2009 SAP BusinessObjects USER CONFERENCE

Powered by the Global BusinessObjects Network

Universe Models for Recursive Data

David G. Rathbun
BI Solutions Architect, PepsiCo B+IS

Presentation Abstract

- Recursive data can present a special challenge to SAP BusinessObjects developers because SQL is not natively able to process the relationships. This presentation will show several different methods for modeling recursive data along with the pros and cons of each. All of the methods shown have been used on real-world projects. Attendees will gain a better understanding of the complexities of dealing with inventory models, organizational hierarchies, and other types of recursive data.

Learning Points

- Review recursive data scenarios and discuss the difficulties they present to a Universe developer
- Review various models that can be considered for representing recursive data in a reporting environment
- Review pros and cons of each model, all based on real-world implementations

About Dave

- Dedicated to BusinessObjects solutions since 1995
  - Consultant and trainer for fifteen years
  - Currently BI Solutions Architect for PepsiCo
- 14 consecutive years presenting at major BI conferences
  - United States, Europe, Australia
- Charter member of BOB
  - http://busobj.forumtopics.com
- I Blog! Dave's Adventures in Business Intelligence
  - http://www.dagira.com
- Selected to the SAP Mentor program for 2009

SAP Mentor 2009

Demonstration Platform

- Demonstration universes
  - Island Resorts Marketing
  - eFashion
  - Prestige Motors
  - Demonstration databases were converted to Oracle
- Software configuration
  - BusinessObjects Enterprise XI 3.0
  - Oracle 10g
- BusinessObjects toolset
  - Web Intelligence Rich Client
  - Universe Designer

 Demonstration icon  Additional demonstrations only if time allows

Agenda

- Overview
  - Define recursion
  - List the challenges
  - Preview the solutions
- Implementation
  - Detail the challenges
  - Review the solutions
  - See the results
- Questions and Discussion
  - Complex topic with many demonstrations
  - Please hold questions until the end

Defining Recursion

- Recursion
  - See "Recursion".
- Or in other words
  - In order to understand recursion you must first understand recursion
- Google says

What is Recursive Data?

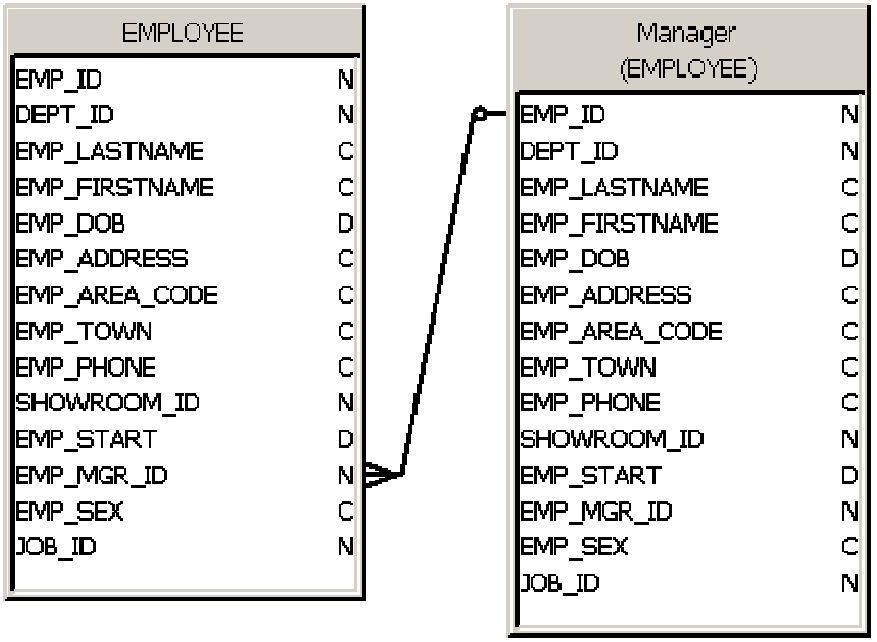
- Hierarchical and self-referencing data often represented by a tree structure
  - Hierarchical because there are defined levels
  - Self-referencing because primary and foreign keys are in the same table
- Hierarchical data is not a huge problem
  - Geographical hierarchies: Country, State, City
  - Time hierarchies: Year, Quarter, Month
- Recursion is a problem
  - SQL is not a procedural language and does not offer native recursive functionality

Recursive Data Is Quite Common

- Company organizational structure
- Inventory BOM (Bill of Materials)
- Project management (WBS task structure)
- Multi-level marketing (Amway, Avon)
- Bernie Madoff's investment strategy

Recursive Relationship From Prestige Motors HR

- When a table joins to itself that is recursive data
- This relationship says:
  - A manager has one or more employees
  - An employee has zero or one manager
  - A person can be both an employee and a manager due to recursion
  - There is nothing to indicate how deep the recursion goes



Raw Data From Prestige Motors HR

- Max depth of 4
- Noakes does not have a manager

EMP_ID	EMP_LASTNAM	EMP_MGR_ID
101	Noakes	
102	Ferrerez	101
103	Field	102
104	Fraser	101
105	Snow	101
106	Speed	105
107	Spencer	105
108	Helen	101
109	Thomas	101
110	Thatcher	109
111	Davis	101
201	Pickworth	101
202	Forest	201

LEVEL	EMP_ID	EMP_NAME
1	101	Noakes
2	102	Ferrerez
3	103	Field
2	104	Fraser
2	105	Snow
3	106	Speed
3	107	Spencer
2	108	Helen
2	109	Thomas
3	110	Thatcher
2	111	Davis
2	201	Pickworth
3	202	Forest
4	203	Brown
4	209	Hilary
3	204	Porter
4	205	Irving
4	206	Bailey
3	207	Duckworth
4	208	Ince
2	301	Dagmar
3	302	Presley
4	303	Perry
4	304	Hubert
3	305	Adamson
4	306	Beaver
4	307	Motson

Typical Recursive Questions

- Company organization structure
  - Who do I work for? Who works for me? What is the total salary for my direct / indirect reports?
- Inventory BOM
  - Where is this component used? What is required to build this assembly? What is the total cost of the materials for this component?
- Project management
  - What is the deliverable for this task? What are all of the activities under this task? What is the total projected hours for this deliverable?

