Slides: How ANSI SQL Inherently Utilizes Natural Structure Semantics To Naturally Perform Nonlinear Hierarchical Processing

Michael M David
Advanced Data Access Technologies. Inc.
Santa Monica, California
www.adatinc.com

Contents

Prefix

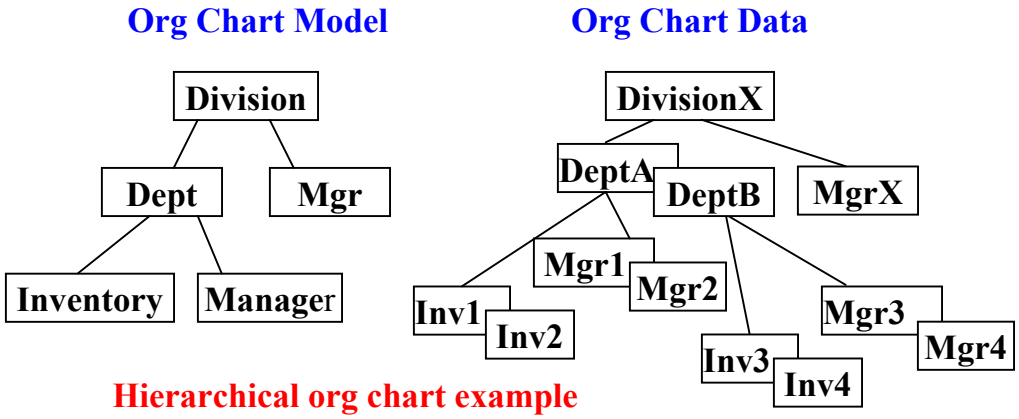
What is problem with XML and hierarchical processing today

- **XML spec does not specify how XML hierarchical data is to be processed**
- **Is not based fully on any solid principled technology**
- **Is not standardized**
- **Limited to linear processing**
- **Requires procedural processing and navigation**
- **Does not support ad hoc processing**

Nonlinear hierarchical processing is the hierarchical processing solution

- **This solution is not new**
- **This solution is already proven**
- **This solution is based on solid hierarchical principles**
- **This solution supports non procedural processing**
- **This solution naturally follows ANSI SQL Hierarchical Processing**

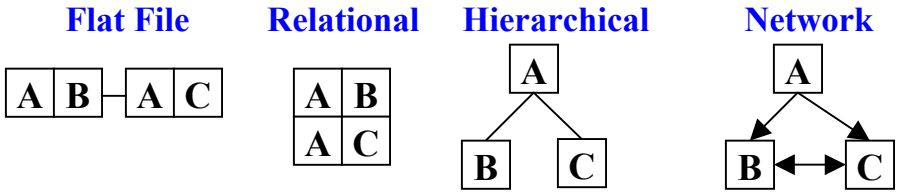
1) Introduction: Nonlinear Hierarchical Structures and their Processing



Hierarchical org chart example

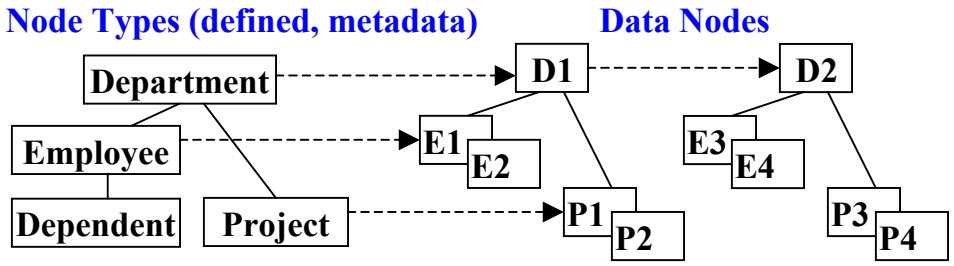
- **Hierarchical structures useful for organizing data meaningfully & efficiently**
- **No principled foundation is being used with hierarchical processing today**
- **Today's basic hierarchical processing limited to single leg linear processing**
- **Multi-leg nonlinear processing based on solid intuitive hierarchical principles**
- **Nonlinear hierarchical processing processes full structure as a whole**
- **Nonprocedural processing possible utilizing semantics between every node**
- **Users can request complex multi-leg queries w/o knowing structure**

2) Review of Data Structures
2.1) Types of Data Structures



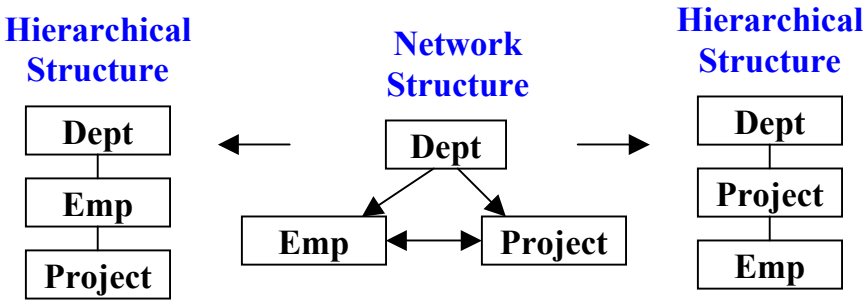
- **Flat files most basic, composed of sequential records of contiguous data fields**
- **Relational most flexible with data independence, comprised of flat tables**
- **Hierarchical & network structures composed of nodes of contiguous fields**
- **Hierarchical tree structure more specific, contains most inherent semantics**
- **Network are graph structures, are flexible but require procedural navigation**
- **Each box represents tightly related data grouped together and is normalized**

2.2 Node Types and Data Nodes, and their relationships



- **Hierarchical structures have a single unambiguous meaning**
- **Single path to each node, can be navigated nonprocedurally**
- **Node relations: root, child, parent, ancestor, descendent, sibling, twin, cousin**
- **Twins not part of structure, siblings are, they have very different semantics**
- **Node Types & Data Nodes use same relationships (parent, child), but different**
- **Node Types resemble runtime control block that vectors data node structure**
- **A root data occurrence and its descendents make a record, or XML document**

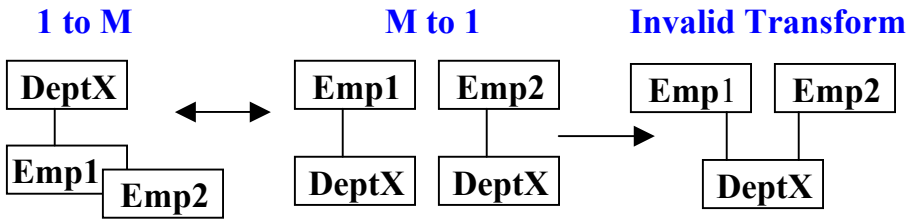
2.3) Hierarchical Vs Network Structures



Network structures can represent multiple hierarchical structures

- **Network structures have interlinked nodes, can represent multiple structure**
- **Do not always represent unambiguous structure, have multiple paths to node**
- **Each node path has own semantics, path taken establishes semantic meaning**
- **Network require procedural navigation, have multiple paths to same node**
- **Unlike hierarchical structures, there is no single parent-child relationship**
- **Networks have no hierarchical data preservation, i.e. no cascading deletes**
- **Converting network structures to hierarchical is possible**

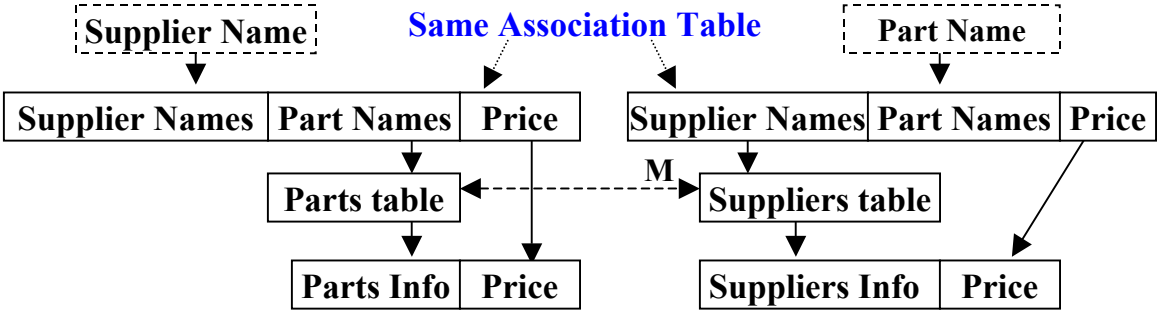
2.4) Entity Relationships Review
2.4.1) One to Many and Many to One Relationships



