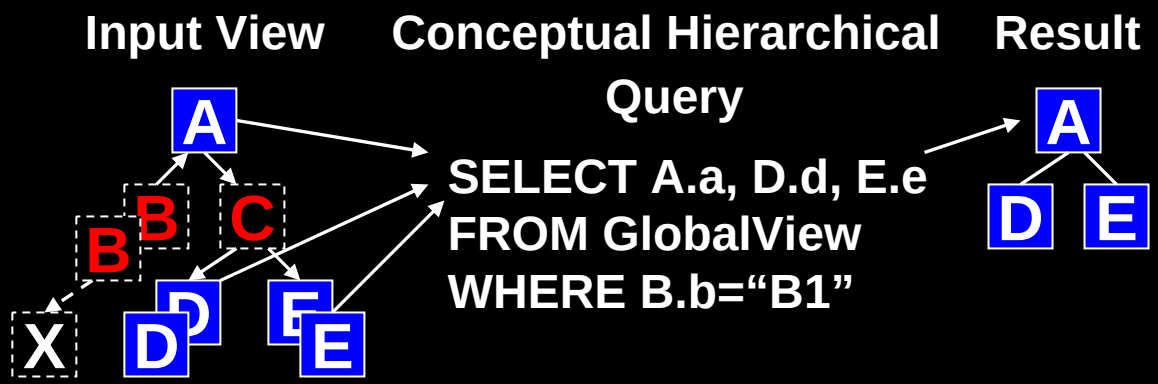
- External Relationships
- M to M Relationships
- Intersecting Data
- Can be Transparent
- Retains Hier Structure

Hierarchical Data Filtering, Auto Output & Structure-free Processing



- Hierarchical WHERE clause global data filtering
  - Cousin nodes like node C above are filtered from B node
  - This makes this a more internally complex multipath query
- Structure-free processing
  - Navigationless XML access, no need to know structure
- Automatic structure-aware output formatting
  - Result structure known from outer join syntax modeling

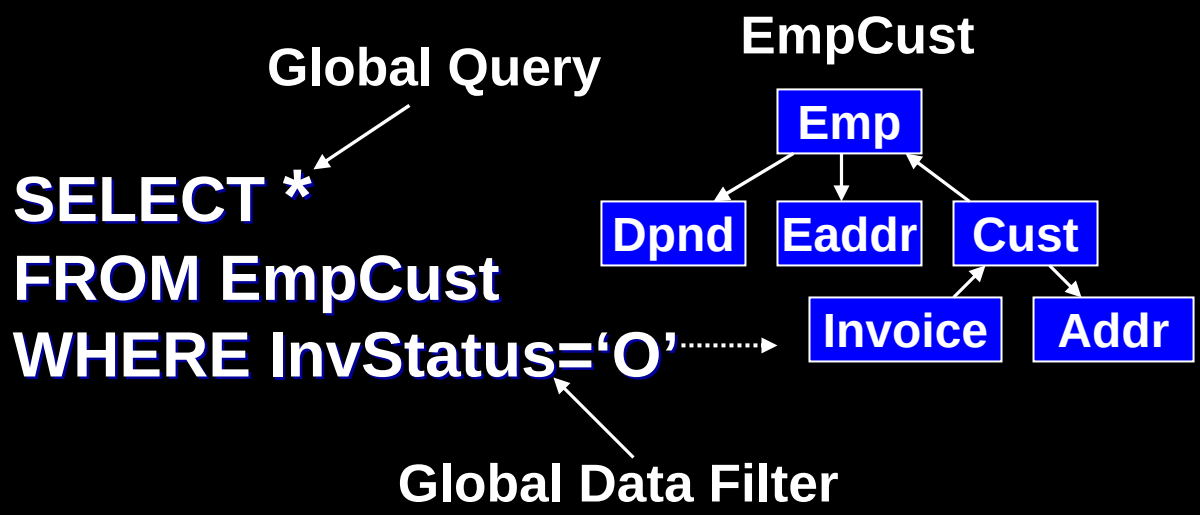
Optimization, Global Views and Node Promotion



This is a conceptual query where the input structure is not known and the output adapts to the dynamic result.

- Hierarchical optimization can remove from access -
  - Unreferenced data nodes not on path to referenced data
  - Dynamically controlled by SQL's variable SELECT list
- Optimization makes all views global views
  - Because they have no overhead for unreferenced fields
- Node promotion closes around unselected nodes
  - This happens in relational processing too

Global Query Uses Global Views



Global views allow entire structures to be defined and queried with no overhead, more user friendly. Global query allows hierarchical filtering of entire view.

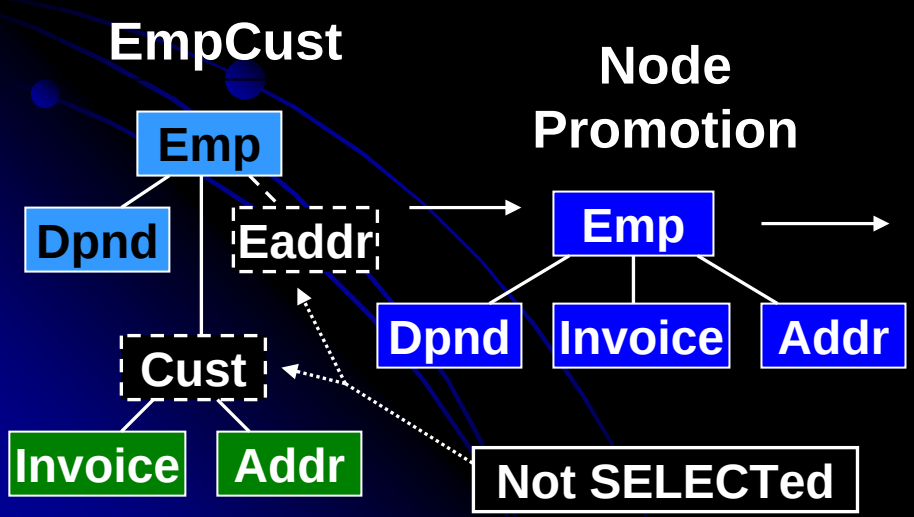
```
<root>
 <emp empid="Emp01" empstoreid="Store01" empcustid="Cust01"
      empstatus="F">
  <dpnd dpndid="Dpnd01" dpndempid="Emp01" dpndcode="D"/>
  <eaddr eaddrid="Addr01" eaddrcustid="Cust01" eaddrstate="CA"/>
  <cust custid="Cust01" custstoreid="Store01">
   <invoice invid="Inv02" invcustid="Cust01" invstatus="O"/>
   <addr addrid="Addr01" addrcustid="Cust01" addrstate="CA"/>
  </cust>
 </emp>
</root>
```

Node Promotion and Nested View

```
CREATE VIEW EmpCust AS
SELECT *
FROM EmpView LEFT JOIN CustView
        ON EmpCustID=CustID
```

Hierarchical views can be nested.
Views can contain views.

