2009 SAP BusinessObjects USER CONFERENCE

Powered by the Global BusinessObjects Network

Universe Best Practices

Session Code: 806

Alan Mayer

Solid Ground Technologies, Inc.

Agenda

- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Introduction

Agenda

- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Ground Rules

- Focus on your audience
  - **Who** will be using your universe?
    - People (End users, Analysts, IT Professionals, …)
    - Applications (Deski, Webi, Crystal Reports, …)
  - **How** will it be used?
    - Retrieve detailed information
    - Discover trends over time
  - **What** will be its primary purpose?
    - Provide information for a department / business sector
    - Act as reporting interface for an application
    - Allow data access across databases, applications

Ground Rules

- Users drive size and complexity
  - End users and analysts require smaller universes
    - Reduce the number of classes and objects
    - Create smaller universes but more of them
    - Structure must be vetted to reduce/eliminate user errors
  - IT professionals can work with larger universes
    - Often more complicated
    - Used to create canned reports or SQL for other purposes
  - Applications have their own requirements
    - Pre-determined SQL statements created
    - Universe structure different than ad-hoc access
    - Tuning constructs can be safely added that speeds retrieval

Ground Rules

- Tools that use universe data matter …
  - Each has its own requirements and limitations
    - Desktop Intelligence
    - Web Intelligence
    - Crystal Reports
    - Live Office
    - Xcelsius
    - Third-party applications via web services

Ground Rules

- In general …
  - Keep the number of objects to 700 - 800
    - Larger universes will require more memory to use
    - This means more Java runtime memory allocated for Web Intelligence users
  - Reduce complexity where possible
  - Maximize your investment
    - Focus your universe efforts
    - Determine how this universe will work with others
    - Implement this universe as one piece of an overall strategic solution
  - Minimize your maintenance

Agenda

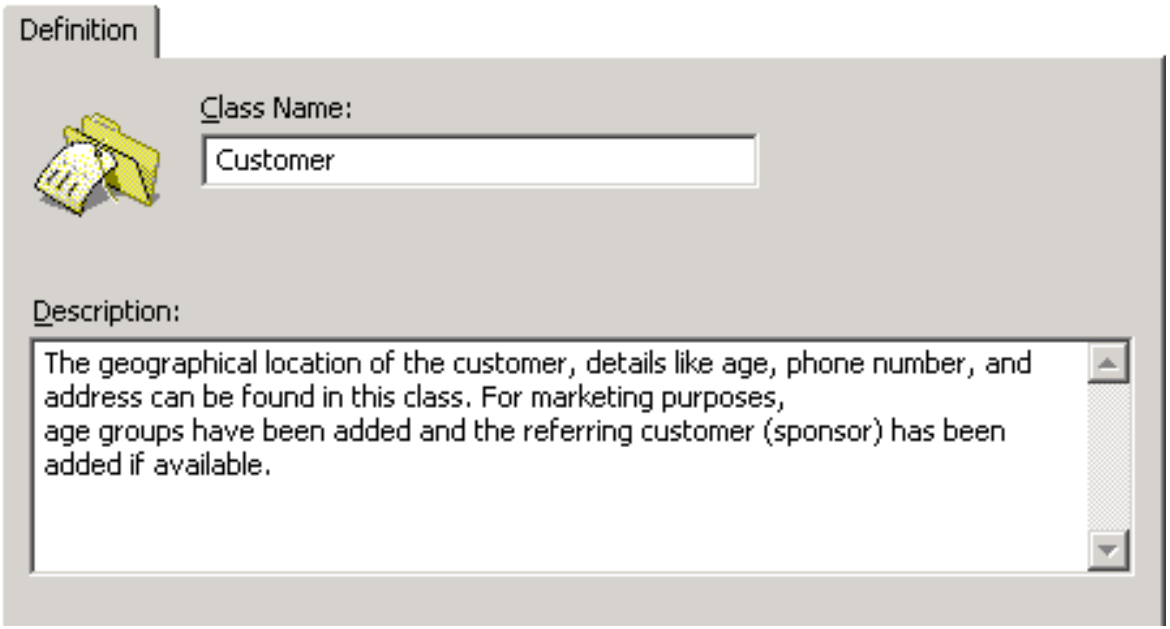
- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Classes

- Classes group logically related business terms (objects) together
- Best practices for classes include:
  - Naming conventions
  - Descriptions
  - Layout
  - Nesting limits (classes within classes)

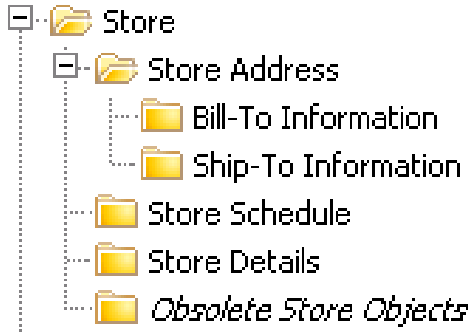
Classes

- Naming Conventions and Descriptions
  - Stick to a reasonable limit for the name (60 chars)
  - Descriptions can be long – be as descriptive as possible
    - How objects can be used
    - Any special filters on this particular class



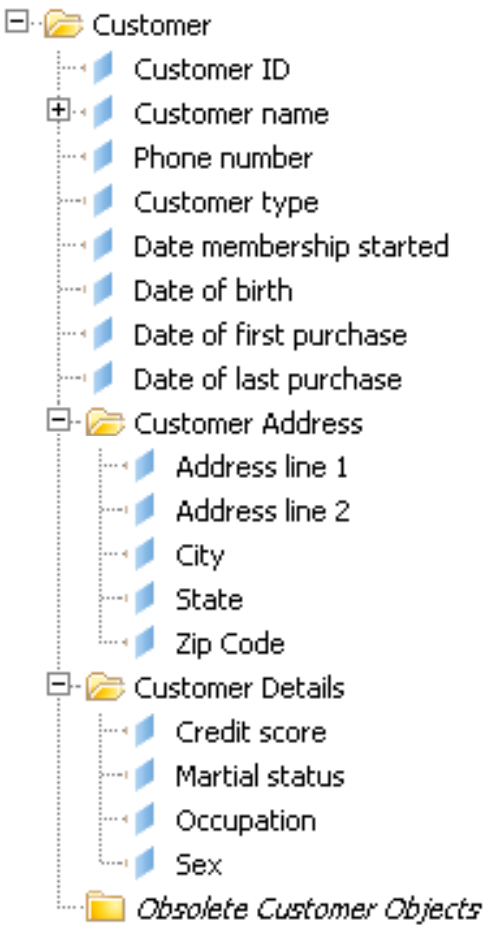
Classes

- Layout
  - Let users drive the names of classes
    - Class names must be unique
    - Classes can be used to separate lesser used objects
  - Control the level of nesting
    - Nesting refers to classes within classes
    - Most companies use 4 levels of nesting maximum
    - Deeper levels may make objects harder to locate
  - Add a hidden class for obsolete objects
    - Removing them could invalidate reports



Classes

- Layout
  - Limit objects per class to 20 – 25 if possible
    - This will reduce scrolling through long lists
    - Use subclasses and detail objects to make this a reality
  - Determine how objects will be listed
    - Most commonly used is most popular
    - Alternatives:
      - Alphanumeric
      - Order by type (dates, calculations, …)
      - Hierarchically (general to specific)
      - Fastest to execute when placed in conditions



Objects

- Objects are business terms that users retrieve as data
- Best practices for objects include rules for:
  - Naming conventions and descriptions
  - Object type
  - Object SQL
  - Calculations
  - Hidden objects
  - List of values
  - Relative objects
  - Object formatting
  - Conditions / filters
  - Linking / Merging