Some Types of Hierarchies

- Clean
  - A hierarchy that has no issues or challenges
- Unbalanced
  - A hierarchy with inconsistent depths
- Ragged
  - A hierarchy with inconsistent node paths
- Lateral
  - A hierarchy with sideways node paths
- All of these will be defined with examples shortly

Methods of Handling Recursive Data

- Universe aliases
- Flattened structures
  - Single table with column values
  - Snowflake tables
- Ancestor or Descendent Model
- Depth First Tree Traversal
  - Assign node “left” and “right” values
- Some other methods not considered today
  - Oracle CONNECT BY
  - Stored procedures

Agenda

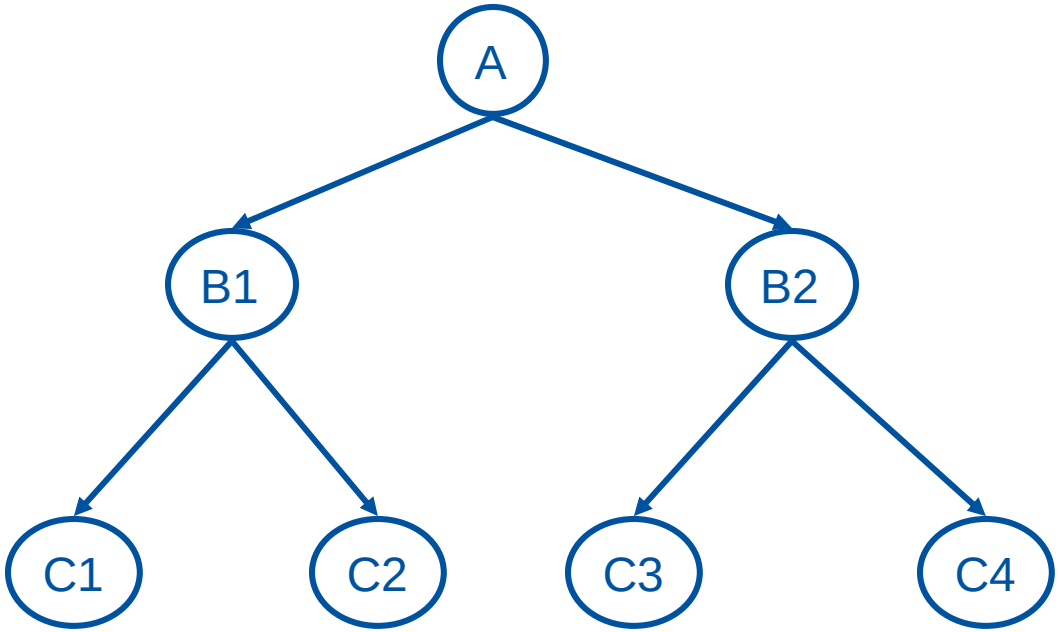
- Overview
  - Define recursion
  - List the challenges
  - Preview the solutions
- Implementation
  - Detail the challenges
  - Review the solutions
  - See the results
- Questions and Discussion

Review Hierarchy Challenges

- The following slides will visually demonstrate different challenges that can occur with recursive hierarchical data
  - Clean
  - Unbalanced
  - Ragged
  - Lateral
- A picture is worth a thousand words…