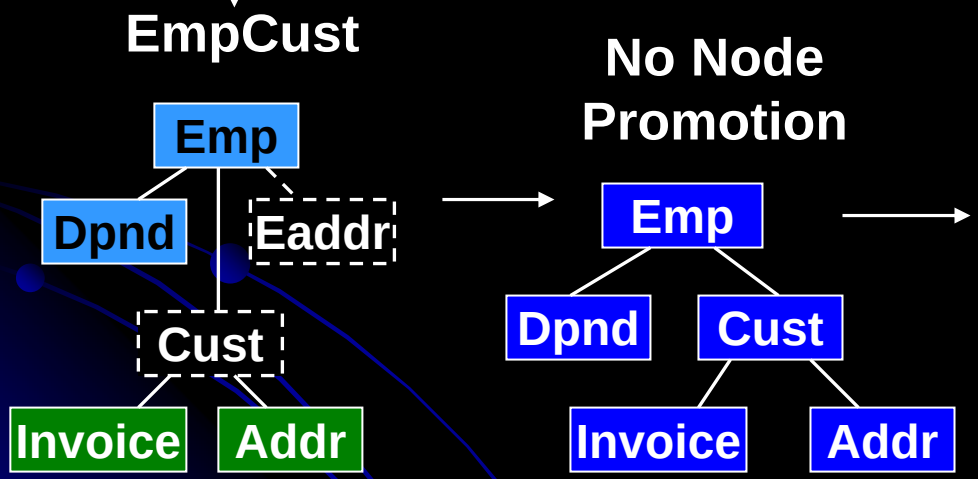
```
SELECT EmpID, DpndID,
        InvID, AddrID
FROM EmpCust
```




```
<root>
 <emp empid="Emp01">
  <dpnd dpndid="Dpnd01"/>
  <invoice invid="Inv01"/>
  <invoice invid="Inv02"/>
  <addr addrid="Addr01"/>
 </emp>
 <emp empid="Emp02">
  <addr addrid="Addr03"/>
 </emp>
</root>
```

Node Promotion Override and XML Format Change

```
SELECT EmpID, DpndID,
       InvID, AddrID
FROM EmpCust
FOR XML ELEMENT KEEP NODE
```



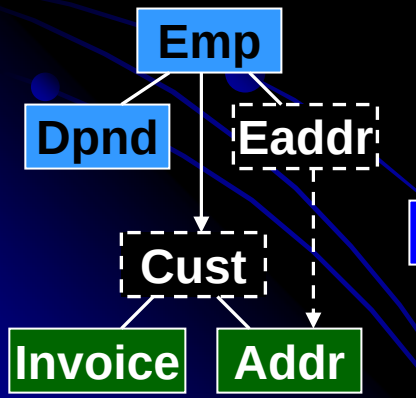
Without node promotion, unselected nodes are output empty. XML output format was changed to Element style.

```
<emp>
 <empid>Emp01</empid>
 <dpnd>
  <dpndid>Dpnd01</dpndid>
 </dpnd>
 <cust>
  <invoice>
   <invid>Inv01</invid>
  </invoice>
  <invoice>
   <invid>Inv02</invid>
  </invoice>
  <addr>
   <addrid>Addr01</addrid>
  </addr>
 </cust>
</emp>
<emp>
 <empid>Emp02</empid>
```

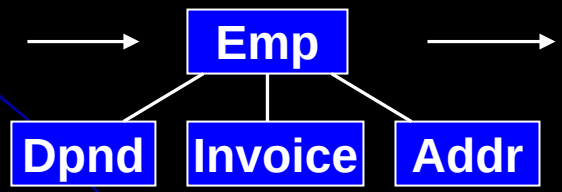
Data Structure Mashup With Node Promotion = Data Virtualization

```
SELECT EmpID, DpndID,
       InvID, AddrID
FROM EmpView
     LEFT JOIN CustView
      ON EaddrID=AddrID
```

Link Below Root



Result Output with Node Promotion




```
<root>
 <emp empid="Emp01">
  <dpnd dpndid="Dpnd01"/>
  <invoice invid="Inv01"/>
  <invoice invid="Inv02"/>
  <addr addrid="Addr01"/>
 </emp>
 <emp empid="Emp02">
  <addr addrid="Addr03"/>
 </emp>
</root>
```

Dashed boxes are unselected, not output.

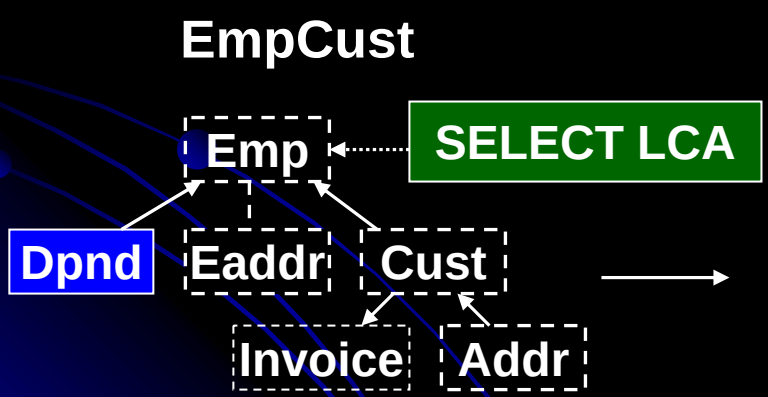
Mashups with node promotion gives aggregated result of the data. This has same effect as data virtualization.

Multipath Query Processing and its Required LCA Processing

- Multipath query references multiple pathways-
  - and uses a WHERE data filtering operation
  - For example: Selecting data based on data in another path
  - This requires a special processing using LCA
- Lowest Common Ancestor (LCA) Node
  - The LCA node is the lowest common ancestor node -
  - Located between two pathway node points in the structure
  - Used to keep the hierarchical query result meaningful
- Two types of SQL multipath LCA usage
  - SELECT with WHERE clause referencing two different paths
  - Compound WHERE clause referencing two different paths
    - This LCA processing solves the XML Keyword Search problem
  - These two uses of LCA can be combined causing nesting
  - Each SELECT data item can have its own LCA

Multipath LCA Query Logic for Single SELECT Data

SELECT DpndID
FROM EmpCust
WHERE AddrID='Addr01'



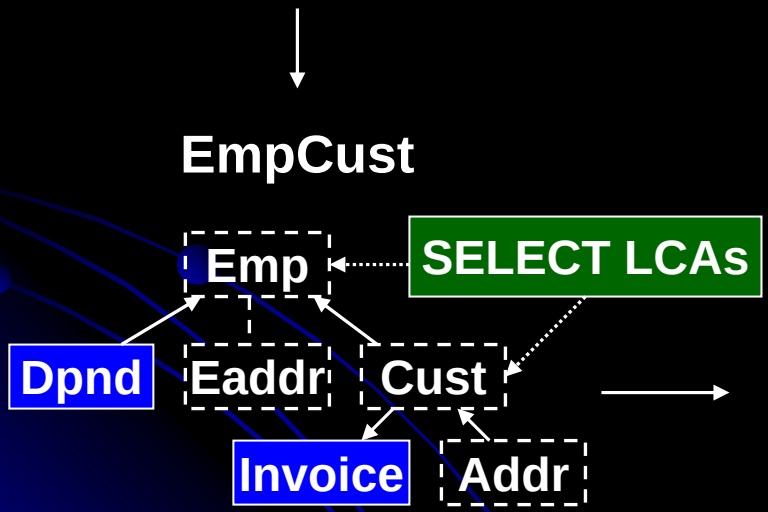
SELECT data types Dpnd and Invoice generate different LCAs.