M to 1 and 1 to M representing the same data differently

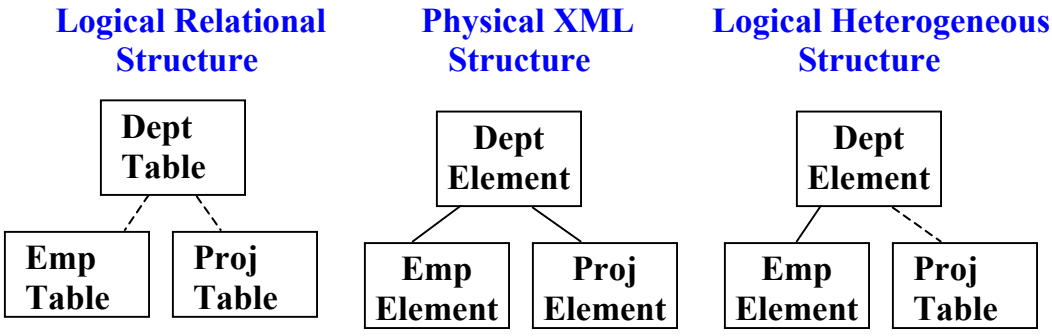
- **Most common entity relationships are one-to-many (1 to M)**
- **1 to M is a good example of twins on the M side**
- **M to 1 is the natural opposite of 1 to M relationship**
- **Depending on how stored, data can be processed with both relationships**
- **Each entity relationship type has its own semantics and they are different**
- **Data replication (DeptX) caused by M to 1 is meaningful and necessary**
- **Sharing DeptX by Emps is invalid, need independents, i.e. Order By Emp**

2.4.2 Many to Many Relationships



- **M to M are naturally occurring relationships (i.e. Supplier <--> Parts above)**
- **Many-to-many entity relationships are not usually naturally occurring data**
- **They naturally occur as two 1 to M relationships comprised of the same data**
- **The M to M data relationship can be combined into an Association table**
- **Association table useful for many reasons such as reuse and maintenance**
- **Association table offers opportunity to add value with intersecting data**
- **Intersecting data (Price), can be placed in lower node (the M of 1 to M)**

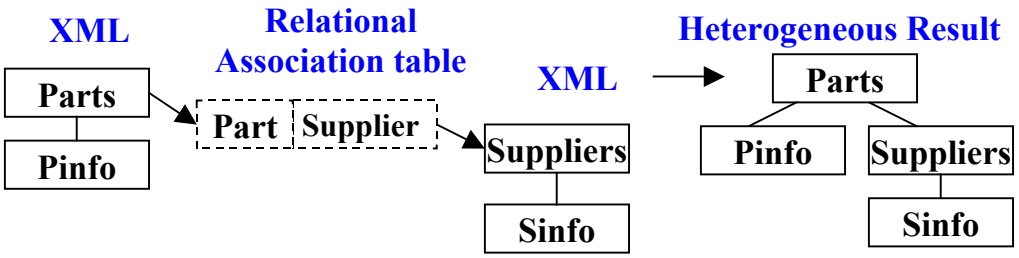
2.5) Logical and Physical Hierarchical Structures



Logical and physical hierarchical structures have same semantics

- **Physical structures like XML usually fixed and can not be easily changed**
- **Logical structures like relational tables are free to change their structure**
- **Logical structures can include physical structures combined in many ways**
- **These logical and fixed structure combinations are heterogeneous**
- **Heterogeneous structures operate consistently because all hierarchical**
- **Hierarchical structures have rigid semantics, requires specific processing**

2.5.1) Heterogeneous Association Table Use

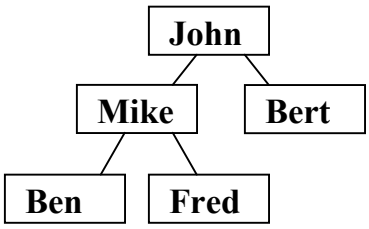


- **Example of a heterogeneous structure using an M to M association table**
- **Three choices in association table processing:**
  - **Association table can be transparent to the result**
  - **Association table can left in place if there is intersecting data**
  - **Association table can be transparent and intersecting data moved down**
- **Better approach then maintaining two XML structures and data**
- **Allows logical and physical structures to be used to their best advantage**

2.6) Hierarchical Data Structure History

- **Fixed hierarchical structures where used in the first database systems**
- **Relational DBs introduced data independence and mathematical soundness**
- **For this reason, relational databases replaced fixed hierarchical databases**
- **Ignored: logical hierarchical structures can be built with physical structures**
- **Logical hierarchical structures still have physical hierarchical capabilities**
- **The advent of XML has re-introduced hierarchical structures again**
- **Nonprocedural nonlinear hierarchical processing ignored or forgotten**

2.7) Single Vs Multiple Node Type Hierarchical Processing



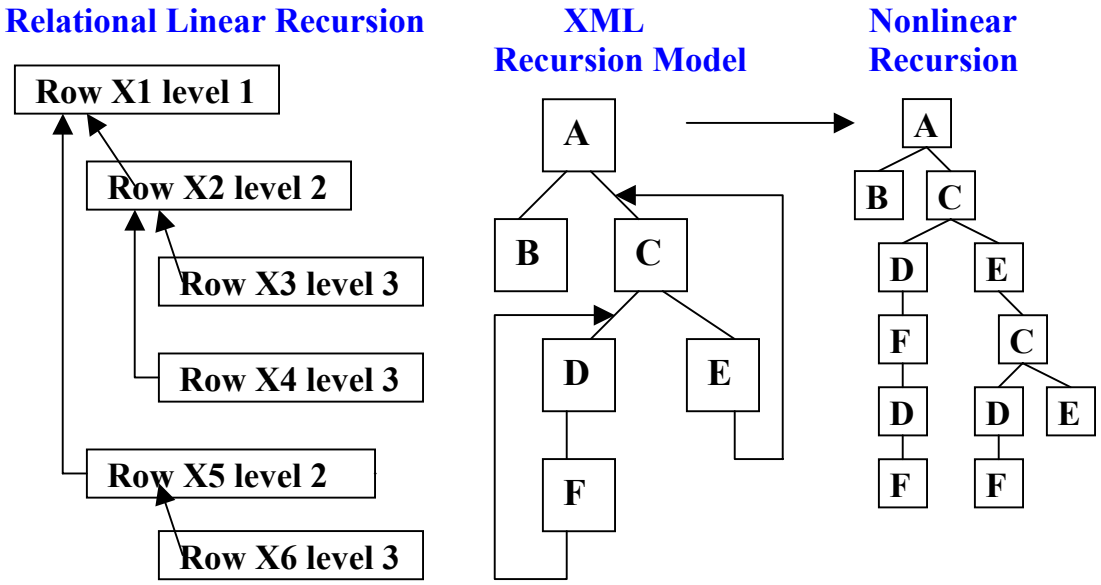
Adjacency List

Offset	Data	Parent	Level
1	John	0	1
2	Mike	1	2
3	Ben	2	3
4	Fred	2	3
5	Bert	1	2

Single node Adjacency List model hierarchical structure

- **Hierarchical SQL processing today is done externally using an adjacency list**
- **Requires building hierarchical processing functions on top of SQL's functions**
- **Adjacency list method supports a single node type system with no twin support**
- **Single node system with no twin support use recursion to form hierarchy**
- **External databases limited but useful, Joe Celko wrote an entire book on them**
- **External single node databases can't support nonlinear structures like XML's**
- **Multiple node type systems support nonlinear hierarchical processing for XML**

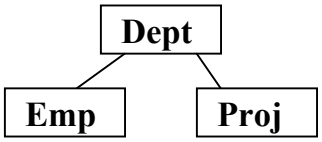
2.8) Recursive Structures and Techniques