Clean Hierarchy

- Consistent depth
- No missing nodes
- No lateral connections
- Challenge: None

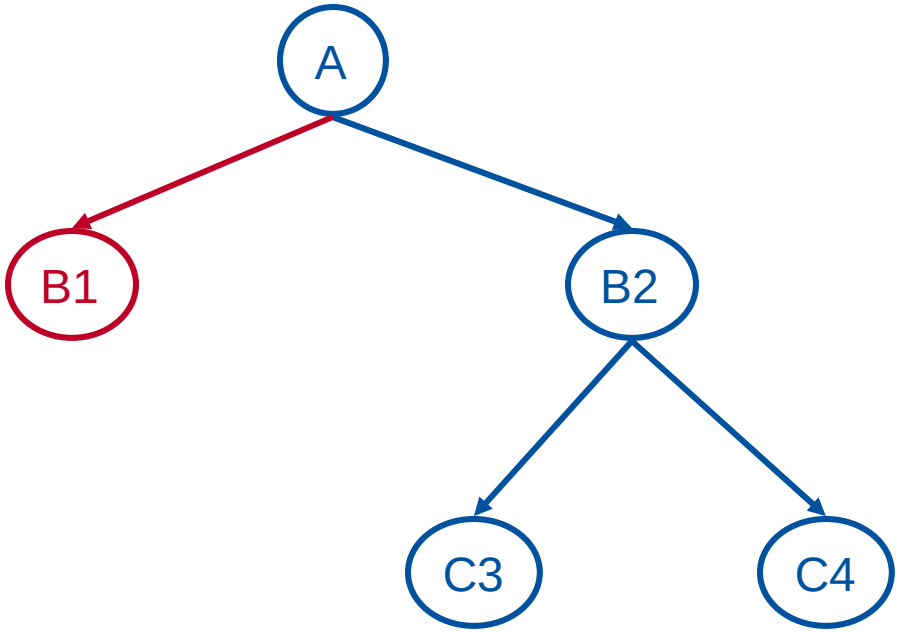


A, B, and C represent different node types or levels in the hierarchy

For example, President, Vice President, and Division Manager

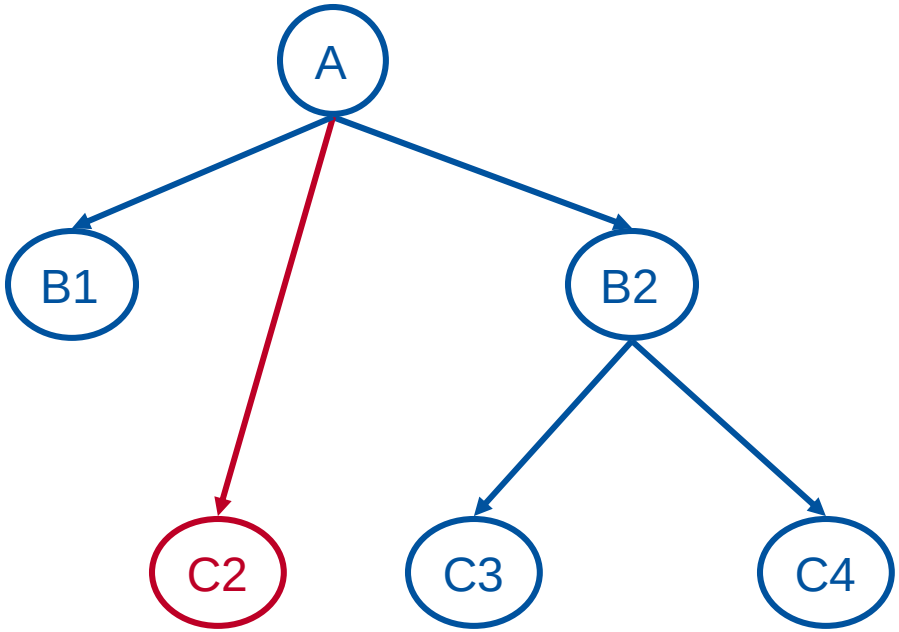
Unbalanced Hierarchy

- Inconsistent depth
- One branch of the tree is not as long as the other branches
- Consider a Vice President of a company with no direct reports
- Challenge: Moderate



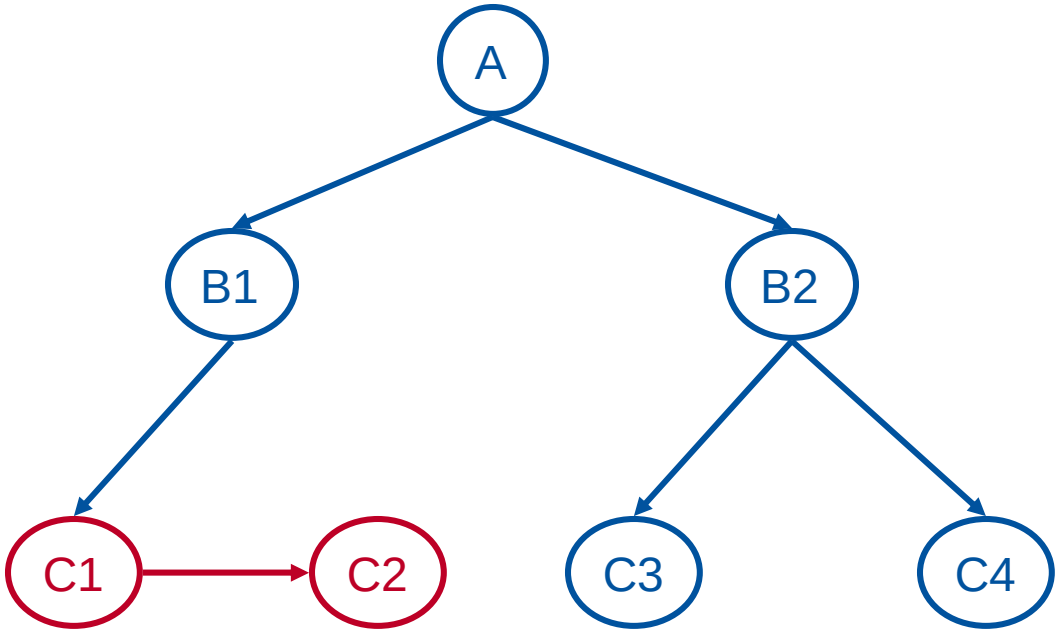
Ragged Hierarchy

- Missing nodes
- Data is missing from the middle of a path
- Consider a division manager that reports direct to the president without going through a VP
- Challenge: Difficult