Lowest Common Ancestor (LCA) processing for the SQL SELECT clause will automatically process multiple LCA nodes when multiple specified output data is located across different pathways.

```
<root>
 <dpnd dpndid="Dpnd01"/>
</root>
```

Multipath LCA Query Logic for Multiple SELECT Data

SELECT DpndID, InvID
FROM EmpCust
WHERE AddrID=’Addr01’



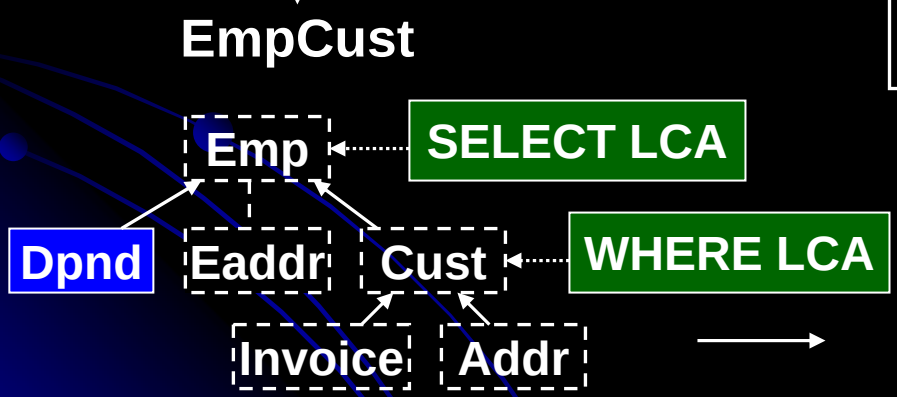
SELECT data types Dpnd and Invoice generate different LCAs.

Lowest Common Ancestor (LCA) processing for the SQL SELECT clause will automatically process multiple LCA nodes when multiple specified output data is located across different pathways.

```
<root>
 <dpnd dpndid="Dpnd01"/>
 <invoice invid=“Inv01"/>
 <invoice invid=“Inv02"/>
</root>
```

Compound WHERE Clauses also Require Their Own LCA

```
SELECT DpndID
FROM EmpCust
WHERE InvID='Inv01'
 AND AddrID='Addr01'
```



This LCA example is actually a combined LCA and nested LCA example

Lowest Common Ancestor (LCA) node processing for multipath queries is necessary. We found it working in SQL naturally on both the SELECT and WHERE operations. Academic projects are currently researching how to add it to XQuery. LCA is the lowest node between node points on separate paths.

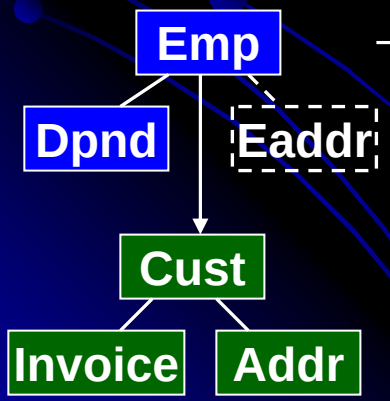
```
<root>
 <dpnd dpndid="Dpnd01"/>
 <dpnd dpndid="Dpnd03"/>
</root>
```

Data Driven Variable Structure Generation

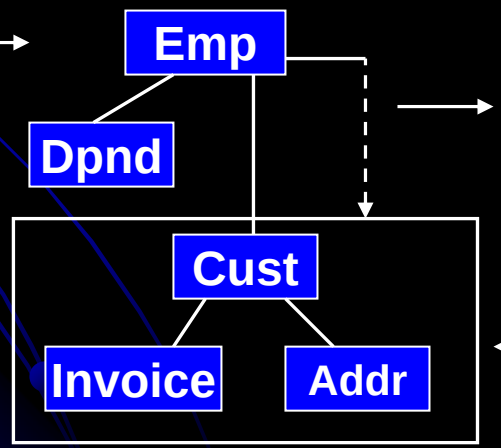
```
SELECT EmpID, DpndID, InvID,
       AddrID, EaddrID,
       CustID, EmpStatus
FROM EmpView
     LEFT JOIN CustView
          ON EmpCustID=CustID
          AND EmpStatus="F"
```

```
<root>
 <emp empid="Emp01"
      empstatus="F">
  <dpnd dpndid="Dpnd01"/>
  <cust custid="Cust01">
   <invoice invid="Inv01"/>
   <invoice invid="Inv02"/>
   <addr addrid="Addr01"/>
  </cust>
 </emp>
 <emp empid="Emp02"
      empstatus="">
 </emp>
</root>
```

Join Structure



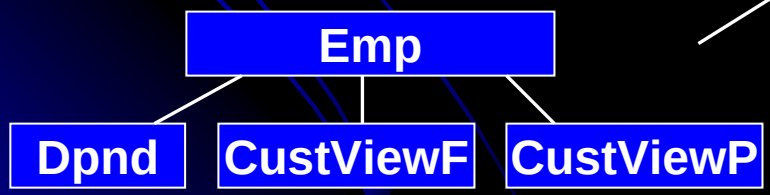
Variable Structure



Current EmpStatus data value controls if this CustView data occurrence expands or not.