Objects

- Naming conventions
  - Decide on a reasonable limit for object names (60 chars)
  - Consistently format names
    - Capitalize first letter of the name or every word
    - Signify embedded prompts by appending special chars (‘?’, …)
    - Show objects that are flags (TRUE/FALSE, 1/0) by appending ‘Flag’ or some type of indicator

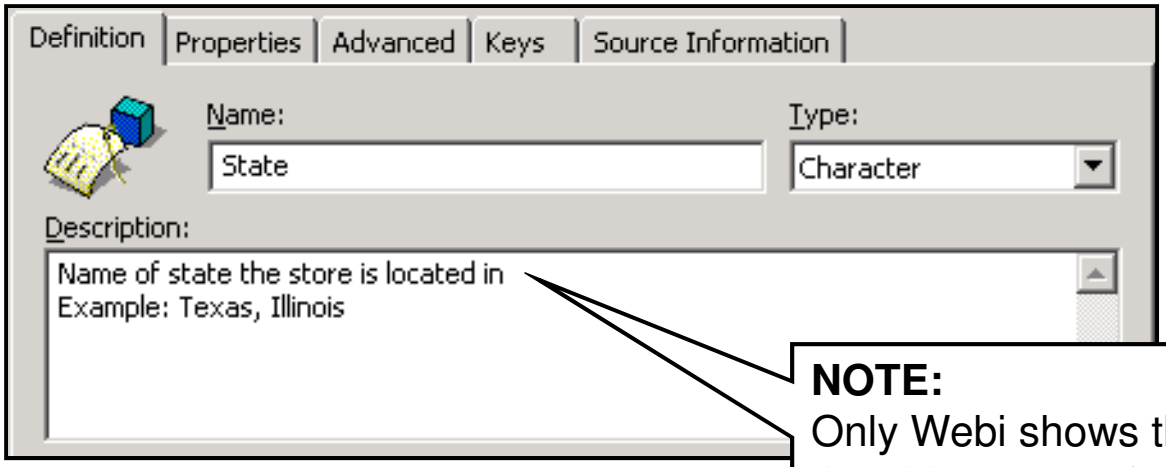
Name	Explanation
Customer Name	Full name (Last, first, middle initial)
Store?	Prompts for store name with LOV
Europe Flag	Returns 1 if European txn, 0 otherwise

Objects

- Naming conventions, cont'd
  - Why not prefix the class name in front of every object?
    - Customer last name
    - Customer first name
    - …
  - Names to not have to be unique
  - Certain tools like Webi now display the class location for every object automatically
  - If using other tools, it might pay to make the name more descriptive

Objects

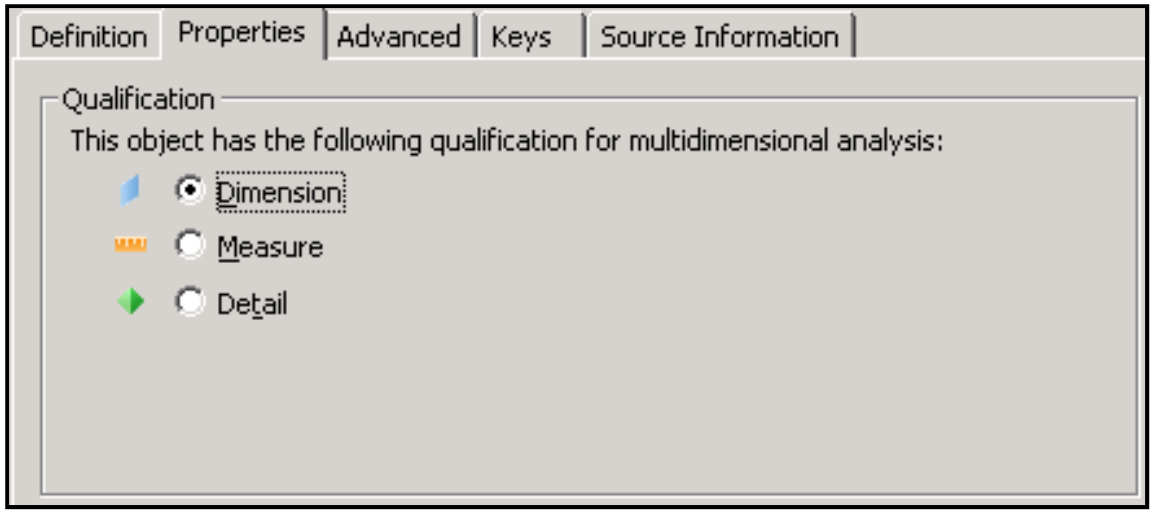
- Descriptions
  - Add help text for EVERY object
    - Add a description then several examples
    - Add format masks (MM/DD/YY) on the first line
    - Optional: Add class location for the object



NOTE:
Only Webi shows the full class path the object came from.

Objects

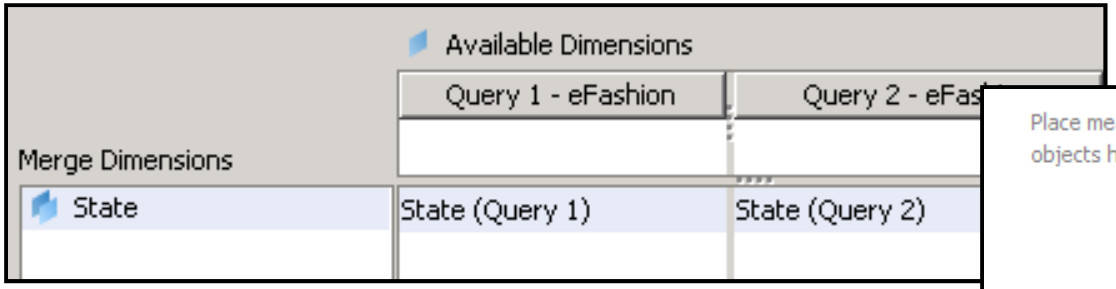
- Object Type
  - Should the object be a dimension, detail, or measure?
    - **Dimension**: Key fact that drives the remainder of the query
    - **Detail**: Additional information that depends on existing dimension
    - **Measure**: Calculation
  - Biggest point of confusion: Dimension or detail?
    - More on this in a moment …



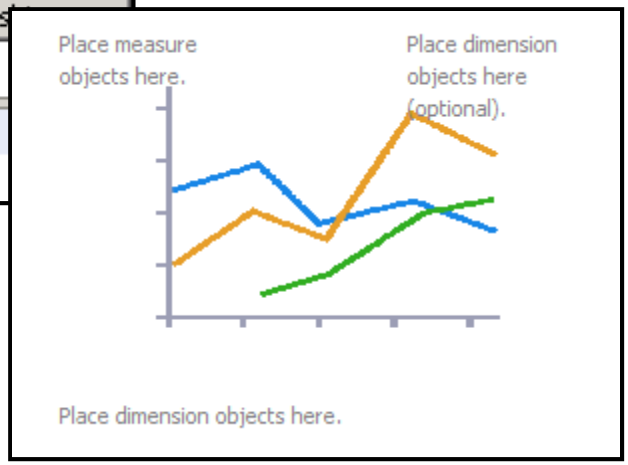
Objects

- Object Type
  - Report functionality depends on object type
    - Hierarchies consist of dimension objects only
    - Query linking (merged dimensions) depend on linked dimensions
    - Report writers like Deski and Webi require measures