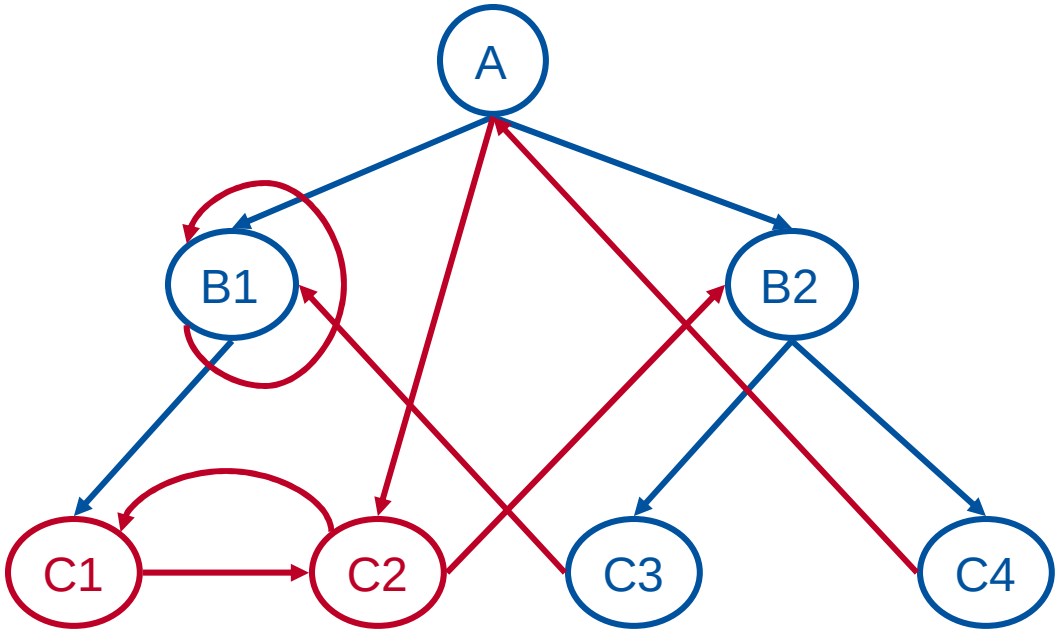
Lateral Hierarchy

- Lateral connections
- One node relates across the tree rather than up or down
- Consider a lateral report in a company hierarchy
- Challenge: Very Difficult



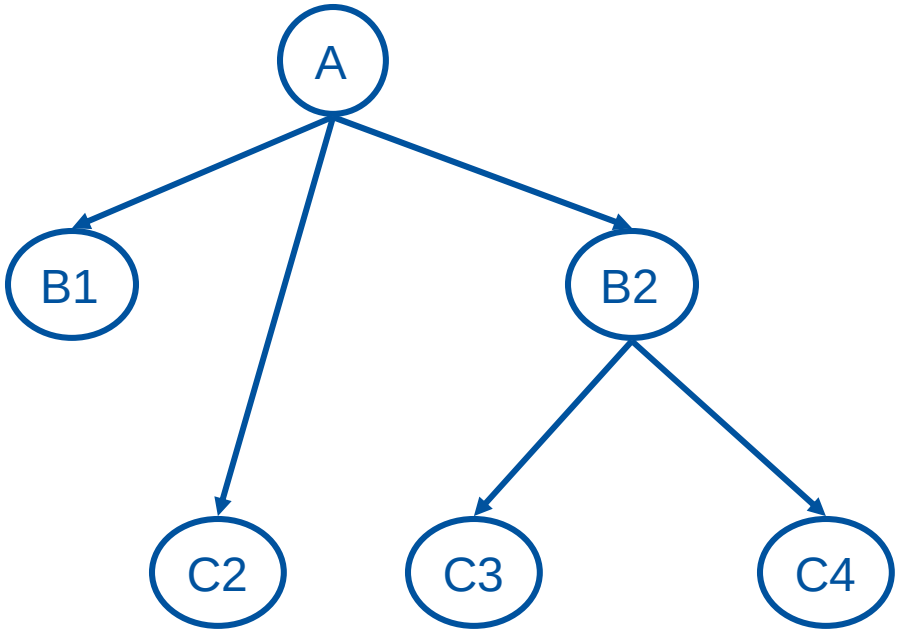
Nightmare Hierarchy

- I hope you never have to deal with something like this…
- Challenge: Impossible!



Combination

- It is possible to have more than one challenge to overcome
- What are the solutions?
- Are there any best practices for representing recursive data in a universe?

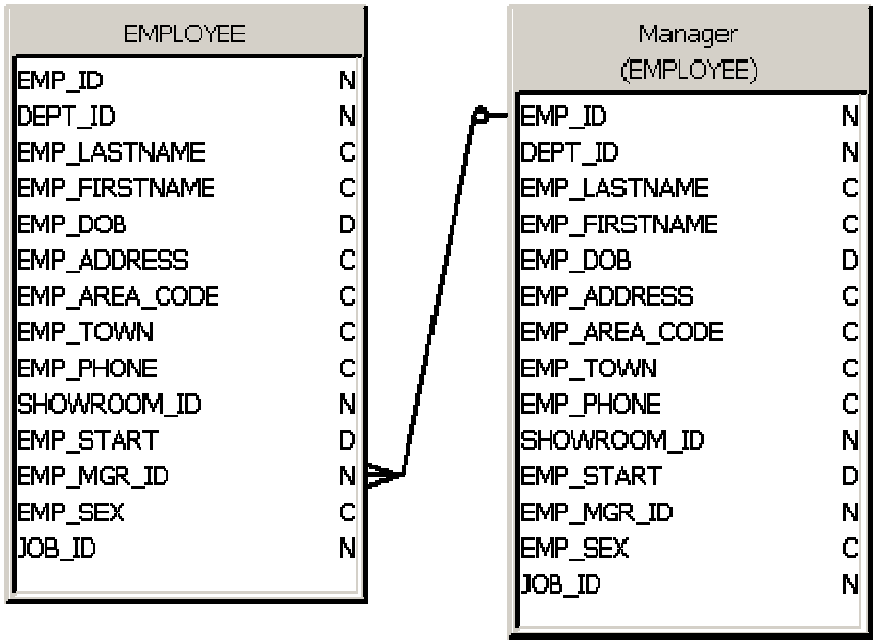


Defining The Solutions

- The next few slides will define each of the solutions
  - Universe aliases
  - Flattened structure (Single or Snowflake)
  - Ancestor / Descendent
  - Tree traversal
- Most of these solutions cannot be completely solved within the universe
- Require ETL to load specific data structures
- Each solutions will be defined, demonstrated, and analyzed for pros and cons