Variable Structure Generation Using Multiple Choice

```
SELECT EmpID, DpndID, InvID,
       AddrID, EaddrID,
       CustID, EmpStatus
FROM EmpView
     LEFT JOIN CustViewF
          ON EmpCustID=CustID
          AND EmpStatus="F"
     LEFT JOIN CustViewP
          ON EmpCustID=CustID
          AND EmpStatus="P"
```



Variable Structure

```
<root>
 <emp empid="Emp01"
      empstatus="F">
  <dpnd dpndid="Dpnd01"/>
 Insert Full Time View Here
 </emp>
 <emp empid="Emp02"
      empstatus="P">
 Insert Part Time View Here
 </emp>
</root>
```

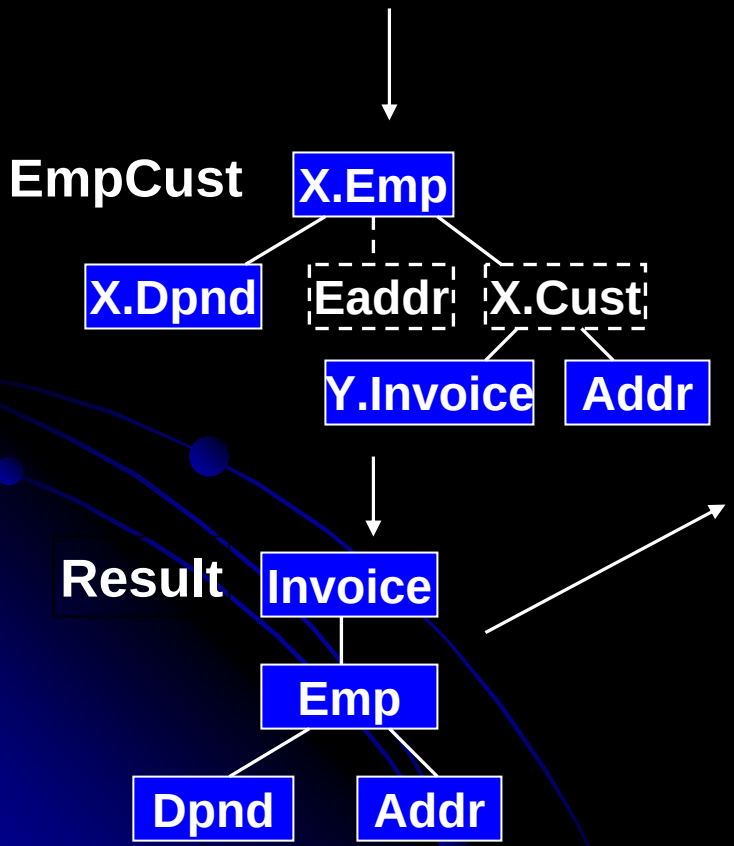
Current EmpStatus data value controls which view expands. Control value is up or down path.

Data Structure Transformation

- Two Basic Types of Data Structure Transformation
  - Restructuring: uses data relationships in the data
  - Restructuring: Uses only the structure semantics in the data
- Restructuring
  - Used to bring out new data relationships
  - Also removes data and nodes
  - New structure is secondary, driven new relationships desired
- Reshaping
  - Used to reshape data structure to a desired new structure
  - No data relationships dependency
  - Any-to-any data structure transformation is possible
  - Reshaping is driven by structures natural semantics
- SQL transformation operation assures accuracy
  - Restructuring and Reshaping operations can be combined

Restructure Transformation

SELECT X.EmpID, X.DpndID, Y.InvID, X.AddrID
FROM EmpCust Y
LEFT JOIN EmpCust X ON Y.InvCustID=X.EmpCustID

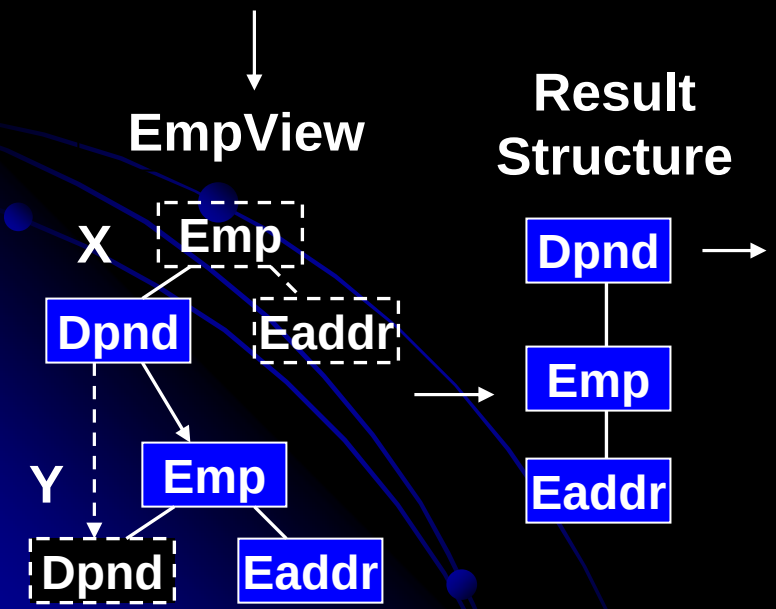



```
<root>
 <invoice invid="Inv01"/>
  <emp empid="Emp01">
   <dpnd dpndid="Dpnd01"/>
   <addr addrid="Addr01"/>
  </emp>
 </Invoice> ....</root>
```

Restructuring SQL transformation:
1) Takes structure apart
2) Disregards unwanted portions
3) Recombines it differently --
4) Uses different data relationships
5) This introduces new semantics

Semantic Any-to-Any Structure Reshaping Transformation

```
SELECT X.DpndID, Y.EmpID, Y.EaddrID
FROM EmpView X
LEFT JOIN EmpView Y
ON X.DpndID=Y.DpndID
```

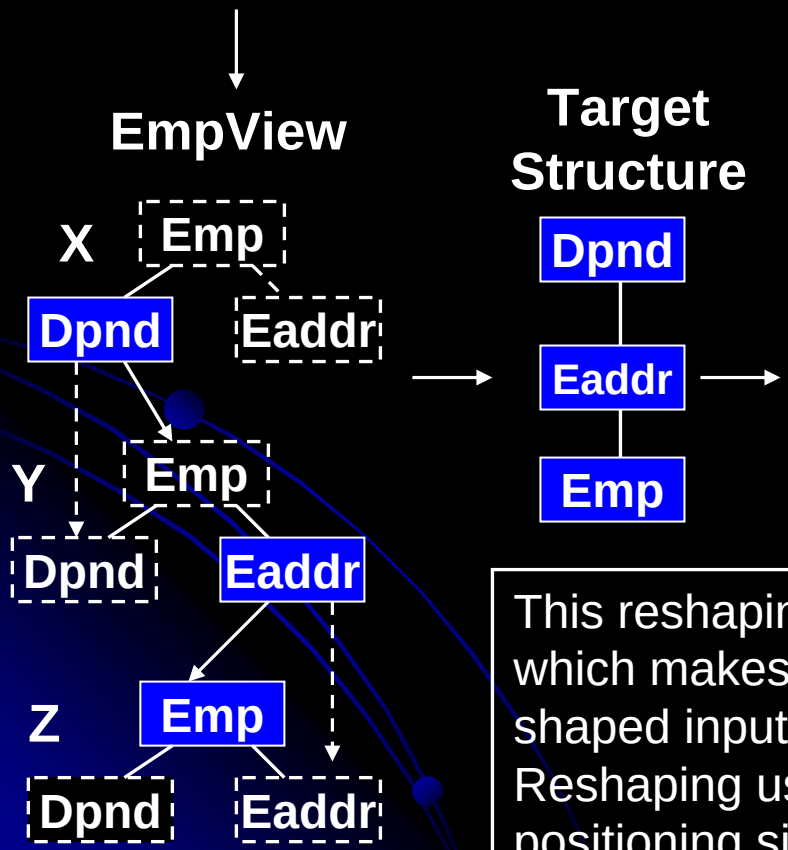



```
<root>
 <dpnd dpndid="Dpnd01">
  <emp empid="Emp01">
   <eaddr eaddrid="Addr01"/>
  </emp>
 </dpnd>
</root>
```

Reshaping uses only the semantics of the data structure to transform it into the desired structure. No new data relationships are needed so this method can always be used. Linking below the root is utilized.