- **SQL supports a linear row recursion as shown above using its recursive union**
- **XML Structure supports multiple node types like A, B,C, D, E, and F above**
- **XML structure definition support complex nonlinear hierarchical recursion**
- **XML supports complex recursions, but there is challenge of navigating them**

3) Hierarchical 4GL Capabilities

DeptView



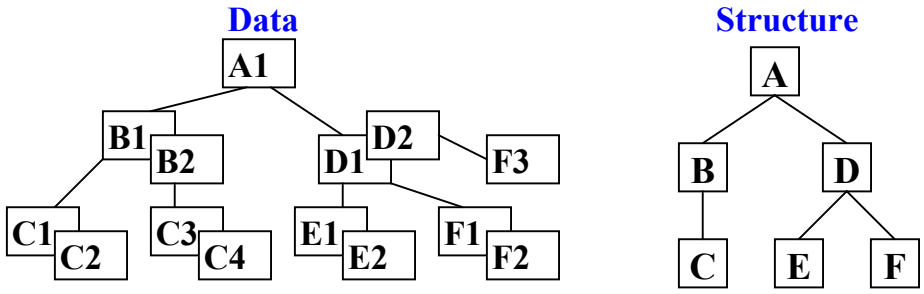
Nonlinear hierarchical query

4GL type query: Display all employees for departments having a Project =”X”

SQL: SELECT Emp FROM DeptView WHERE Proj=”X”

- **4GLs are nonprocedural languages also known as declarative languages**
- **4GLs specify what to do, not how, use no procedural processing or navigation**
- **4GLs need to know the data structure, so the user does not have to**
- **Complex multi-leg queries not practical can be specified dynamically by 4GLs**
- **Semantically complex 4GL queries specified easily and eliminate logic errors**
- **Automatic use of semantics by 4GLs dynamically increases value of the data**

3.1) Hierarchical Semantics with Multiple Data Occurrences



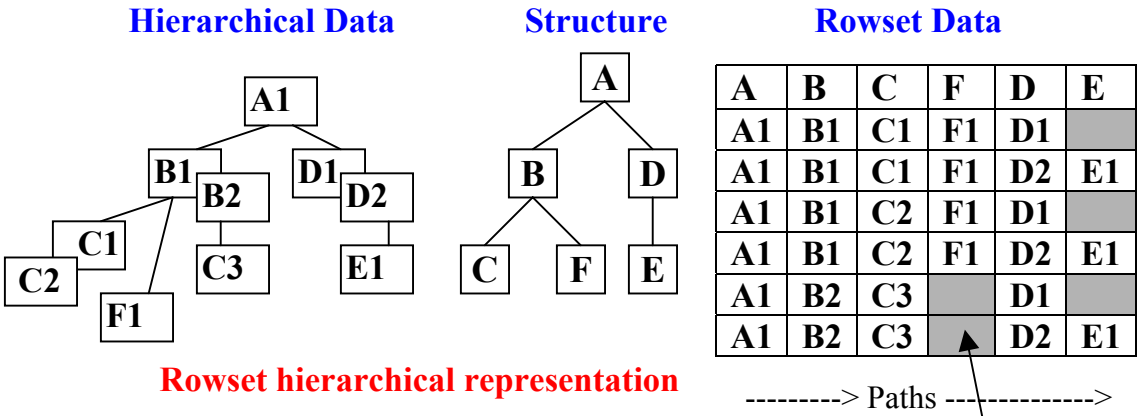
- **Twin support needed for complex hierarchical data structures like XML**
- **Twin data increases hierarchical processing complexity considerably**
- **Siblings are part of structure, twins are not, this is why twin use overlooked**
- **Twins related by parent data occurrence, C1&C2 related and C1&C3 are not**
- **Twins in other siblings related by Lowest Common Ancestor (LCA) node**
- **LCA data occurrence controls processing between legs for meaningful result**
- **LCA indicates that E1 & F2 related by LCA D1, and E1 & F3 are not**

3.2) Hierarchical SQL Opportunity

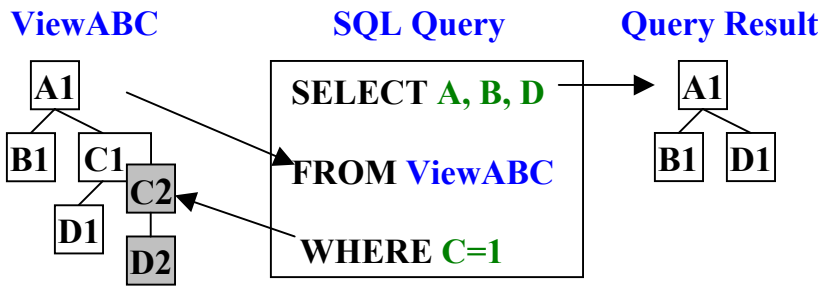
Flexible Outer Join data modeling examples

- **SQL-92 can model hierarchical structures directly in executable SQL**
- **Data modeling SQL when executed performs nonlinear hierarchical processing**
- **Left Outer join preserve data nodes (tables, XML elements) hierarchically**
- **ON clause at each join point specifies where data nodes are linked in structure**
- **SQL hierarchical processing means processing is subset of relational processing**
- **This means they share relational processing's mathematical foundation**
- **Automatic use of XML hierarchical semantics goldmine has staggering value**

4) ANSI SQL as a Hierarchical 4GL Query Processor



4.1) Hierarchical Query Specification and Operation



- **SQL nonprocedural nonlinear hierarchical query processes operates seamlessly**
- **SQL's standard SELECT FROM WHERE offers full hierarchical control**
- **SELECT specifies data fields (table columns, XML attributes) to be returned**
- **FROM specifies input nodes (tables, XML elements) and hierarchical structure**
- **WHERE specifies data filtering hierarchically applied to input structure**
- **Seamless hierarchical view use adds significantly to the ease of use**

4.2) FROM Clause Controls Hierarchical Data Modeling

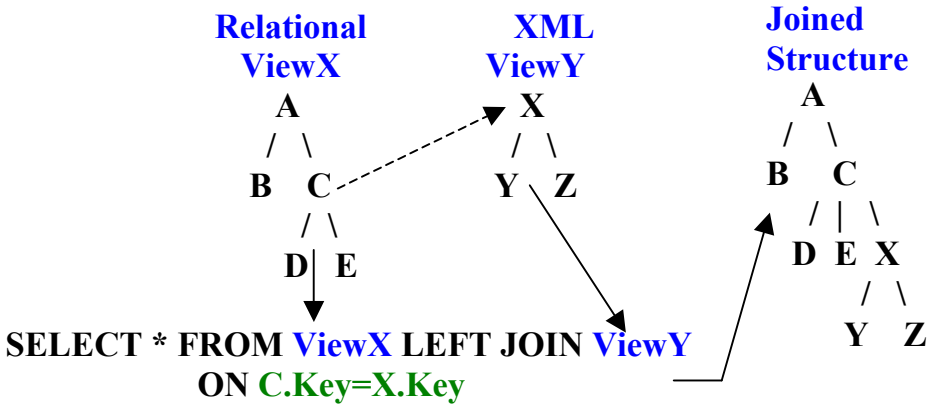
```
ViewX       CREATE VIEW ViewX AS
  A         SELECT * FROM A
 / \        LEFT JOIN B ON A.a=B.a
B   C       LEFT JOIN C ON A.a=C.a
   / \      LEFT JOIN D ON C.c=D.d
  D  E      LEFT JOIN E ON C.c=E.c
```

To more easily understand the data modeling, read "LEFT JOIN" as "over" and "ON" as "linked by".