Merged Dimensions

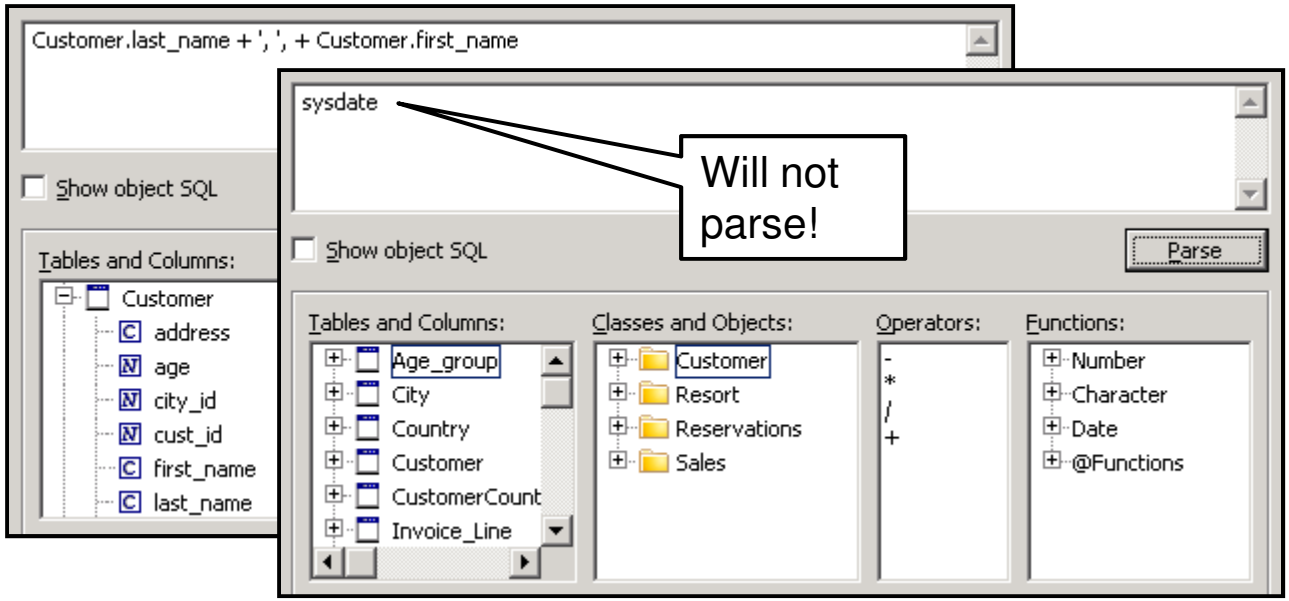


Graphs



Objects

- Object SQL
  - Use the SELECT clause editor to select tables/columns
    - This will help avoid silly spelling errors
  - Always parse objects!
    - Not all objects will parse.
    - For example, any object not based on a table ('Today')



Objects

- Complicated SQL
  - Build the desired object in layers
  - Create objects that will be referenced using @SELECT
  - In this way, very complicated SQL expressions can be created

Europe Flag

```
decode(   CustomerCountry.country,
          'Holland', 1, 'Germany', 1, 'UK', 1, 'France', 1, 0)
```

2000 Flag

```
decode( to_char(Sales.invoice_date, 'YYYY'), '2000', 1, 0)
```

Europe 2000 Revenue

```
Sum(  @Select(Resort\Europe Flag) * @Select(Sales\2000 Flag) *
      Invoice_Line.days * Invoice_Line.nb_guests * Service.price )
```

Objects

- The WHERE Clause
  - Avoid adding SQL in the WHERE clause of any object
  - This is especially true for ad-hoc universes
  - Report writers will combine those conditions using '**AND**'

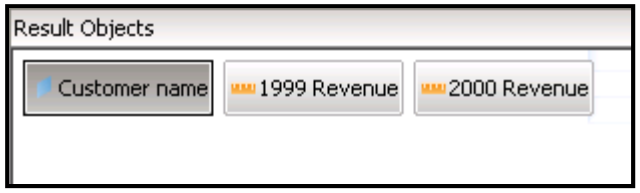
1999 Revenue

```
WHERE  to_char(Sales.invoice_date,'YYYY') = '1999'
```

2000 Revenue

```
WHERE  to_char(Sales.invoice_date,'YYYY') = '2000'
```

Final Query




```
WHERE
to_char(Sales.invoice_date,'YYYY') = '1999'
AND
to_char(Sales.invoice_date,'YYYY') = '2000'
```

Objects

- WHERE Clause, cont'd
  - Use DECODE or CASE logic in the SELECT clause instead
  - Our flag logic presented earlier works well here
    - … plus the yearly test is reusable!

```
SELECT
  sum(
    decode( to_char(Sales.invoice_date, 'YYYY'), '1999', 1, 0) *
    Invoice_Line.days * Invoice_Line.nb_guests * Service.price )
```

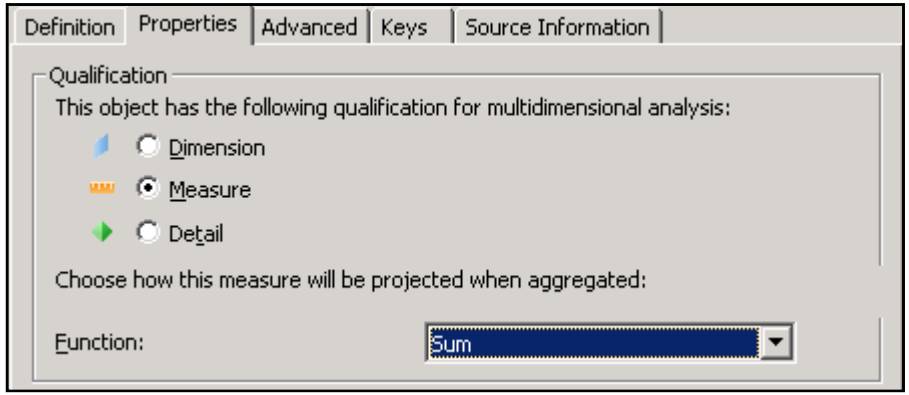
  - Condition objects could also be used
    - Users can change **AND** to **OR** in the query panel

Objects

- Calculations
  - Calculations are performed by measures
  - In general, an aggregate function should be used
    - These include SUM, COUNT, MIN, MAX, AVG
    - This forces the aggregation to occur on the database server
  - Certain ratios (a/b) should be created by distributing the functions
    - SUM(a)/SUM(b) rather than SUM(a/b)
    - This allows the calculation to cover the group, not just the transaction
  - Count using the DISTINCT keyword
    - COUNT(DISTINCT <indexed column>)

Objects

- Calculations Projections
  - Projections control how Deski and Webi work with measures
    - Specifies how measures will be aggregated AFTER data is returned
  - The projection for COUNT is usually SUM

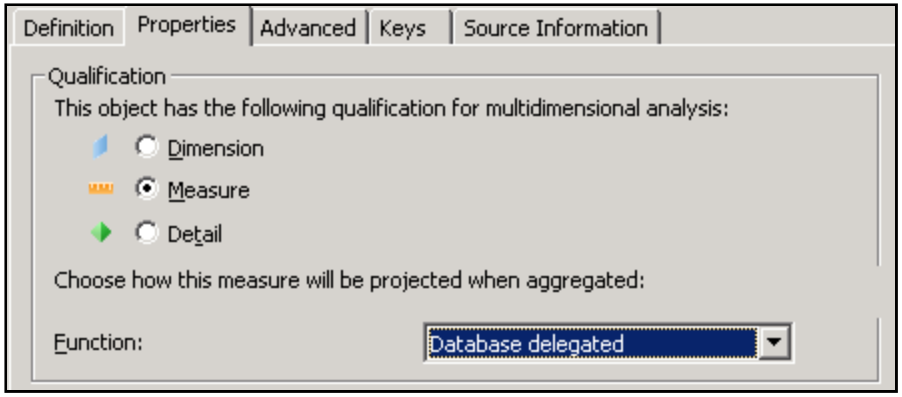


Country	Resort	Revenue
France	French Riviera	835,420
US	Bahamas Beach	971,444
US	Hawaiian Club	1,479,660