Polymorphic Any-to-Any Reshaping

```
SELECT X.DpndID, Y.EaddrID, Z.EmpID
FROM EmpView X
LEFT JOIN EmpView Y ON  X.DpndID=Y.DpndID
LEFT JOIN EmpView Z ON Y.EaddrID=Z.EaddrID
```

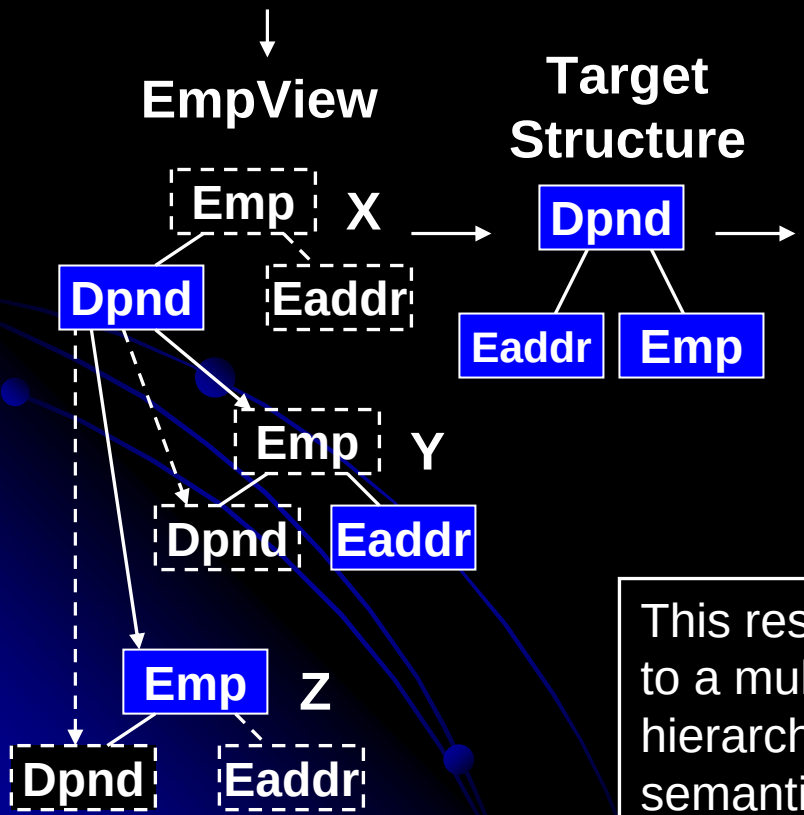



```
<root>
 <dpnd dpndid="Dpnd01">
  <eaddr eaddrid="Addr01">
   <emp empid="Emp01"/>
  </eaddr>
 </dpnd>
</root>
```

This reshaping example moves only one node at a time which makes its operation polymorphic allowing any shaped input structure that uses the same data names. Reshaping uses comparing the structure to itself for positioning since data relationships are not relied on.

Reshaping to Multipath Structure

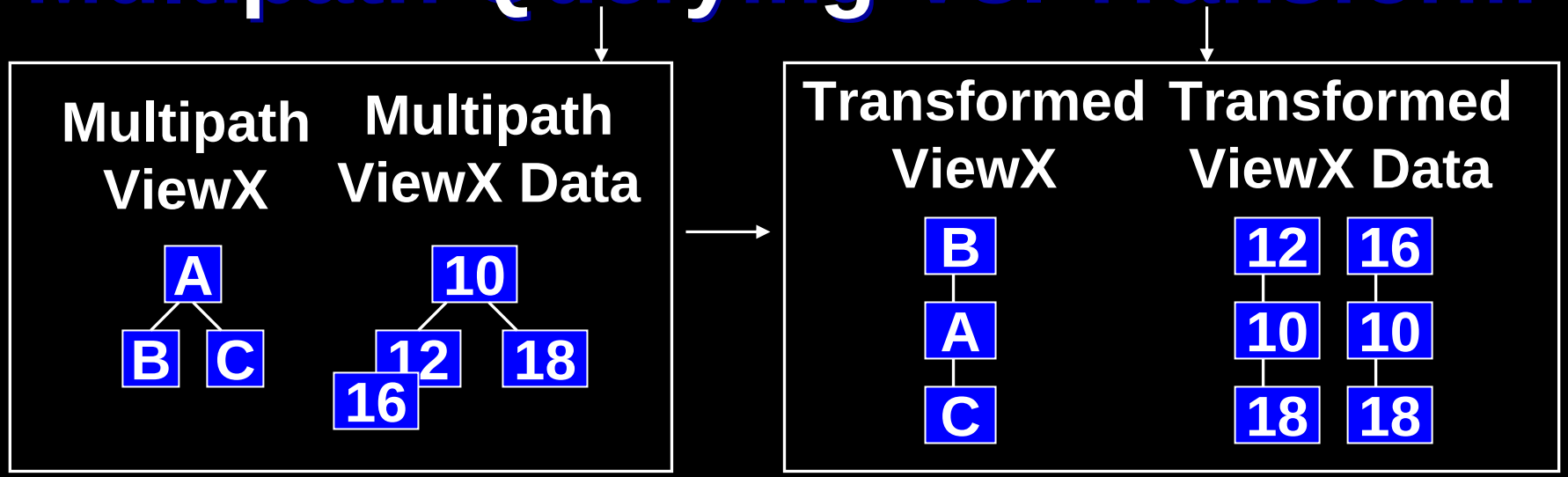
```
SELECT X.DpndID, Y.EaddrID, Z.EmpID
FROM EmpView X
LEFT JOIN EmpView Y ON  X.DpndID=Y.DpndID
LEFT JOIN EmpView Z ON X.DpndID=Z.DpndID
```




```
<root>
 <dpnd dpndid="Dpnd01">
  <eaddr eaddrid="Addr01"/>
  <emp empid="Emp01"/>
 </dpnd>
</root>
```

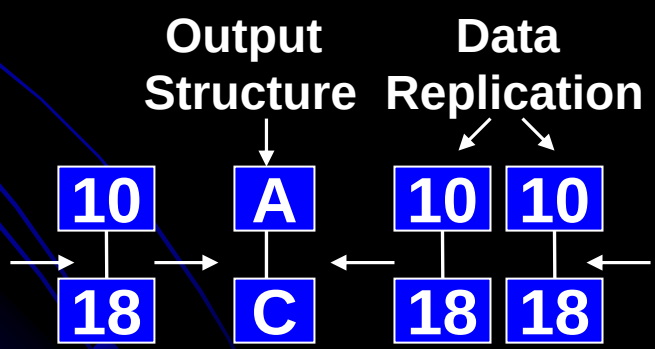
This reshaping example demonstrates reconstructing to a multipath nonlinear structure using SQL's hierarchical data modeling. The SQL hierarchical semantic operation helps preserve correctness.