SQL hierarchical data view

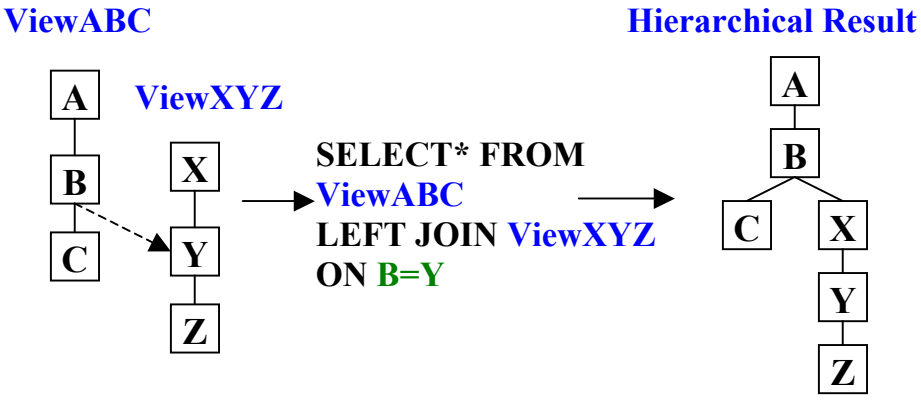
- **Hierarchical data modeled structures can be stored separately in SQL views**
- **Left Outer Join preserves left node over right node**
- **The ON clause unambiguously specifies the link points of the Left Outer Join**
- **WHERE clause does not work well to link structures, not very specific**
- **ON clause can specify formation of a new leg or appending to an existing leg**

4.2.1) Joining Hierarchical Structures Conceptually



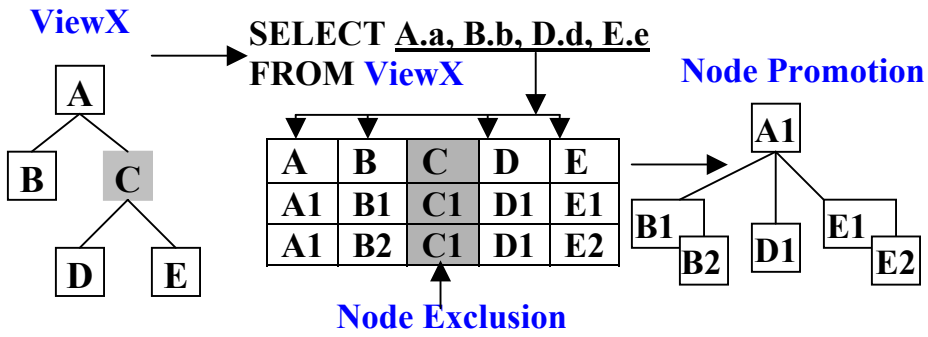
- **Combines hierarchical structures at high conceptual level using views**
- **Views manipulated same as any node type (SQL Table, XML element)**
- **Combined structures preserves and enhances hierarchical semantics**
- **ON clause specifies link points in both structures being joined**
- **Logical & physical views can be specified, they have same semantics**
- **Dynamic operation for structure joining and all hierarchical operations**

4.2.2) Linking Below the Lower Level Structure's Root



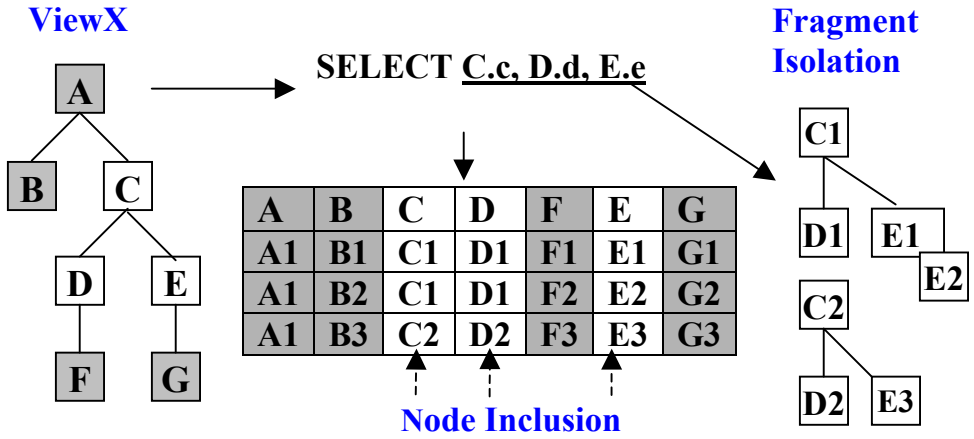
- **Linking below the root of the lower level structure is possible**
- **It is semantically sound and there are no standard usage restrictions**
- **The semantics always makes the root the lower structure link point**
- **Filtering occurs at specified link point as you would expect**
- **This hierarchical linking capability is very user friendly**
- **Supports unrestricted joins, users do not need to know the structure**

4.3) SELECT Clause Controls Node Promotion and Collection



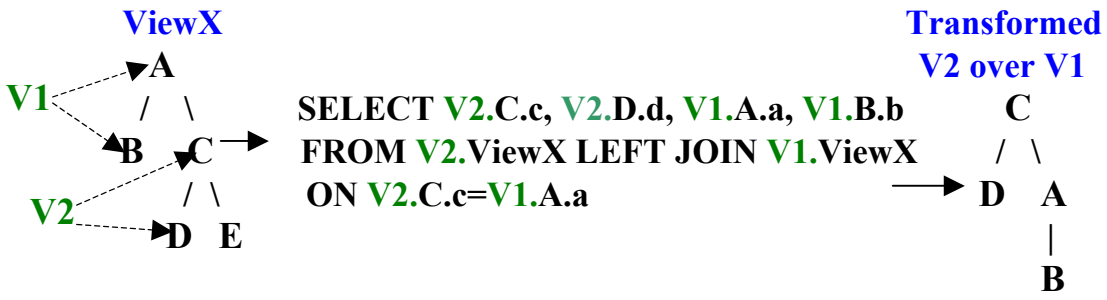
- **Node promotion is a standard hierarchical operation**
- **Controlled easily in SQL by which data is not SELECTed**
- **Unselected nodes are sliced out of the output structure**
- **SQL hierarchical processing supports this capability naturally**
- **As seen in this example, this is a vertical type of filtering**

4.3.1) Fragment Isolation Using Node Promotion



- **Fragment isolation is standard XML hierarchical operation**
- **Nonprocedurally controlled by what data is SELECTed**
- **Unselected nodes are sliced out from output structure**
- **Node promotion is utilized automatically within fragments**
- **SQL hierarchical processing supports this capability naturally**

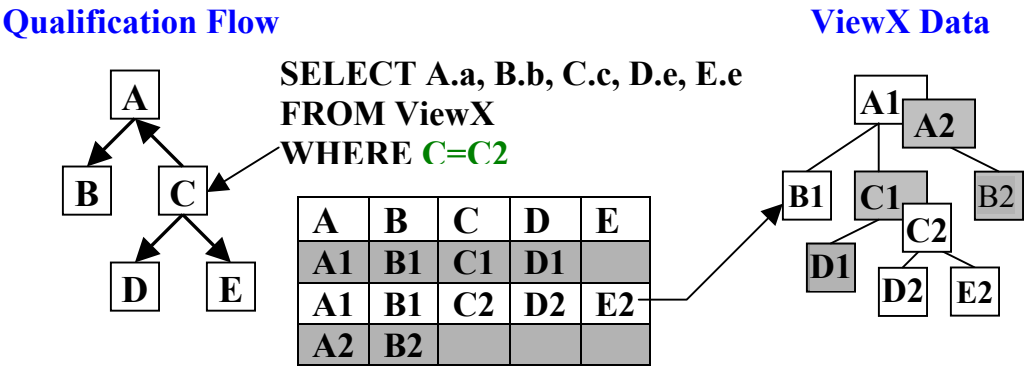
4.3.2) Structure Transformation Using Fragment Isolation



Structure transforming using single Structure

- **This structure transformation transforms a single structure**
- **It separately isolates two different fragments from the same structure**
- **This requires prefix use (V1 and V2) to differentiate the two fragments**
- **Fragments joined differently than original structure, transform structure**
- **Nonprocedural high level conceptual hierarchical structure transform**
- **Transforming with single fragments from multiple views is less complex**