Universe Aliases Solution

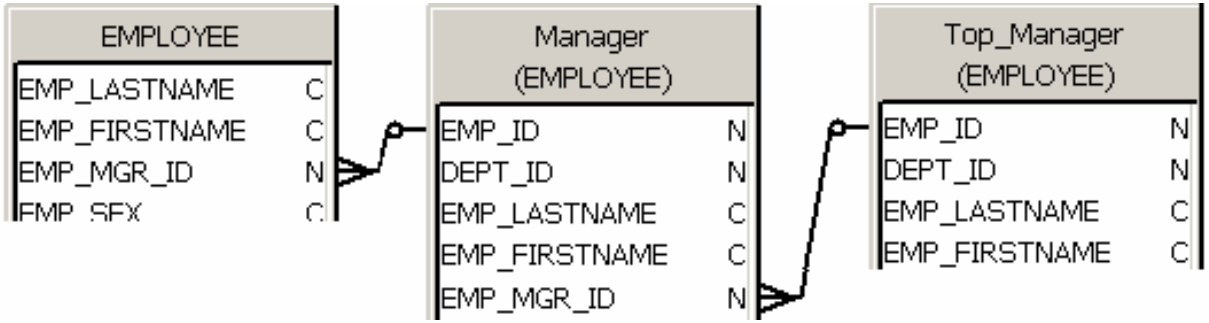
- Only recursive solution that can be done completely in the universe



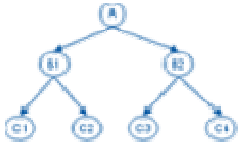
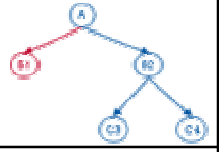
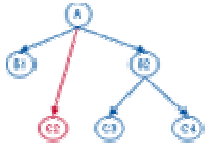
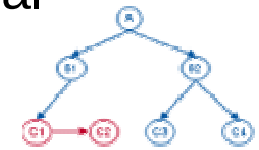
- Build aliases and query hierarchy

Universe Aliases Pros and Cons

- Pros
  - Self-contained solution that does not require ETL
- Cons
  - Does not handle lateral relationships at all
  - Must use an outer join
    - Required for the top node of the tree
    - Required for unbalanced hierarchy
  - Must have an alias for each level of the hierarchy
    - Universe maintenance if levels are created or dropped



Universe Alias Scorecard

Clean 		The universe designer must know the depth of the hierarchy in order to build the correct number of aliases but it works well
Unbalanced 		An unbalanced hierarchy could be handled by making all of the joins outer joins but drilling up from an undefined node fails
Ragged 		Cannot effectively represent missing nodes as there is no “bridge” to the next level of the hierarchy
Lateral 		Cannot effectively represent a lateral relationship with simple aliases

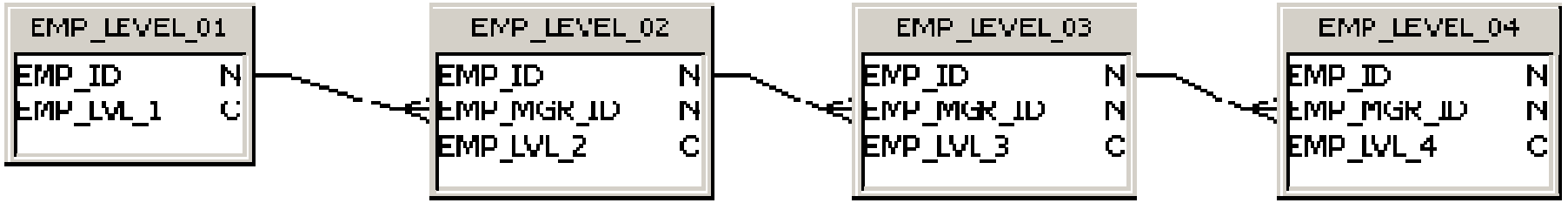
Flattened Structure Solutions

- Data in rows is pivoted to columns

EMP_ID	EMP_LASTNAM	EMP_MGR_ID
101	Noakes	
102	Ferrerez	101
103	Field	102
104	Fraser	101

EMP_LVL_1	EMP_LVL_2	EMP_LVL_3
Noakes	Ferrerez	Field
Noakes	Fraser	

- Columns can be converted to snowflake tables



- Snowflake process is very similar to using aliases

Flattened Structure Columns Pros and Cons

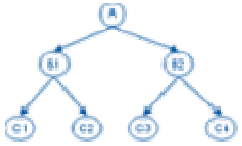
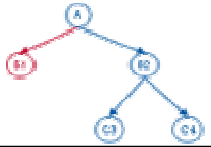
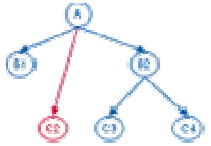
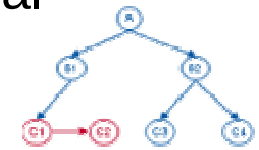
- Pros
  - Handles unbalanced hierarchies better than aliases
  - Missing value is a null column rather than missing relationship
  - Can handle ragged hierarchies with “plug nodes”

> A plug node fills in missing values to represent the relationship
>
> Level 1 » **Level 1-3 Plug** » Level 3

- Cons
  - Must have a column defined for each level
  - Universe and ETL maintenance if levels change