Multipath Querying Vs. Transform



SELECT A, C
FROM ViewX
WHERE B>10

Multiple use multipath schema-free query uses & preserves data structure



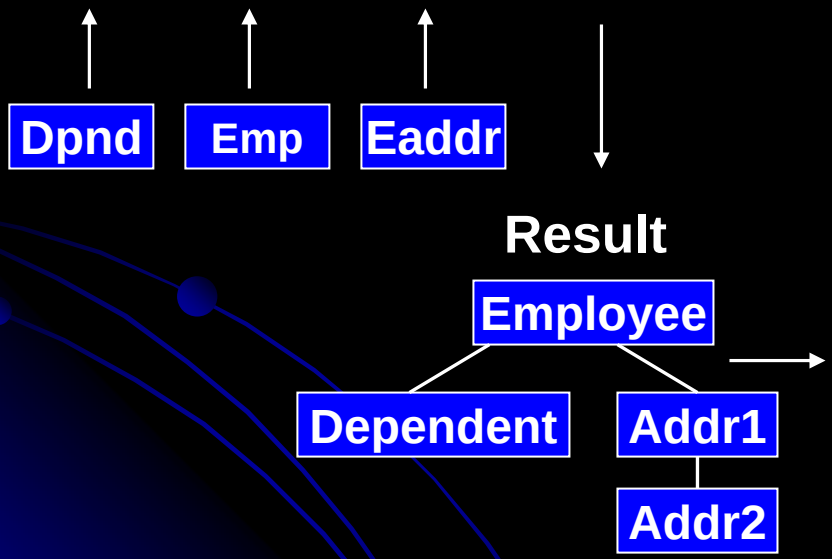
SELECT A, C
FROM ViewX
WHERE B>10

Specifically transforms structure A/B to B/A changing semantics from 1-to-M to M-to-1

A simple multipath query can usually avoid transforms and preserve structure with more accurate results.

Renaming, Replicating, & Splitting Nodes

```
SELECT EmpID EmpName, DpndID AS DpndName,
            Addr1.EaddrID AS AddrName,
            Addr2.Eaddrstate AS AddrState
FROM Emp AS Employee
LEFT JOIN Dpnd AS Dependent ON EmpID=DpndEmpID and DpndCode='D'
LEFT JOIN EAddr AS Addr1 ON EmpCustID=Addr1.EAddrCustID
LEFT JOIN EAddr AS Addr2 ON Addr1.EaddrCustID=Addr2.EAddrCustID
```



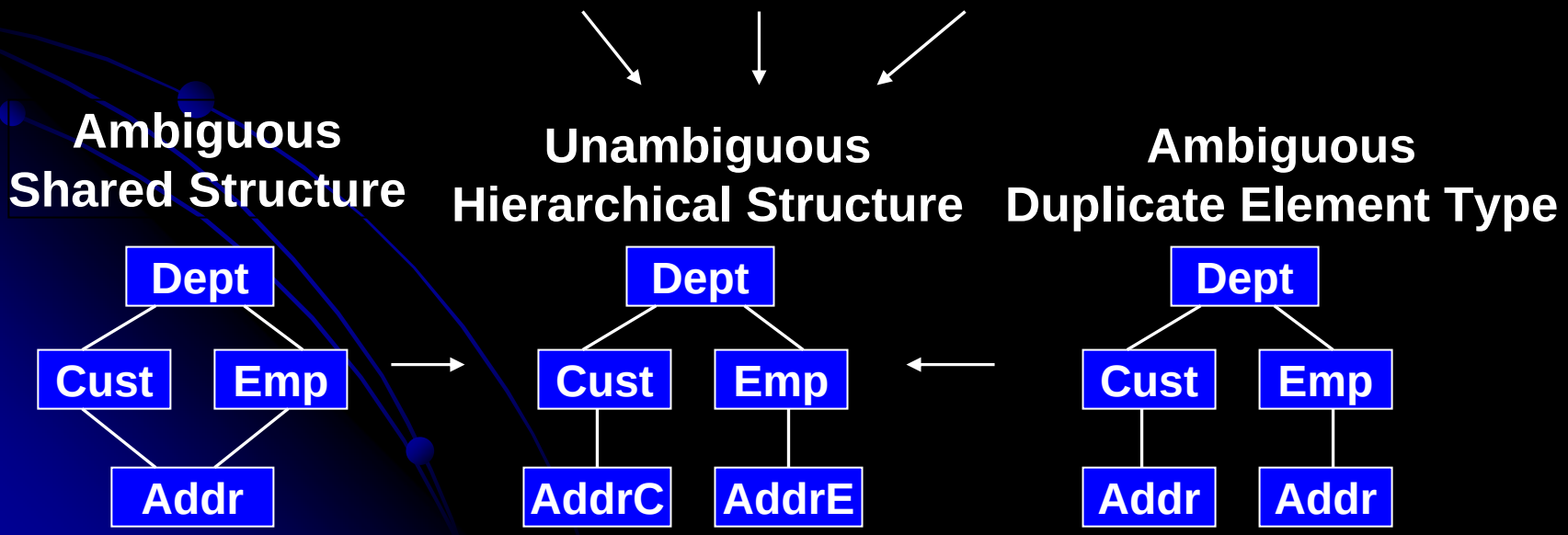
The AS keyword renames XML data items and nodes, it can be used to replicate nodes to help split them.

```
<root>
 <employee empname="Emp01">
  <dependent dpndname="Dpnd01"/>
  <addr1 addrname="Addr01">
   <addr2 addrstate="CA"/>
  </addr1>
 </employee>
 <employee empname="Emp02">
  <addr1 addrname="Addr03">
   <addr2 addrstate="NV"/>
  </addr1>
 </employee>
</root>
```

XML Duplicate and Shared Unambiguous Node Processing

```
SELECT * FROM Dept
LEFT JOIN Cust ON DeptID=CustDeptID
LEFT JOIN Emp ON DeptID=EmpDeptID
LEFT JOIN Addr AS AddrC ON CustID=AddrCustID
LEFT JOIN Addr AS AddrE ON EmpID=AddrEmpID
```

This same SQL handles both duplicate and shared data.