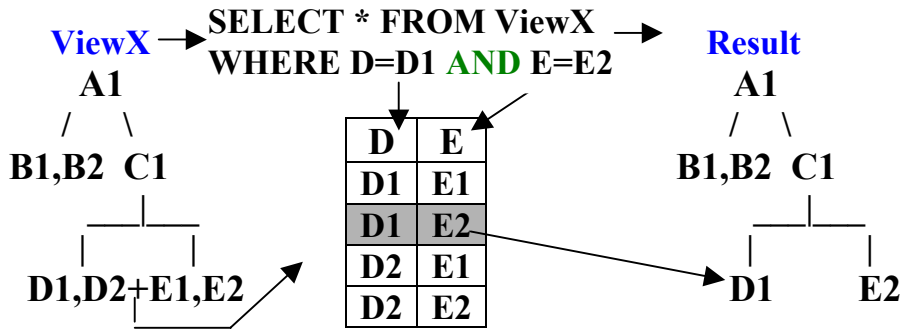
4.4) WHERE Clause Controls Hierarchical Data Filtering
4.4.1) Hierarchical Data Qualification



A	B	C	D	E
A1	B1	C1	D1	
A1	B1	C2	D2	E2
A2	B2			

- **WHERE clause data filtering operates on the entire hierarchical structure**
- **Easier to perceive this operation as qualifying data rather than filtering data**
- **WHERE clause qualifies its referenced nodes (C2 with WHERE C=C2)**
- **Related path occurrence above qualified node (C2) is qualified (A1)**
- **Related path occurrences under qualified node (C2) are qualified (D2 & E2)**
- **Related paths under a cousin node occurrence are qualified (B1), LCA logic**

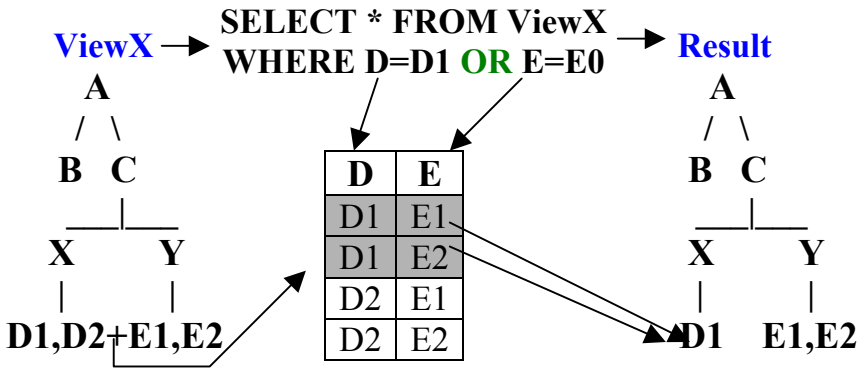
4.4.2) Multi-leg Filtering Semantics with AND Logic



AND filtering across legs

- **AND logic requires both sides of AND operation to be true**
- **Cartesian product generates combinations of sibling legs**
- **Combination of sibling legs simulates LCA logic**
- **SQL qualification on a single row by row tests all combinations**
- **WHERE clause with Compound LCA will be covered later**

4.4.3) Multi-leg Variable Semantics with OR Logic



Variable semantics for OR filtering across legs

- **OR logic used in hierarchical query selection requires both sides to be tested**
- **Left, right, or both sides true, each generate different semantics and result**
- **If D1 is true and E0 is not, D1 qualifies itself and E1 & E2 using LCA logic**
- **LCA logic generates combinations of sibling legs to simulate proper processing**
- **SQL processing and qualification works row by row by testing all combinations**

4.4.4) Single Path Data Filtering (Data Model Rules)

```
SELECT Emp, Dpnd
FROM Emp LEFT JOIN Dpnd ON EmpId=DpndEmpId…
```

….. AND Age<18 or WHERE Age<18 Join without filtering

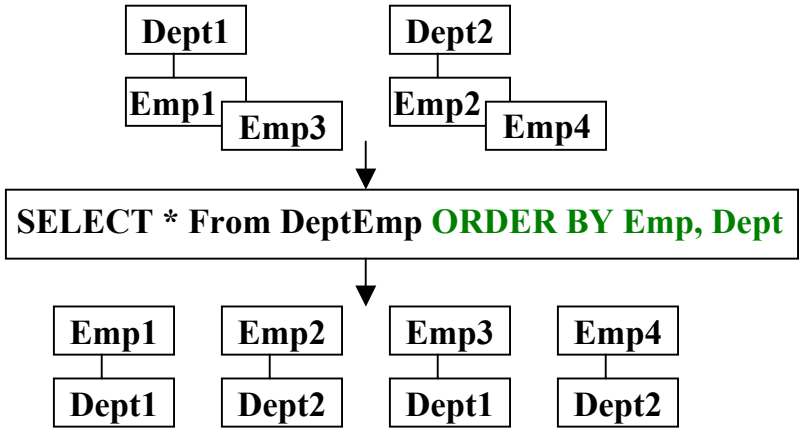
Results: Emp1 Null Emp2 Dpnd2
Emp2 Dpnd2

Emp	Dpnd	Age
Emp1	Dpnd1	20
Emp2	Dpnd2	17

Difference between WHERE and ON clause data filtering

- Most SQL professionals do not know that WHERE and ON operate differently
- WHERE clause tests and affects the entire row, which will remove entire rows
- This means that WHERE clause logically operates after all joins are performed
- ON tests each join point separately, they only affect their portion of rows
- This means ON operations perform as they are processed at row construction
- The ON filtering processing is very similar to XPath filtering, very useful

4.5) Order By for Hierarchical Data

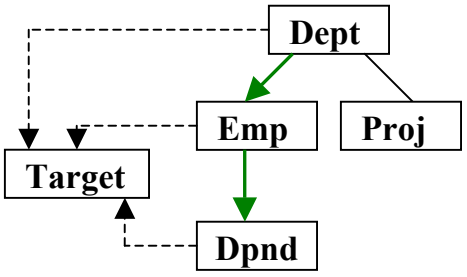


How ordering hierarchical data can change its structure

- **Order By can easily and inadvertently change the hierarchical structure**
- **Ordering needs to be controlled in order of hierarchical data nodes**
- **Nodes can be ordered in any way within their own node occurrences**
- **If nodes not ordered within hierarchical order, structure will change**
- **When structure changed by Order By, may change intended meaning**

4.6) SQL Update from Hierarchical Data

DeptView



SQL Insert using Hierarchical Data

```
INSERT INTO Target (DeptID,
EmpID, DpndID, Dpndnam,
DpndBD)
SELECT DeptNo, EmpNo,
DpndNo, DpndName, DpndBday
FROM DeptView
WHERE DeptNo=124 and
EmpNo=628 and DpndNo=456
```

SQL Insert operation using a hierarchical data source

- **Allows updating SQL databases using hierarchical data**
- **Uses hierarchical views directly**
- **WHERE clause navigates to specific hierarchical node occurrence**
- **Data can also be SELECTed from the active path to the target node**

4.7) Namespace Capability Through Data Qualification

Owner Qualified:	**SELECT** **Inv07**.Cost, **Inv09**.Cost **FROM** **Inv07**, **Inv09** **WHERE** **Inv07**.Cost> **Inv09**.Cost
Qualifier Qualified:	**SELECT** **Purchased**.Cost, **Sold**.Cost **FROM** **Purchased**.Inv07, **Sold**.Inv09 **WHERE** **Purchased**.Cost>**Sold**.Cost
Alias Qualified:	**SELECT** **Purchased**.Cost, **Sold**.Cost **FROM.** Inv07 **AS Purchased**, Inv09 **AS Sold** **WHERE** **Purchased**.Cost>**Sold**.Cost

- **Owner, qualifier or alias qualified works with Tables and Views**
- **Allows Tables and Views to have same column (field) names**
- **Similar characteristics to XML with More choices than XML**
- **Integrated into SQL operation**
- **Nesting of SQL Views supported automatically**