EMP_LVL_1	EMP_LVL_2	EMP_LVL_3
Noakes	Ferrerez	Field
Noakes	Fraser	

Flattened Structure Columns Scorecard

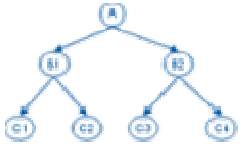
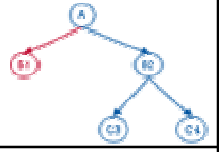
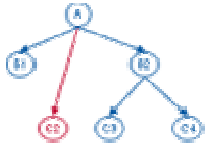
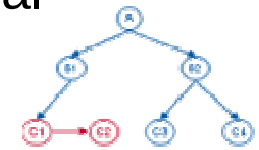
Clean 		Must know in advance how many levels to process, but a clean hierarchy works well when flattened
Unbalanced 		Since everything to the maximum expected depth has a column, unbalanced nodes are simply null columns in a row
Ragged 		Drilling becomes a problem with ragged hierarchies; this can be addressed with a good “plug node” strategy
Lateral 		A flattened structure has no way to store two values in a single column

- Demonstrate flattened hierarchy

Flattened Structure Snowflake Pros and Cons

- Pros
  - Smaller storage requirement
  - Some queries are more efficient
  - Easier maintenance than single table
    - When new levels occur simply add a new table with same structure
- Cons
  - Must have a table defined for each level
  - Universe and ETL maintenance if levels change
  - Same outer join requirement as aliases
  - Plug nodes become “plug rows” which increases complexity

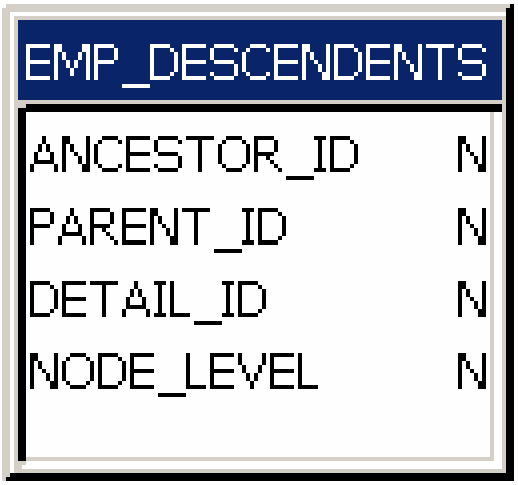
Flattened Structure Snowflake Scorecard

Clean 		Must know in advance how many levels to process, but a clean hierarchy works well with snowflake tables
Unbalanced 		Just like aliases, a snowflake suffers from unbalanced nodes and would require outer joins
Ragged 		Just like aliases, a snowflake suffers from ragged nodes as the drill path is not complete
Lateral 		Just like a flattened row, a snowflake structure cannot store two values at the same level

- Demonstrate flattened snowflake hierarchy

Ancestor / Descendent Solution

- One primary issue with recursion is the unknown depth
- This model represents the entire tree – no matter how deep – with the same data structure
- A user can start at any node of the tree
- Three types of data
  - Ancestor is any starting node for the tree
  - Parent is a parent node
  - Detail is a child node
- Row structure skips generations but still maintains the structural information



Building Ancestor Records

- Ancestor table is populated using Oracle CONNECT BY or other recursive techniques

EMP_ID	EMP_LASTNAM	EMP_MGR_ID
101	Noakes	
102	Ferrerez	101
103	Field	102
104	Fraser	101
105	Snow	101
106	Speed	105
107	Spencer	105
108	Helen	101
109	Thomas	101
110	Thatcher	109
111	Davis	101
201	Pickworth	101
202	Forest	201

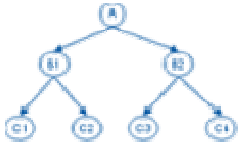
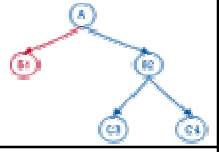
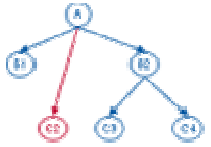
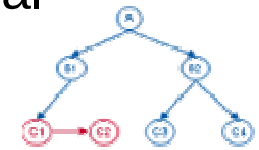
Ancestor Id	Node Level	Parent Id	Detail Id
101	0	101	101
101	1	101	102
101	1	101	104
101	1	101	105
101	1	101	108
101	1	101	109
101	1	101	111
101	1	101	201
101	1	101	301
101	2	102	103
101	2	105	106
101	2	105	107

- Demonstrate queries from ancestor hierarchy solution

Ancestor / Descendent Pros and Cons

- Pros
  - Answers all of the typical recursion questions
  - Does not require outer joins
  - Does not require plug nodes
- Cons
  - Does not support standard drilling technique
  - Inflates the row count
    - 27 employees becomes 80 descendent records
    - Each possible parent node is stored as an ancestor
  - May need to change objects to answer different questions
    - Use the ancestor to answer “Who do I work for?”
    - Use the child to answer “Who works for me?”
  - Can’t generate proper tree output