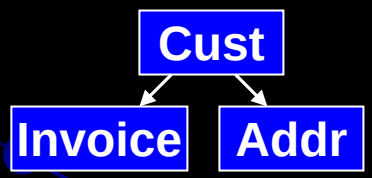
Nonlinear Hierarchical ORDER BY

```
SELECT CustID, InvID, AddrID
FROM CustView
ORDER BY AddrID Desc,
         InvID Desc,
         CustID Desc
```

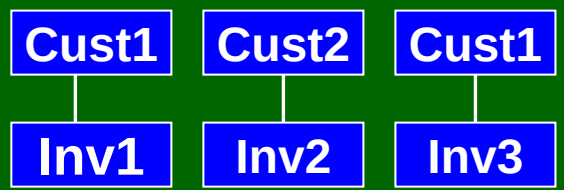
→

```
<root>
 <cust custid="Cust03">
  <addr addrid="Addr03"/>
 </cust>
 <cust custid="Cust02">
  <invoice invid="Inv03"/>
  <addr addrid="Addr04"/>
  <addr addrid="Addr02"/>
 </cust>
 <cust custid="Cust01">
```

CustView Ordering



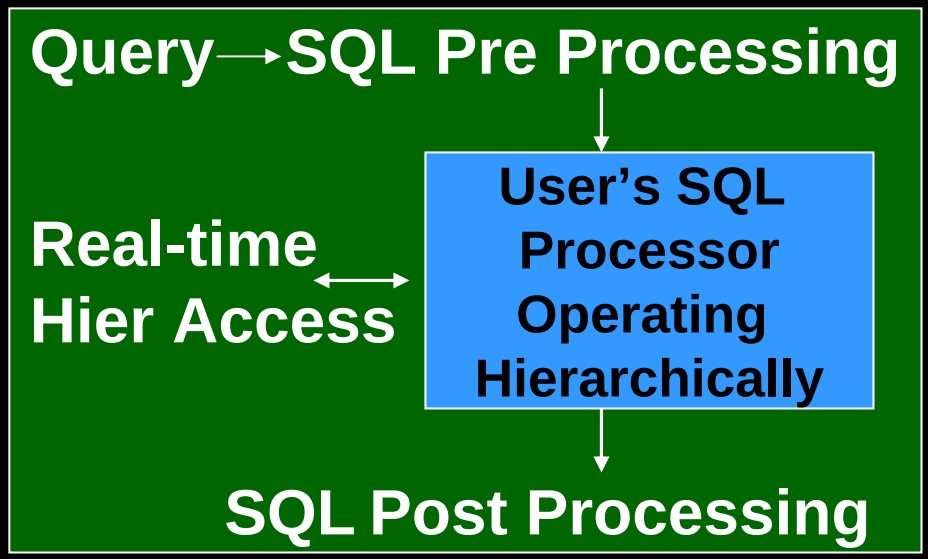
Ordering Inv before Cust is Trouble:



Cust becomes out-of-order

Hierarchical ops like ORDER BY require special nonlinear hierarchical processing to make sense for SQL Hierarchical processing. With Order By, each path is ordered independently as in XML above.

Hier & XML Middleware Enabler



This Unique Implementation:

- Supports XML enables hier processing
- Uses in place SQL processor
- Needs no restaging of data
- Not XML centric, no learning curve
- ANSI SQL-92, vendor neutral
- Operates seamlessly & transparently
- Protects current SQL investment

Before Query:

- Establish Views
- Preload XML (ETL)

SQL Pre Processing:

- Determines structure
- Optimizes structure
- Rewrite & submit SQL

Real-time Access:

- Structure-aware
- Retrieve XML (EII)
- Return XML as rowset
- Retain XML data order

SQL Post processing:

- Remove replicated data
- Nonlinear data ordering
- Rowset to XML

Dynamic Data Structure Creation Recap

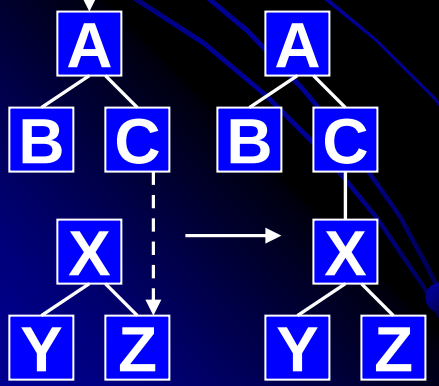
- SELECTed Data Output
  - Controls processing and output result structure
  - With automatic node promotion and collection
- Combining Data Structures
  - Joining and mashup of heterogeneous structures
  - Mashup & node promotion = data virtualization
- Generating Variable Data Structures
  - Multiple choice of view structure generation
  - Driven by data value further up or down the path
- Transforming Data Structures
  - Restructuring using data relationships
  - Reshaping using structure semantics
  - Node splitting and renaming

The Power of Hierarchical LEFT Outer Join Syntax Recap

```
ViewA LEFT JOIN ViewX
ON C.c=Z.c AND A.a=10
```

Views Expanded

```
A LEFT JOIN B ON A.a=B.a
   LEFT JOIN C ON A.a=C.a
LEFT JOIN
X LEFT JOIN Y ON X.x=Y.y
   LEFT JOIN Z ON X.x=Z.z
ON C.c=Z.c AND A.a=10
```

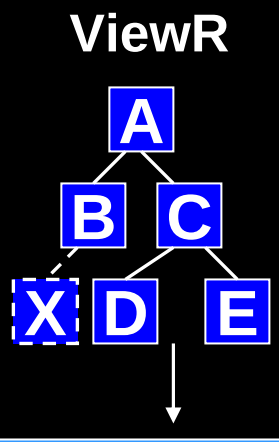


Outer join models join of views, semantics define result structure.

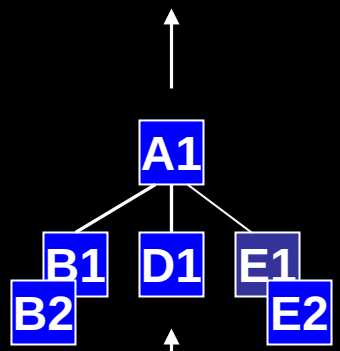
Shown on this slide:

- Automatically Combines and Unifies Heterogeneous Structures
- Path Filtering Based Up/Down Path
- Directly Executable by SQL Engine
- Flexible Recombinant Properties
- Referencing Below Root is valid
- Access optimization: paths sliced out
  - Processing optimization: reorganized
  - Naturally Hierarchically Distributable

Combining Basic Capabilities



Hierarchical XML Result




```
CREATE View ViewR AS
SELECT * FROM A
LEFT JOIN B ON A.a=B.a
LEFT JOIN C ON A.a=C.a
LEFT JOIN D ON C.c=D.c
LEFT JOIN E ON C.c=E.c
```