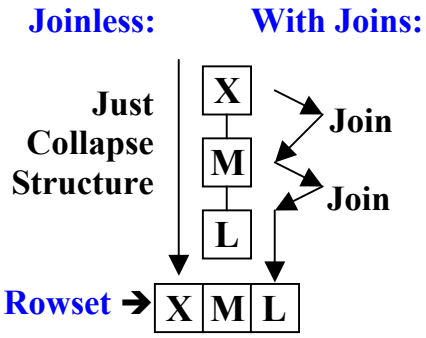
4.7.1) Renaming Qualified Data Fields

```
CREATE VIEW EasyView AS
SELECT  Purchased.Cost AS Input , Sold.Cost AS Output
FROM    Purchased.Inv07, Sold.Inv09
WHERE   Purchased.Cost>Sold.Cost
```

SELECT Input, Output FROM EasyView

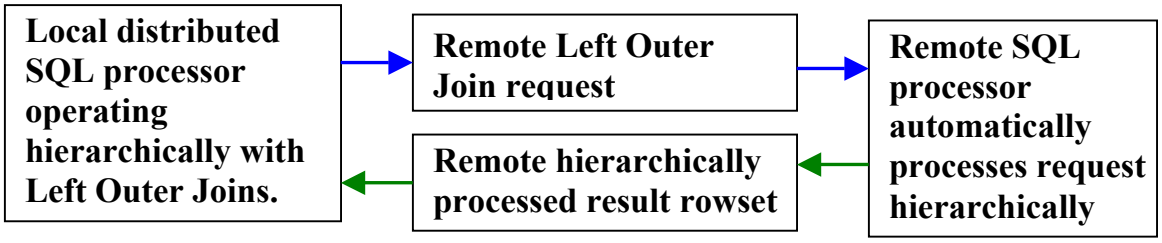
- **Renaming makes namespace capability more user friendly**
- **Renaming works with all forms of namespace capabilities:**
  - **Owner, ViewName.DataField**
  - **Qualifier, Prefix.DataField**
  - **Alias, AliasName.DataField**
- **Renaming works with views and tables/nodes**

5) Other Hierarchical Based Capabilities
5.1) XML Joinless Access



- **XML is still overly influenced by SQL and relational processing**
- **When SQL accesses XML expensive relational joins are simulated**
- **Left Outer Joins model hierarchical structures exactly**
- **Left Joins can optimize and access hierarchical structures directly**
- **This can efficiently collapse XML directly into a rowset w/o joins**
- **More powerful hierarchical optimizations can also be utilized**

5.2) Automatic Distributed Hierarchical Processing



- **SQL like XML, can automatically carry its hierarchical structure around with it**
- **The SQL Left Outer Join defines the hierarchical structure & goes with the SQL**
- **Distributed hierarchical processing is automatic with SQL distributed processors**
- **Modeled SQL is automatically processed hierarchically at the remote processor**
- **Hierarchically processed & preserved rowsets are returned to the home processor**

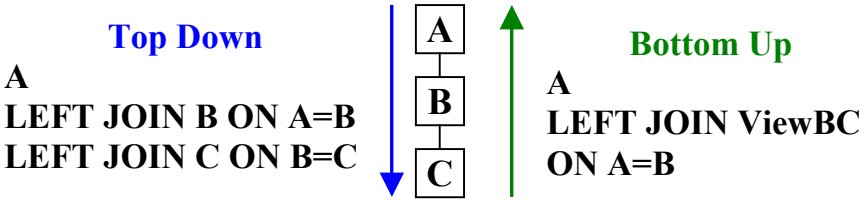
5.3) Full Ad Hoc Nonprocedural Processing Supported

SQL Ad Hoc Ability	Other XML Processors
SELECT list: Parameter driven	**Selected items placed in code**
FROM : Nonprocedural processing	**Physical procedural coding**
WHERE: Automatic nonlinear filtering	**No automatic LCA processing**
Hierarchical views: Flexible metadata	**Less flexible physical views**

SQL ad hoc querying advantages

- **SQL Nonprocedural hierarchical processing makes ad hoc processing possible**
- **The SELECT list is parameter driven, just add or remove a SELECT list item**
- **The FROM clause accepts hierarchical views and can join them dynamically**
- **The WHERE clause performs hierarchical filtering naturally using LCA logic**
- **All of these SQL operations can operate dynamically**

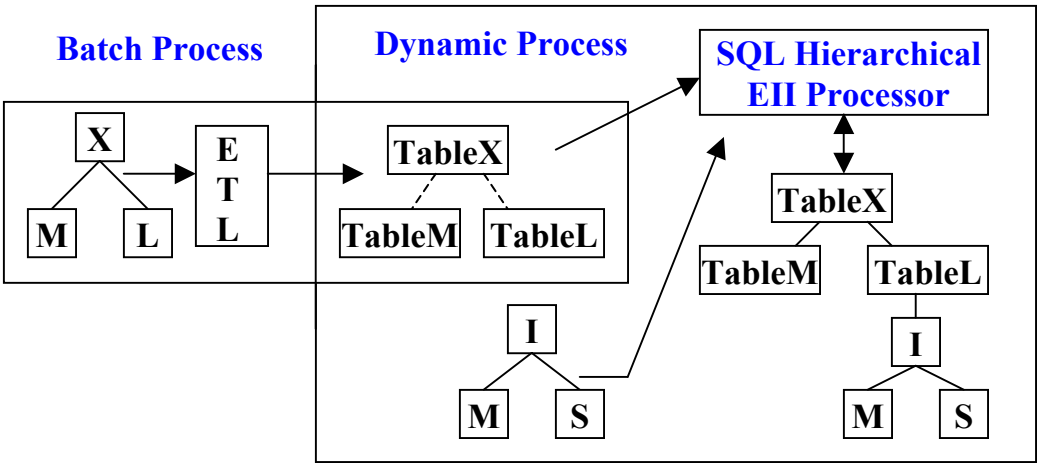
5.4) Hierarchical Structure Construction Order



Hierarchical structure construction order, down or up

- **Hierarchical structures, with their semantics, can be built in any order**
- **Bottom-up, top-down, left to right, or a combination of these methods**
- **Sibling order not meaningful to structure, but may be application specific**
- **Top-down is most natural and efficient way, avoids bottom-up throwaways**
- **This flexibility is useful with structures already stored in views as shown**
- **Expanded views can usually be rewritten to force top-down processing**

5.5) XML Batch ETL Integration at Hierarchical Dynamic EII Level

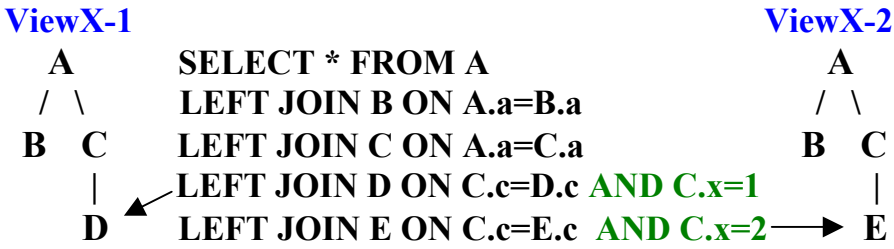


- **XML ETL integration shreds the data into relational tables**
- **SQL hierarchical processing elevates ETL shredded data to hierarchical data**
- **Enables XML ETL and SQL native XML to operate at hierarchical level**
- **XML ETL and SQL native XML seamlessly integrate hierarchically**
- **User has choice between ETL (batch) and EII (dynamic) with less tradeoffs**

6) Processing XML's Unconventional Hierarchical Structures

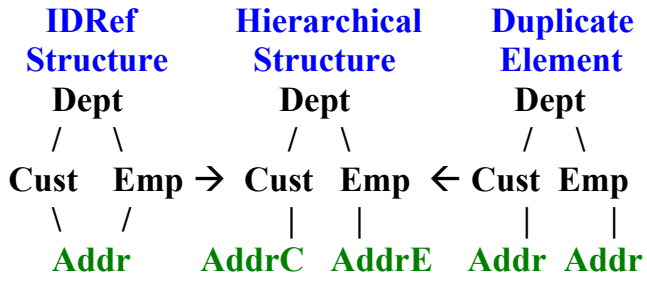
- **XML designed to specify hierarchies for XML markup not for database**
- **XML specification defines hierarchical data, but not how to process it**
- **XML structures are not as fixed as conventional hierarchical data structures**
- **Unconventional reuse of node types and use of network type structures**
- **No XML processor can be expected to support all capabilities**