Ancestor / Descendent Scorecard

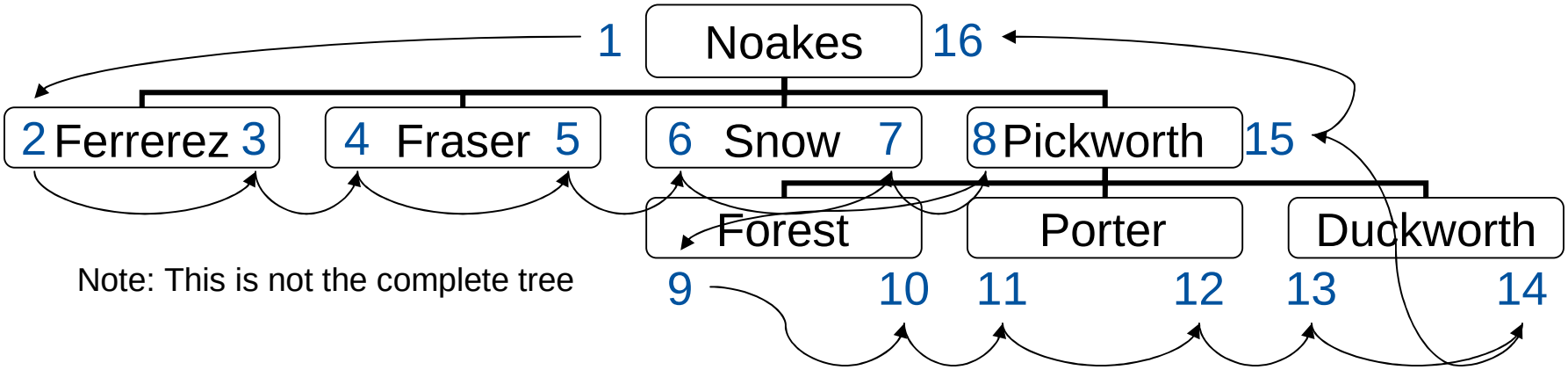
Clean 		Can represent a clean hierarchy just fine but cannot recreate the tree structure
Unbalanced 		Unbalanced hierarchies do not present a problem as the rows are not created
Ragged 		Ragged hierarchies do not present a problem because the node type is not important (preserved) in this structure
Lateral 		Lateral nodes can be modeled by allowing a node and its parent to have the same level but it's not a perfect solution

Tree Traversal Solution

- Store the node relationships on each row
  - Each row gets a left and right node value
  - These values are assigned based on a preorder or Depth First tree traversal algorithm
  - Optionally calculate and store the node level at the same time
- Original data remains intact
  - No transformation is performed
  - 27 rows of employee data remains 27 rows

Tree Traversal Node Assignments

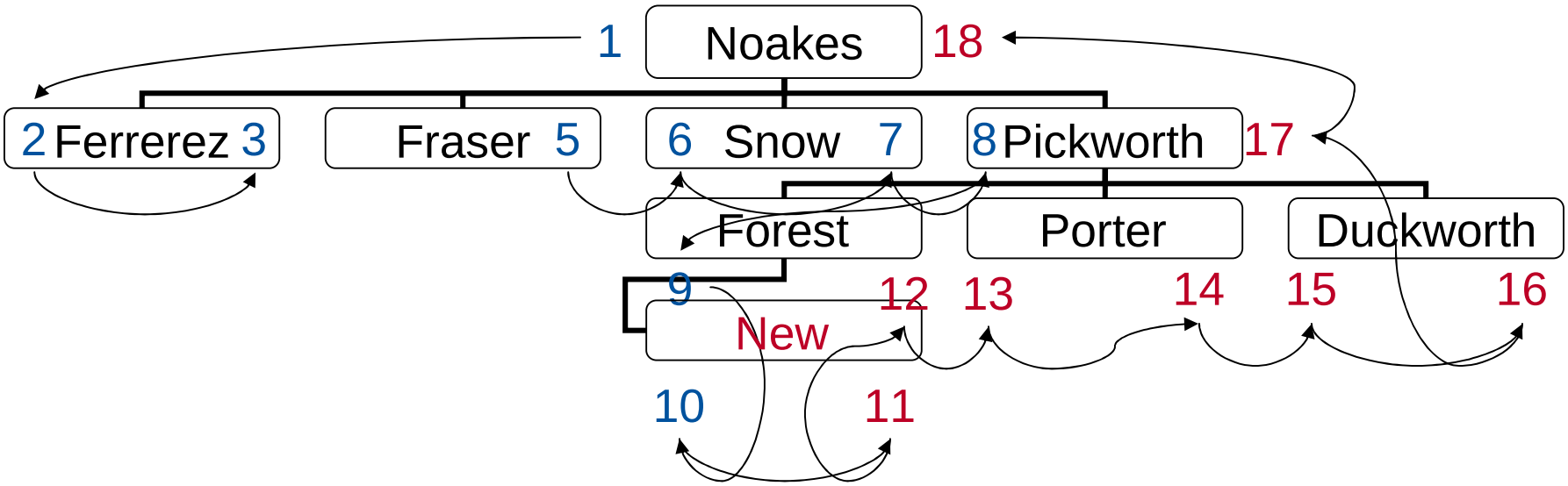
- Graphical representation of node assignments by “walking the tree”



- Tree traversal means walk around the outside edge
  - Number each node each time you pass the left or right side
  - Store those values in the original source table

Tree Traversal Maintenance

- Inserts and updates must adjust left and right values




```
Update table set left = left + 2 where left >= 10;
Update table set right = right + 2 where right >= 10;
Insert (name, left, right) values ('New', 10, 11);
```

Using Tree Data

- Who do I work for?
  - Left < My Left and Right > My Right and Level = My Level - 1
- Who are my direct descendants?
  - Left > My Left and Right < My Right and Level = My Level + 1
- Who are my indirect descendants?
  - Left > My Left and Right < My Right and Level > My Level + 1
- Who are all of my descendants?
  - Left > My Left and Right < My Right
- What is my level in the hierarchy?
  - Provided directly by the table
- Create a complete tree
  - Display all records order by LEFT indent by LEVEL