Country	Revenue
France	835,420
US	2,451,104

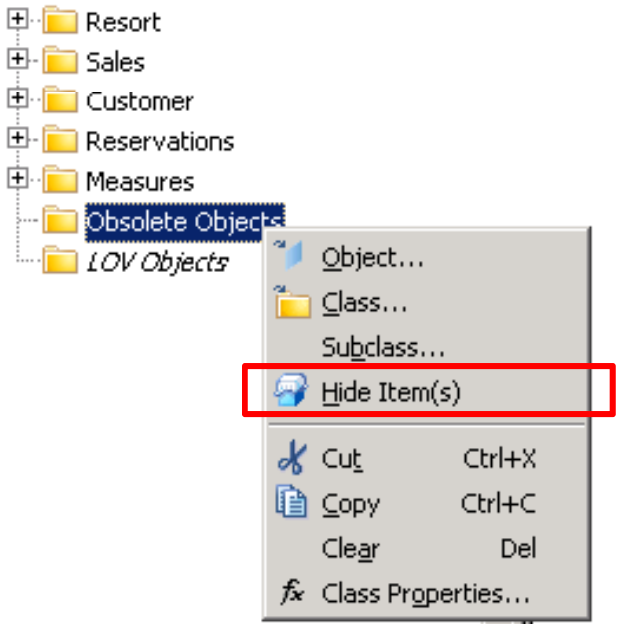
Objects

- Calculation Projections, cont'd
  - Use the Delegated Measure feature for AVG, %
    - This forces the report writer to re-run SQL every time dimensions or details within the block change
    - This prevents incorrect calculations
    - Can't automatically calculate the average of an average



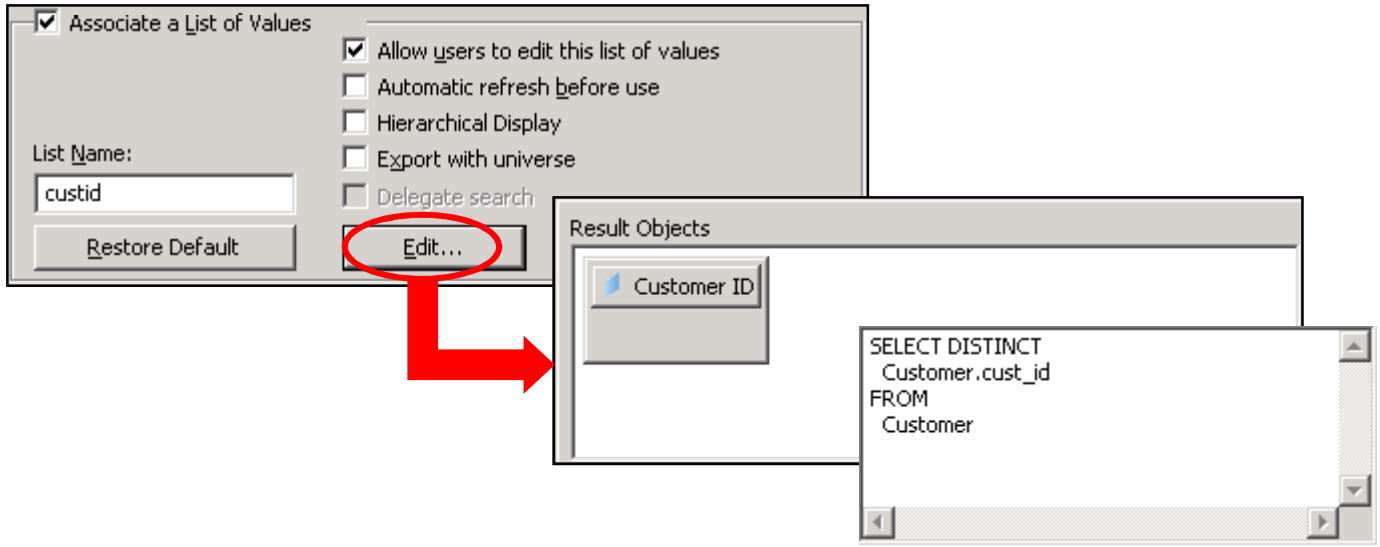
Objects

- Hidden Objects / Classes
  - Hide objects / classes that are obsolete
  - Extremely useful technique for creating more complicated objects
    - Can also be used to accelerate List of Value queries



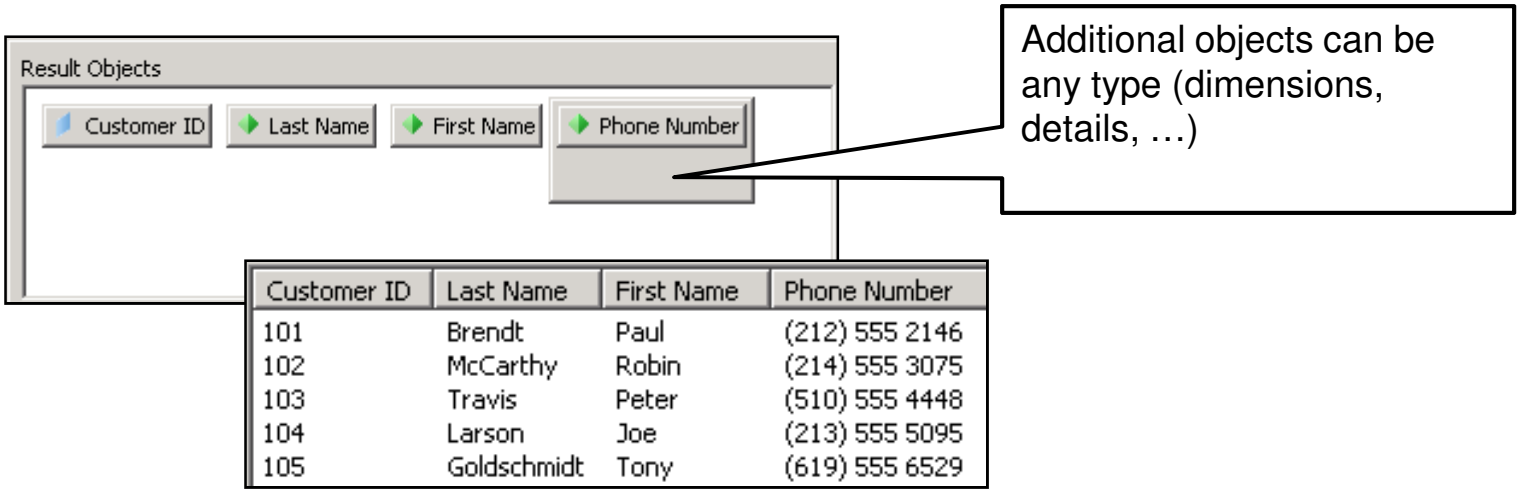
Objects

- List of Values
  - These lists allow users to complete a query condition
  - Default LOV queries are not very informative
    - SELECT DISTINCT <object SQL>
  - Alter that SQL query to include codes and descriptions