6.1) Variable Structure Generation Controlled by Data



- **XML structures can dynamically vary in structure.**
- **SQL hierarchical structures can also vary their structure:**
  - **Using a "Depending On" operation**
  - **Uses the ON clause with a condition based on a value in the DB**
  - **If ON condition not true, join of the dependent occurrences skipped**
- **While SQL's variable method is limited, XML's is unlimited**
  - **Unlimited use is not advisable for DB use**
- **XML can change structures at will, but navigation is very difficult**

6.2) Mapping Network Structures to Hierarchical Structures



Hierarchical Mapping

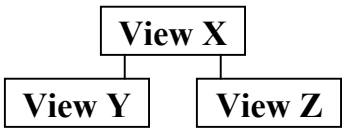
```
SELECT * FROM Dept
LEFT JOIN Cust ON …
LEFT JOIN Emp ON …
LEFT JOIN Addr AS AddrC …
LEFT JOIN Addr AS AddrE …
```

- **Network structure are ambiguous, can not be nonprocedurally processed**
- **XML IDref specifies alternate path, a node can be reached by multiple paths**
- **Duplicate node type on different paths is also ambiguous**
- **Renaming ambiguous node type, maps it to a hierarchical structure**

7) Hierarchical SQL View Use and its Importance

7.1) Hierarchical View Advantages

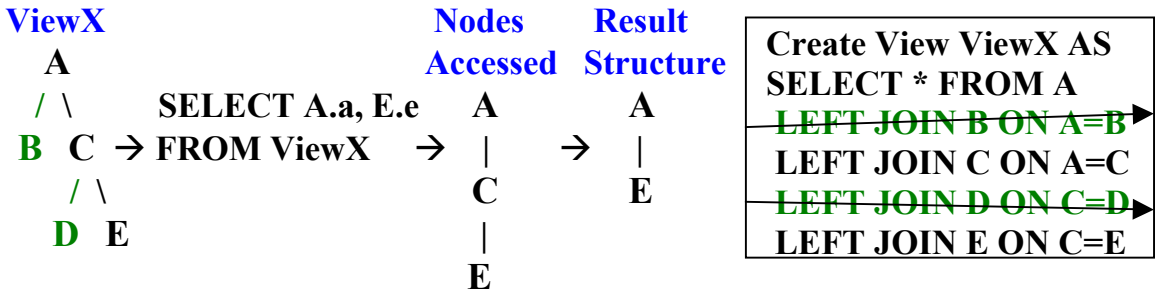
ViewXYZ:



SELECT Name, Address
FROM ViewXYZ

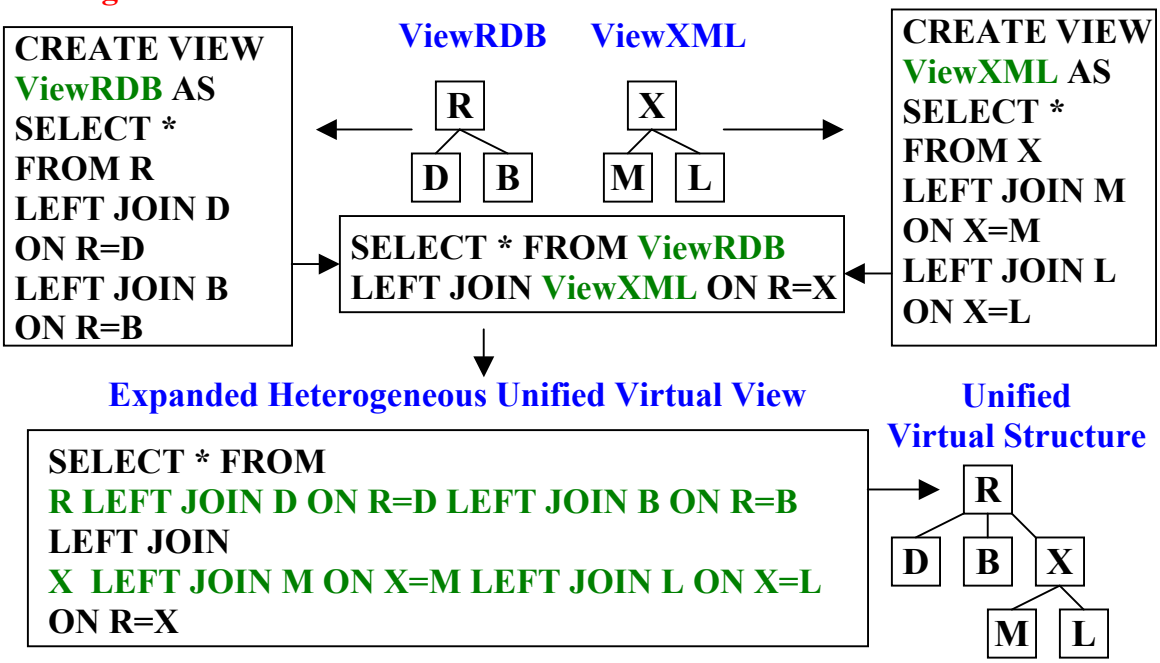
- **Abstraction: User does not need to know structure**
- **Reusable: Can be reused to build larger structures**
- **Optimizable: Meta data semantic optimization**
- **Embedded or joined views form larger structures preserving semantics**
- **Prefix and alias capabilities support XML namespace type of use**
- **Views of complex hierarchical structures operate correctly**
- **Views operate seamlessly with no limitations on use**
- **Many view capabilities come together synergistically**

7.2) Hierarchical Query Access Optimization



- **SQL & expanded view can be hierarchically optimized using Left Join metadata**
- **Only nodes referenced or on a path to a referenced node require access (A, C, E)**
- **Optimized out nodes (B and D) easily performed by removing their Left Joins**
- **Removing unneeded Left Joins maintains structure semantics unlike Inner Joins**
- **Very dynamic optimization based on which data items are selected for each query**
- **Besides significantly decreasing access, optimization also cuts data explosions**
- **Optimizations allows single user friendly global data view without overhead**

7.3) Heterogeneous Unified Virtual Views



- **Expanded views create seamless consistent heterogeneous hierarchical view**
- **Entire virtual view can be hierarchically optimized**
- **Semantically defined entirely in ANSI SQL, ultimate unified virtual views**
- **It is also directly executable in SQL guaranteeing consistent operation**

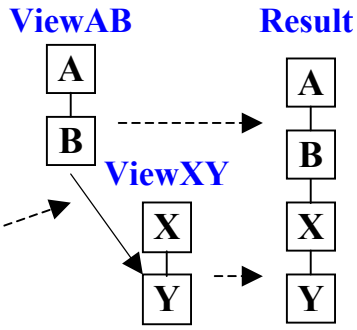
8) More on the Internal Processing of this Technology

8.1) Right Sided View Nesting

```
SELECT * FROM
    A LEFT JOIN B ON A.a=B.b ←ViewAB
LEFT JOIN
    X LEFT JOIN Y ON X.x=Y.y ←ViewXY
ON B.Key=Y.key
```

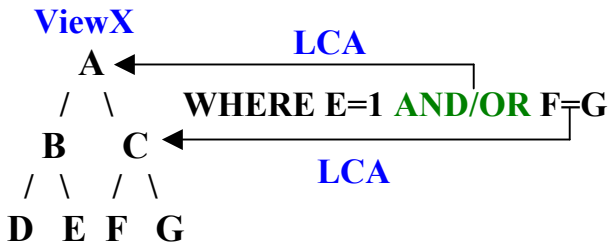
Link

Right sided view is automatically nested



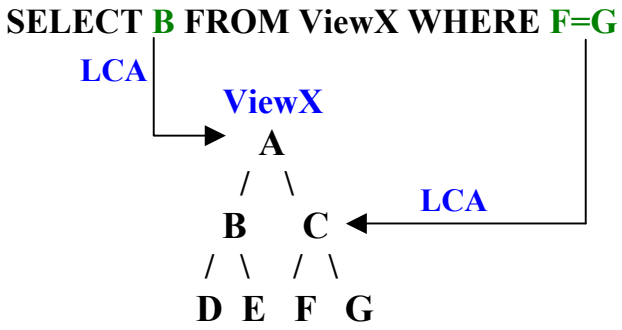
- **Right sided nesting needed when lower node level referenced before root**
- **The root still needs to be accessed first, right sided nesting makes possible**
- **Physical structures need right sided nesting for node access in any order**
- **Right sided nesting delays joining until view is fully materialized**