Employee Hierarchy Tree

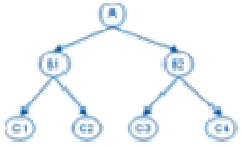
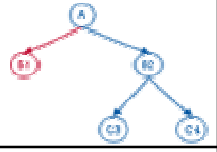
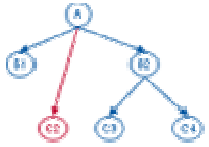
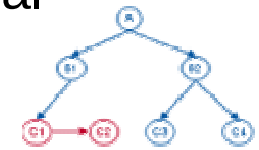
ID	Full Name	Left Tree	Right Tree
101	Noakes, Nicholas	1	54
102	Ferrerez, Ferdinand	2	5
103	Field, Felicity	3	4
104	Fraser, Frank	6	7
105	Snow, Sara	8	13
106	Speed, Sonya	9	10
107	Spencer, Steve	11	12
108	Helen, Harrison	14	15
109	Thomas, Tom	16	19
110	Thatcher, Terry	17	18
111	Davis, Diana	20	21
201	Pickworth, Paul	22	39
202	Forest, Florence	23	28
203	Brown, Bella	24	25
204	Porter, Pete	29	34
205	Irving, Ira	30	31
206	Bailey, Ben	32	33
207	Duckworth, Dave	35	38
208	Ince, Ian	36	37
209	Hilary, Hibbs	26	27

Name	Descendants	Parents	Level
Noakes, Nicholas	26	0	1
Ferrerez, Ferdinand	1	1	2
Field, Felicity	0	2	3
Fraser, Frank	0	1	2
Snow, Sara	2	1	2
Speed, Sonya	0	2	3
Spencer, Steve	0	2	3
Helen, Harrison	0	1	2
Thomas, Tom	1	1	2
Thatcher, Terry	0	2	3
Davis, Diana	0	1	2
Pickworth, Paul	8	1	2
Forest, Florence	2	2	3
Brown, Bella	0	3	4
Hilary, Hibbs	0	3	4
Porter, Pete	2	2	3
Irving, Ira	0	3	4
Bailey, Ben	0	3	4
Duckworth, Dave	1	2	3
Ince, Ian	0	3	4

Tree Traversal Pros and Cons

- Pros
  - Solves all four recursive challenges presented today
  - Does not require inflated row count like ancestor model
- Cons
  - Requires scripts or triggers to maintain tree node values
  - Does not support native drilling process
  - Requires the user to understand tree structure

Tree Traversal Scorecard

Clean 		A tree works very well with a clean hierarchy
Unbalanced 		A tree can handle unbalanced nodes because that branch simply stops
Ragged 		A tree can handle ragged nodes because the node type no longer defines the structure
Lateral 		A tree can even handle lateral nodes because it leaves the nodes in place and follows the relationship

Drilling Considerations

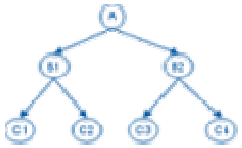
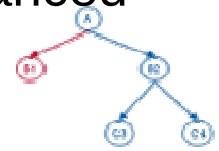
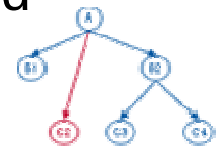
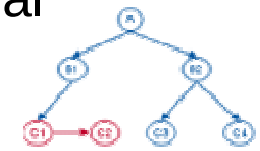
- Our expectation is that we will be able to drill on our data
  - Drilling is done from column to column
  - Level 1 » Level 2 » Level 3
- Some of the models discussed today are not drillable
- These models leave the data in rows rather than columns
  - Ancestor
  - Tree
- Flattening pivots the hierarchy
- Other models leave the recursion in place but provide enough information to build reports
  - Use the OpenDocument() function to drill on these models

Drilling Using OpenDocument()

- The OpenDocument() function can be used to simulate drilling
  - Native drilling returns the entire cube to the report
  - OpenDocument() drilling requires a query refresh
- Parameters
  - Starting Node
  - Depth

💻 Demonstrate drilling on a tree model using the OpenDocument() function

Overall Scorecard

	Alias	Flat	Snowflake	Ancestor	Tree
Clean 	🙂	🙂	🙂	🙂	🙂
Unbalanced 	😐	🙂	😐	🙂	🙂
Ragged 	☹	😐	😐	🙂	🙂
Lateral 	☹	☹	☹	😐	🙂

Special Considerations

- Ancestor model requires more disk space
  - Each node exists as a parent with all of its children
  - Duplicated data for maximum flexibility
- Tree model requires most complex coding
  - Don’t let the smiling faces on the prior slide fool you
  - This is a complex data model that can handle just about any recursive structure, but…
  - … the user experience is altered substantially
- Drilling becomes a challenge
  - Native drilling works on flattened or snowflake
  - OpenDocument() works on ancestor and tree

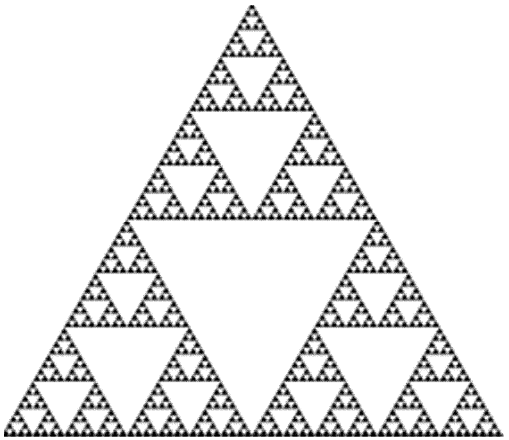
Questions and Discussion

- This presentation will be posted on my blog after the conference
  - As time permits I will provide slide narration as blog posts

- Author Information
  - David G. Rathbun
  - BI Solutions Architect, PepsiCo B+IS
  - BI Blog www.dagira.com

Supporting Links

- Interesting Wikipedia Links
  - http://en.wikipedia.org/wiki/Recursion
  - http://en.wikipedia.org/wiki/Tree_traversal
- Other links
  - http://articles.sitepoint.com/article/hierarchical-data-database/1



2009 SAP BusinessObjects USER CONFERENCE

Powered by the Global BusinessObjects Network

Thank you for participating

Please remember to complete and return your evaluation form following this session.

SESSION CODE: 1012