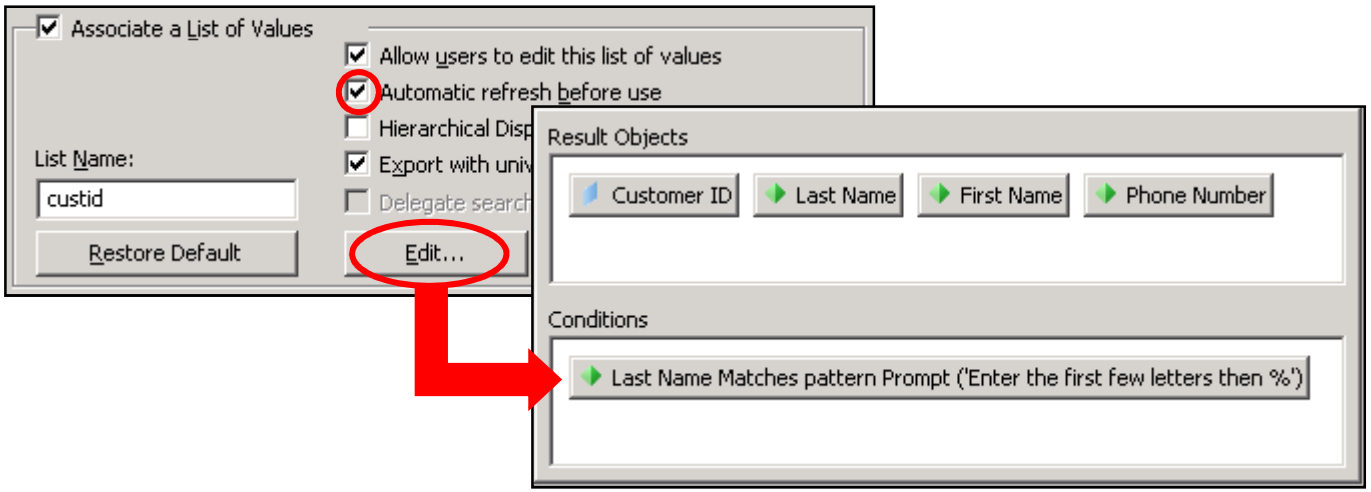
Objects

- List of Values, cont'd
  - Additional objects can be added to the LOV query
    - This may assist some users in selected the correct value
    - Only the left-most column is returned as the value



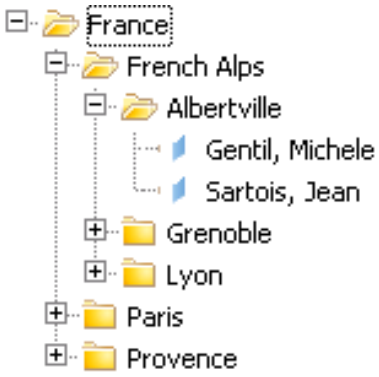
Objects

- List of Values, cont'd
  - Conditions can also be added to further refine possible values
    - Embedded prompts are popular to reduce long lists (1000 or more)
    - Pattern matching can be used to reduce the list further
    - Make sure to automatically refresh LOV queries with prompts



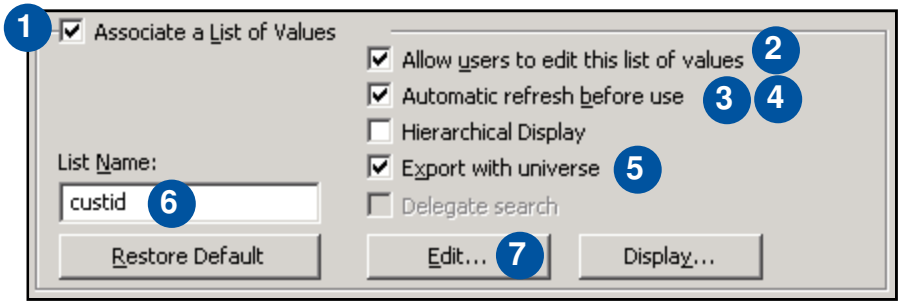
Objects

- List of Values, cont'd
  - Hierarchical LOV queries
    - LOV results can be displayed in list or hierarchical format
    - If the latter is desired, arrange LOV objects in drilled order
      - Left-most object is returned as final value
      - Next object would represent the top of the hierarchy
      - Third object would server as the second hierarchical level
      - Second through the last object should be sorted



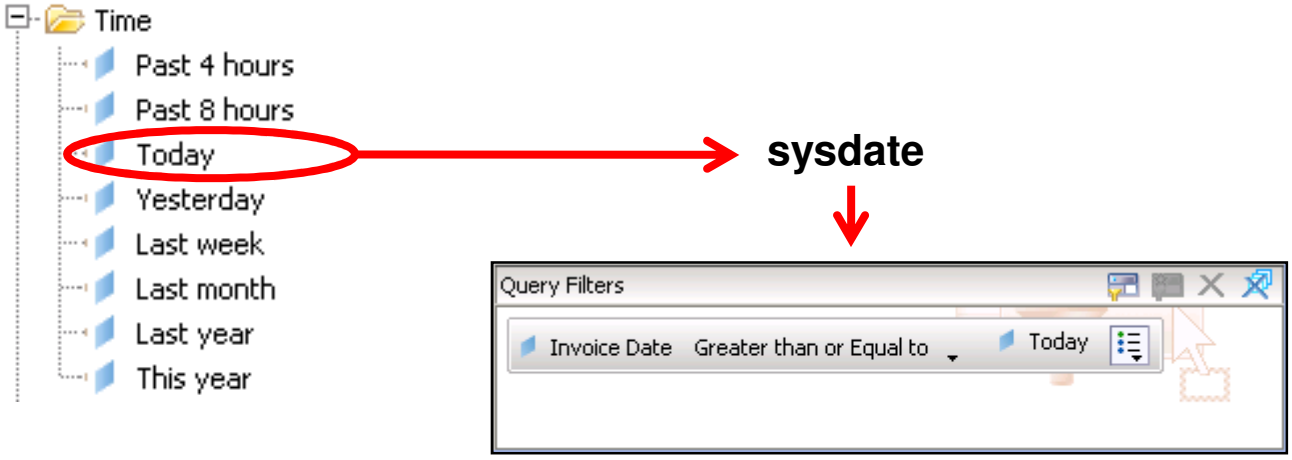
Objects

- List of Values Best Practices
  1. Don’t maintain list of values for dates, calculations
  2. Most users are not allowed to edit their List of Values
  3. Always refresh a list that includes a prompted condition
  4. Don’t refresh a list that is relatively static
  5. Always export customized list of values
  6. Name a customized LOV query (other objects can reuse it)
  7. Except for static lists, don’t save data with the L OV queries



Objects

- Relative Objects
  - These objects retrieve values based on a point in time
  - Usually not based on physical tables
  - Great for scheduled reports whose conditions change over time
  - Be careful with time (HH:MI:SS) vs. dates (MM-DD-YYYY)
  - These objects can be dimensions, details, or condition objects
    - Advantage as dimension: Can use to complete ANY query condition



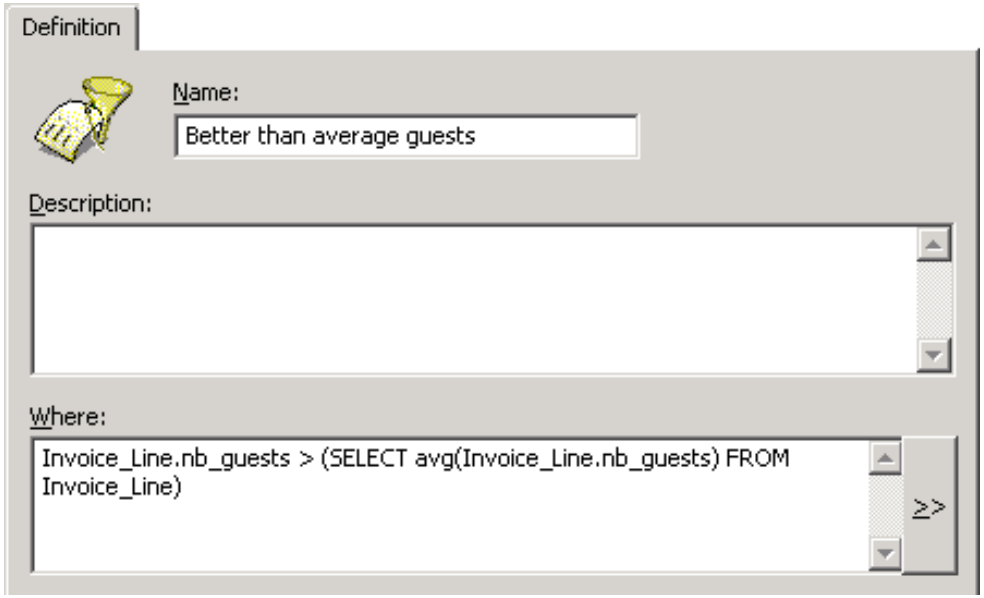
Objects

- Formatting
  - Formatting the way objects appear within a report saves time
    - Format once in the universe rather than once per report