A1	B1	D1	E1
A1	B2		E2

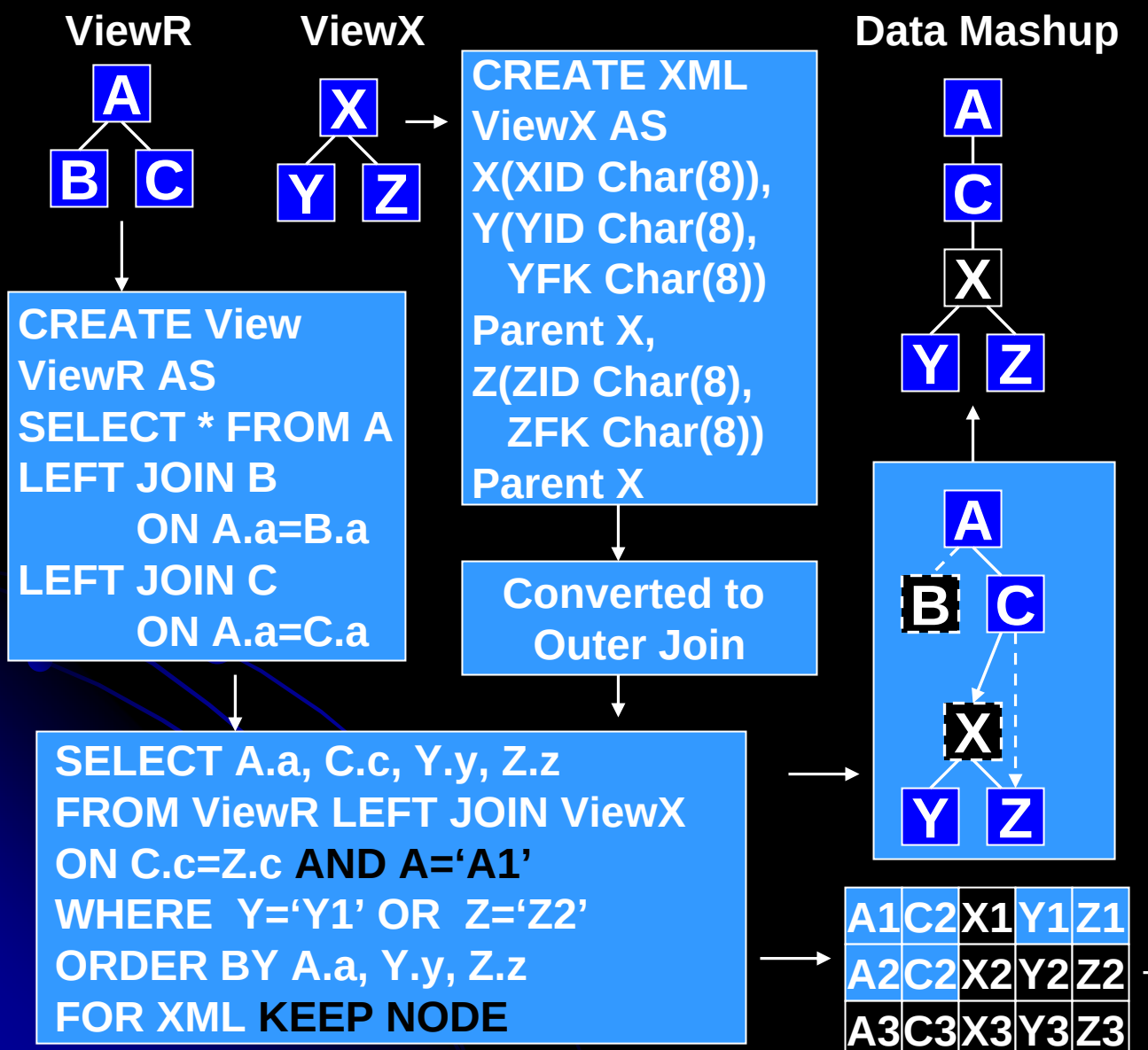
A1	B1	C2	D1	E1
A1		C1	D1	
A1	B2	C2		E2

```
SELECT A.a,B.b,D.d,E.e
FROM ViewR
WHERE  C='C2'
```

Examples:

- Hier View Support
- Data Modeling
- Hier. Preservation
- Var. Length Paths
- Multiple Paths
- Multi-node Types
- Dyn. Data Select
- Hier. Data Filtering
- Node Promotion
- Node Collection
- Optimized Access
- Global View
- Schema-free

Combining Advanced Capabilities



Examples:

- XML View
- Heterogeneous Data Integration
- Mashup
- Var. Structures
- LCA Processing
- Linear Filtering
- Hier. Ordering
- Preserve Nodes
- Unified View
- XML Keyword Search

Advanced Capabilities 1

- **Multipath nonlinear processing**
  - Dynamic increase of data value using structure semantics
  - Processes all queries regardless of internal complexity
  - Hierarchical data filtering-- XML Keyword Search
- **Multipath hierarchical data structure joins**
  - Performed by simple join of hierarchical structure views
  - Can be performed interactively and heterogeneously
  - Dynamically combines hierarchical structure semantics
- **Linking below root allows structure mashup**
  - Enables capability to mashup structures meaningfully
  - Includes powerful look-back and look-ahead capability
  - Mashup + node promotion = data virtualization
  - Enables SQL Transformations

Advanced Capabilities 2

- **Dynamic automatic variable structure generation**
  - Dynamically builds structures based on values in data
  - Can utilize cascading and embedded view operation
  - Operates across multiple paths independently
- **Dynamic user control of structure**
  - Dynamic SELECT list controls processing structure
  - Dynamic combining views builds data structure
  - Transformations Change Structure
- **Dynamic Structured Output Formatting**
  - Structure-aware processing knows active structure
  - Uses structure of result to format output data
  - Active structure is dynamically controlled by user

Advanced Capabilities 3

- **Polymorphic any-to-any structure transform**
  - Uses data relationships or just structure semantics
  - Utilizes SQL and hierarchical rowset data
  - Two types of transforms, Restructure and Reshaping
- **Multipath nonlinear hierarchical Operations**
  - Single Order By orders multiple pathways separately
  - Aggregation operates on separate pathways
- **Multipath internal LCA nonlinear processing**
  - LCA processing limits the domain across pathways
  - Is automatic in ANSI SQL, but not in XQuery
  - Responsible for keeping multipath result meaningful
- **Untested natural and automatic operations**
  - Natural hierarchical distributed processing
  - Automatic parallel processing is possible
  - Semantic web RDF to SQL, SQLfX driven by RDF

Advanced Capabilities Summary

1) ANSI SQL standard and mathematically sound
2) Ease of use (nonprocedural, navigationless, schema-free)
3) Hierarchically correct (principled multipath processing)
4) Greater efficiency (hierarchical processing optimization)
5) Fully interactive (dynamically process native XML)
6) Conceptual hierarchical processing on full structures
7) Queries can operate across the entire hierarchical structure
8) Nonlinear multipath LCA hierarchical processing
9) Full nonlinear hierarchical data structure mashups
A) Variable data generated structure control
B) Any-to-any polymorphic structure transformations
C) All operations are semantically controlled and accurate
D) Data virtualization supported
E) Natural distributed hierarchical processing
F) Automatic parallel processing is possible

All of the capabilities shown in this presentation can be reproduced on our online interactive demo at www.adatinc.com/demo.html