8.2) Advanced Lowest Common Ancestor Logic
8.2.1) Compound Lowest Common Ancestor (LCA) Logic



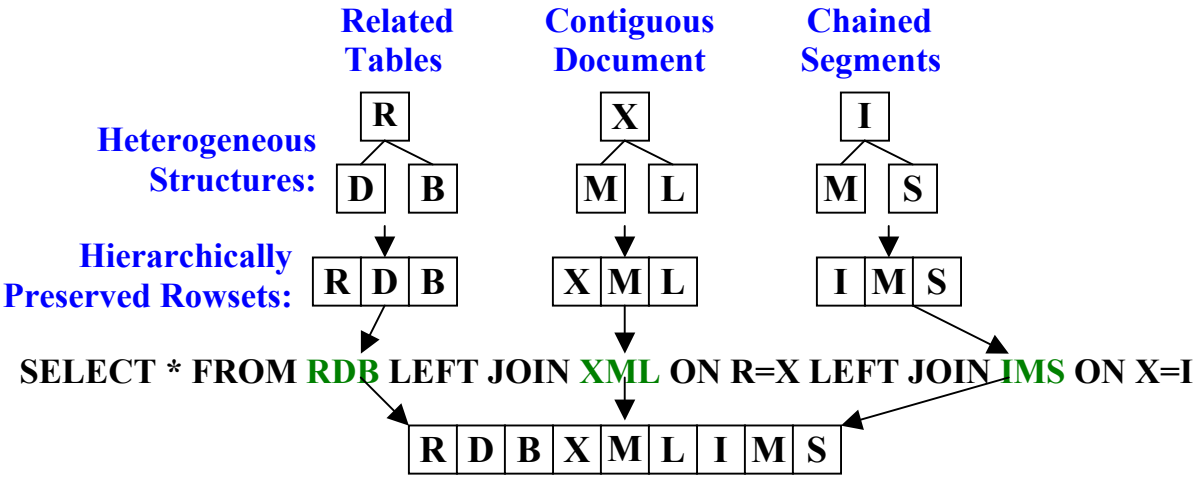
- **LCA processing is a long known hierarchical principled technology**
- **LCA is also known as Nearest/Closest Common Ancestor connecting two nodes**
- **The LCA Node data occurrence between two node sets controls meaningful relationships**
- **LCA also used in many fields of study such as AI and Bio Tech**
- **The more legs accessed in a query, the more LCAs used in processing**
- **Multiple LCAs processing naturally nested by Cartesian Product (CP)**
- **Nested LCA processing occurs naturally row at a time by RDB engine**

8.2.2) Lowest Common Ancestor Logic Type 2



- **Two types of LCAs for hierarchical query, Type 1 and type 2**
- **LCA Type 1 used with WHERE clause to produce CP under LCA (F+G under C)**
- **LCA Type 2 used with SELECT/WHERE for qualification (Select B, Where LCA F,G)**
- **LCA Type 1 and type 2 can be combined as in example above (Select B, Where F=G)**

8.3) Logical and Physical Structure Processing Consistency

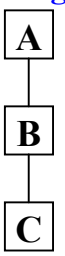


- **Having same semantics, logical & physical hierarchical structures operate the same**
- **Hierarchical operations are based on solid principles**
- **For above reasons, they can be combined seamlessly and processed consistently**
- **Heterogeneous hierarchical structures form a unified virtual view in SQL rowset**
- **Logical and physical hierarchical structures are seamless and contiguous in rowset**
- **With contiguous rowset, hierarchical operations perform accurately & consistently**

8.4) Nonlinear Internal Hierarchical Navigation

8.4.1) Single Leg Navigation

Single Leg Structure

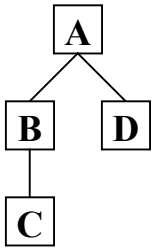



```
          GF  A
Loop3: GF
Loop2: GF
Goto Proce
Loop1: GN C
Process: Perform
Processing
If C Found Goto Loop1
          GN B
IF B Found Goto Loop2
          GN A
If A Found Goto Loop3
End of DB
```

- **Figure above shows internal hierarchical navigation in hierarchical query engines**
- **Get First (GF) establishes beginning position for specified node type**
- **Get Next (GN) reads within specified node's data occurrences under parent node**
- **Get First & Get Next can support a search argument (skip sequential processing)**
- **Navigation depends on parent's position, from top to bottom, then bubbles up**
- **Controls range of access so that lower node access can or cannot move parent node**
- **With knowledge of data needed, GN processing can be delayed for a path call later**

8.4.2) Multiple Leg Navigation

Multi-leg Structure



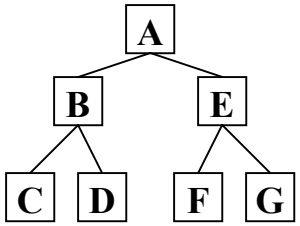
Multi-leg Navigation

```
        GF  A
Loop4: GF  B
Loop3: GF  C
Loop2: GF  D
Goto Process
Loop1: GN D
Process: Perform Processing
If D Found Goto Loop1
        GN C
If C Found Goto Loop2
        GN B
IF B Found Goto Loop3
        GN A
If A Found Goto Loop4
End of DB
```

- **Multiple positioning is standard, allows simultaneous position on multiple legs**
- **Cartesian product (CP) produces all combinations between sibling legs**
- **CP contains LCA logic necessary for hierarchical processing (WHERE C=D)**
- **Get First (GF) and Get Next (GN) offer very flexible navigation and processing**
- **Coordination between legs of LCA is key to hierarchical processing**
- **Positioning is lost on every dependent node of an accessed node**

8.5) Data Structure Extraction (DSE) Technology

Data Structure



Node Edge Table

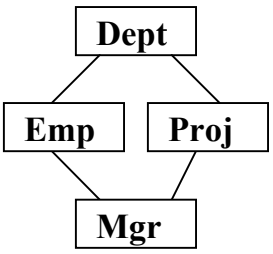
Node Name	Node No.	Hier Level	Parent No.
A	1	1	0
B	2	2	1
C	3	3	2
D	4	3	2
E	5	2	1
F	6	3	5
G	7	3	5

Node edge table representing data structure

- **DSE technology examines expanded Left Join to derive hierarchical structure**
- **Can occur dynamically at runtime to derive entire unified structure**
- **Builds a node edge table similar to above to represent structure**
- **Enables extending hierarchical processing out of SQL engine**

9.0 Advanced Topics
9.1) Network Structures

Network Structure **SQL Network Structure Definition**




```
CREATE VIEW DeptView AS
SELECT * FROM Dept
LEFT  JOIN Emp ON DeptKey=EmpFKey
LEFT  JOIN Proj  ON DeptKey=ProjFKey
LEFT JOIN  Mgr ON MgrKey=EmpEKey
                  OR MgrKey=EmpPKey
```

Network structure definition using SQL

- **Defined by SQL Left Join Node reference of two or more legs with OR op**
- **Multiple paths to a node make for ambiguous nonprocedural queries**
- **Navigation required, nonprocedural query needs no procedural navigation**

Conclusion

- **Nonlinear processing can naturally utilize goldmine of hierarchical semantics**
- **Nonprocedural 4GLs automatically process most complex nonlinear queries**
- **User does not need to know the hierarchical structure to pose a query**
- **User does not need to supply processing or navigation logic**
- **Automatic utilization of semantics increases value of data**
- **Nonlinear hierarchical processing is based on long proven hierarchical principles**
- **Hierarchical processing, a relational processing subset, is mathematically sound**
- **ANSI SQL hierarchical process helps standardized hierarchical processing**
- **Logical & physical hierarchical structures have same semantics, easy integration**
- **Hierarchical processing can be specified at a high conceptual level in SQL**