Datatype	Formatting Mask
Number (Integer)	0
Number (Count)	Positive: #,##0 Negative: (#,##0) Zero: Blank
Currency	Positive: $#,##0.00 or #,##0.00 Negative: (#,##0.00) Zero: Blank Note: Place a dollar sign ($) on all subtotals and grand totals. Skip the dollar sign for detailed currency values

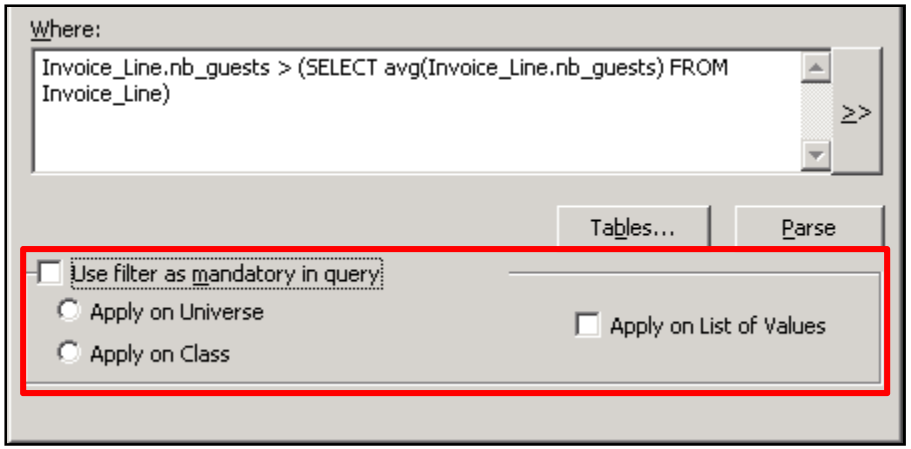
Objects

- Condition Objects
  - Condition objects act as pre-programmed query filters
  - Great for frequently used and difficult conditions
    - Subqueries, correlated subqueries
  - Once created, users can combine in a query using **AND**, **OR**



Objects

- Condition Objects, cont'd
  - Conditions can now be added to classes
  - Every object inside the class inherits the condition
  - Different from security restrictions – not based on a group or user
  - Much better than trying to restrict objects based on implicated tables



Objects

- Query Linking
  - Queries can be combined in Deski or Webi
    - This is done by linking/merging dimensions
    - The dimensions can come from different universes
  - A few rules must be followed for this technique to work:
    - The data returned by linked dimensions must be identical
      - Different formats will not work!
    - Object names can be different
      - Not the best course of action
      - Users may have trouble finding dimensions to link

Objects

- Query Linking, cont'd
  - The resulting report block can contain:
    - Linked dimensions
    - Details of linked dimensions
    - Measures
  - Unlinked dimensions or details of unlinked dimensions can never (reliably) be added

Correct

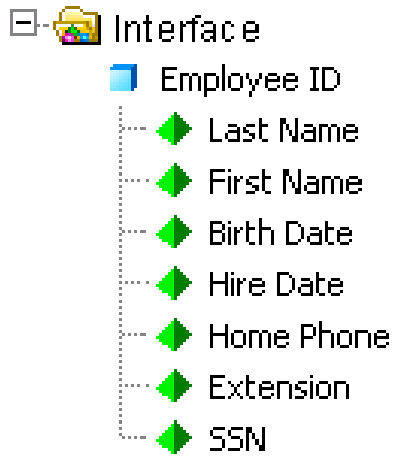
Country	Number of guests	Future guests
France	446	46
US	1,105	56

Incorrect

Country	Resort	Number of guests	Future guests
France	French Riviera	446	46
US	Bahamas Beach	565	56
US	Hawaiian Club	540	56

Objects

- Query Linking, cont'd
  - Add interface classes to your universe to simplify linking
    - Users quickly adapt to looking for these classes
    - Results are accurate and reliable
  - This will also drive your object type decisions
    - Dimension vs. detail becomes much clearer

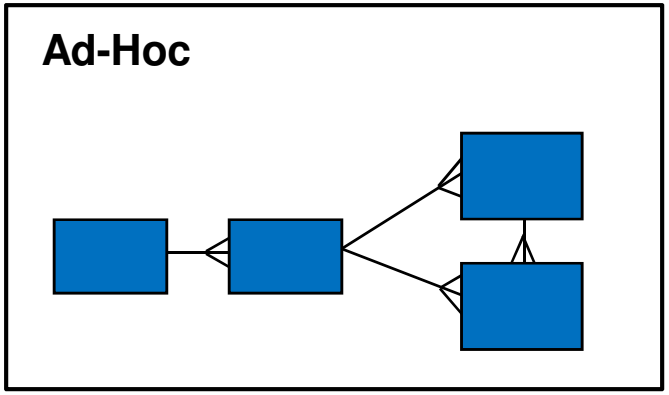


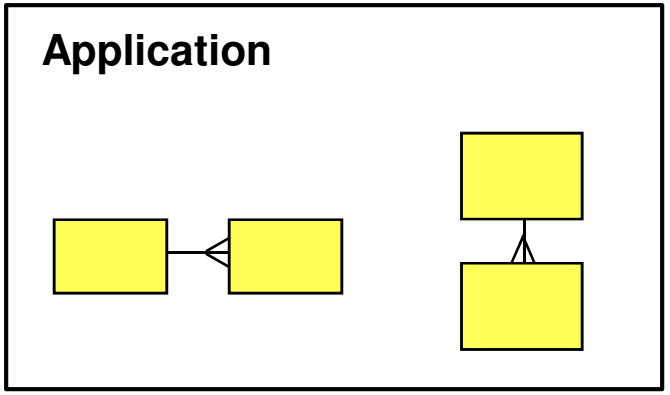
Agenda

- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Joins

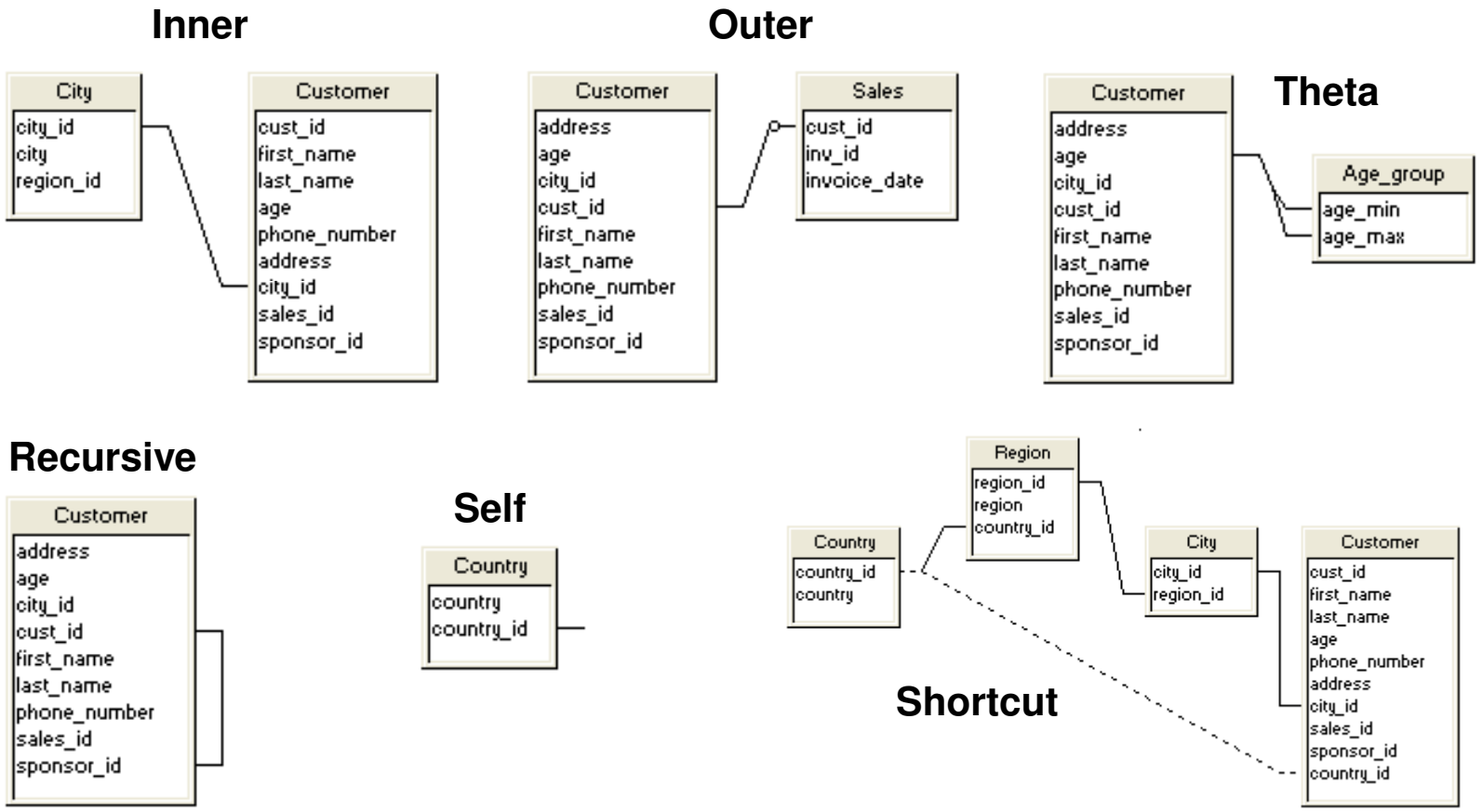
- Join strategy depends on how this universe will be used
  - Ad-hoc universes require most tables to be joined
    - Exception: Keeping tables that are aliased elsewhere
    - Prevents Cartesian products
  - Universes that feed dashboards and apps are different
    - “Clusters” of joined tables are acceptable
    - Queries are pre-programmed by developers





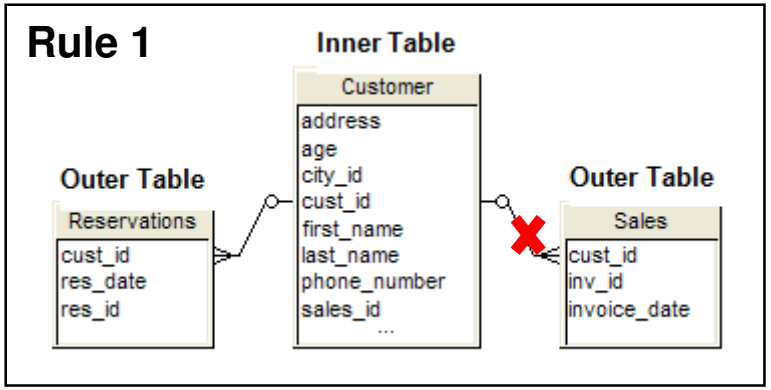
Joins

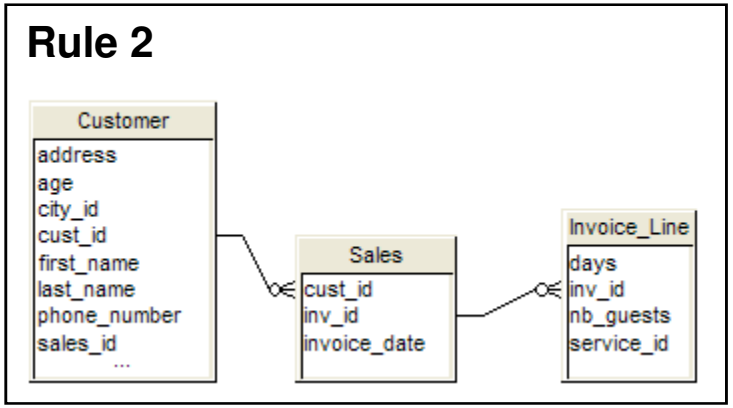
- Many different types of joins are available



Joins

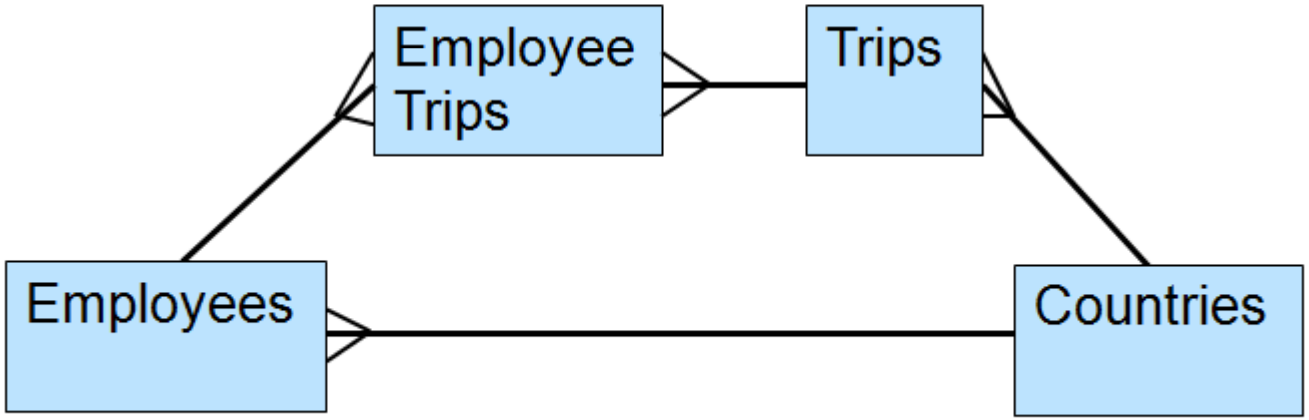
- Outer joins have special considerations
  - Not the best performing join
  - Two rules that are forced by SQL:
    1. Inner table of an outer join cannot be used as the inner table of another outer join
    2. Outer joins must cascade





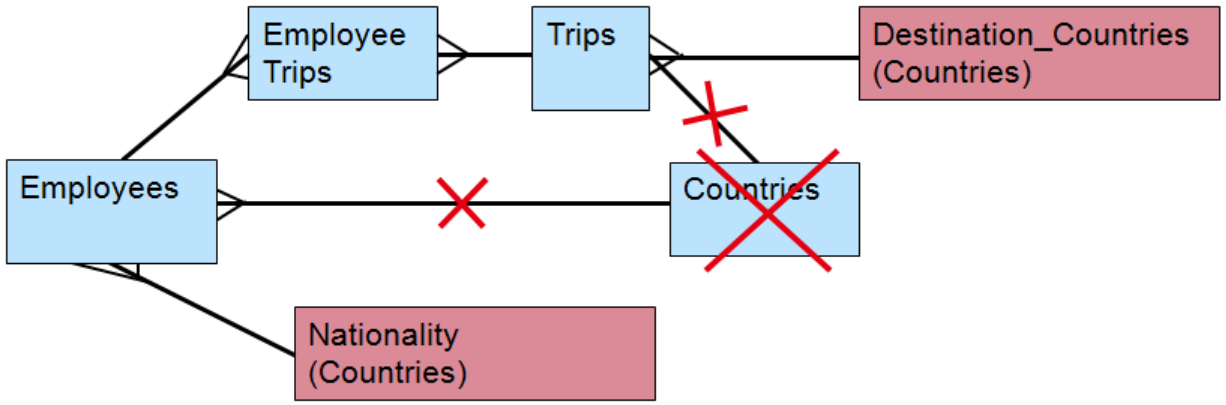
Joins

- Loops
  - Two or more paths between tables
  - Developers must resolve loops to allow users full query access



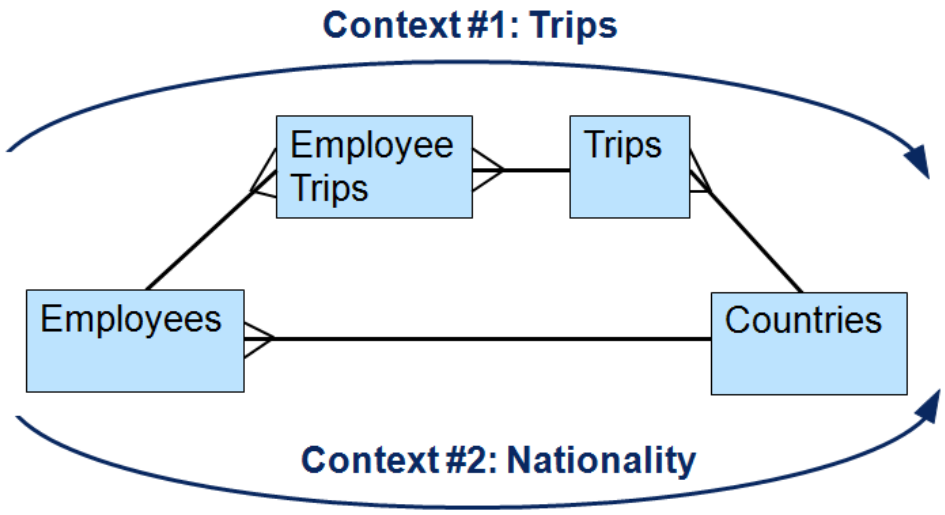
Joins

- Aliases
  - One method to resolve loops
  - Creates a logical copy of a table to be used to break the loop
    - Breaks the loop at design time
  - Helpful naming convention
    - Capitalize the first letter of every word



Joins

- Contexts
  - Second method for resolving loops
  - Lists the paths between tables
  - Worst case – user asked to choose between paths
  - Best case – path is inferred
  - Loop is resolved at query run-time



Joins

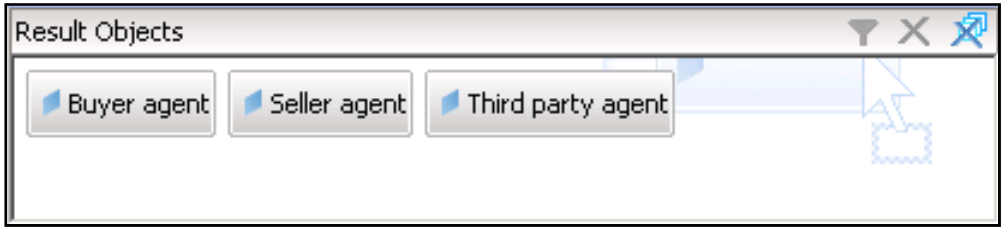
- Aliases vs. Context Comparison

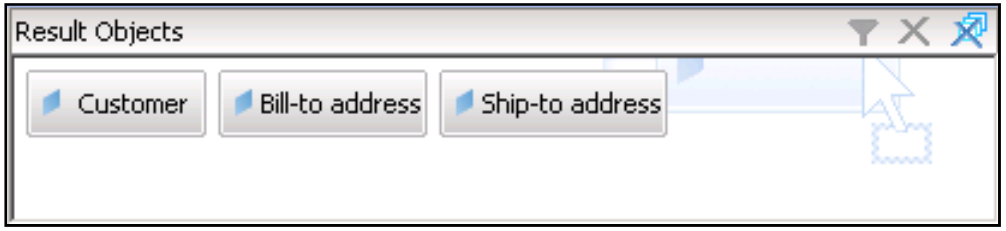
Aliases	Contexts
Resolves loop at design time	Resolves loop when query is run
Creates more objects	No additional objects added
Aliases cascade	Context selection may be forced on user
	Every join must be part of one context

- Which method is better?
  - It depends on the situation
  - More advice in a minute …

Joins

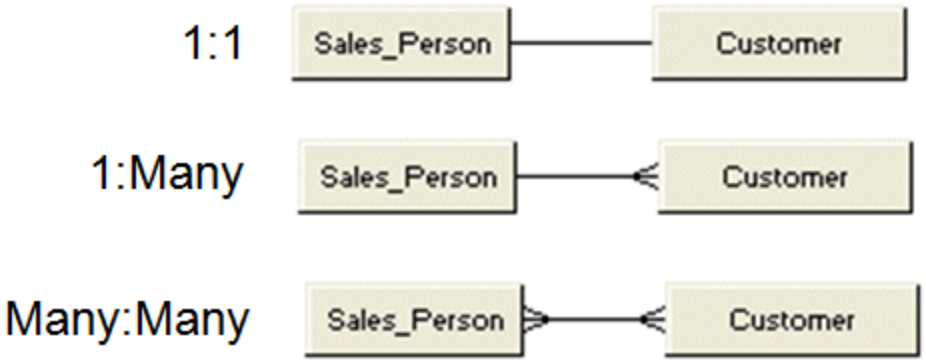
- ACID Test for Aliases
  - Place all objects created from aliases in a query
  - Would this make sense to a user?
    - If so, aliases must be used to simultaneously represent values
  - Aliases used to resolve chasm traps, lookup tables





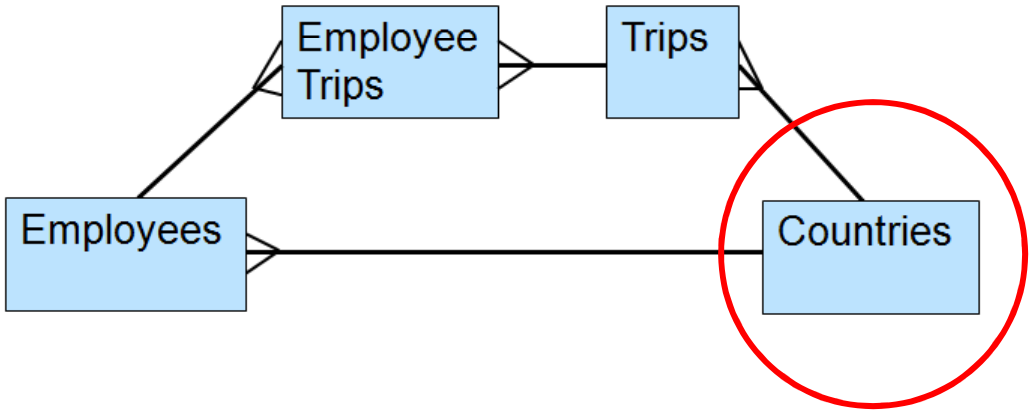
Joins

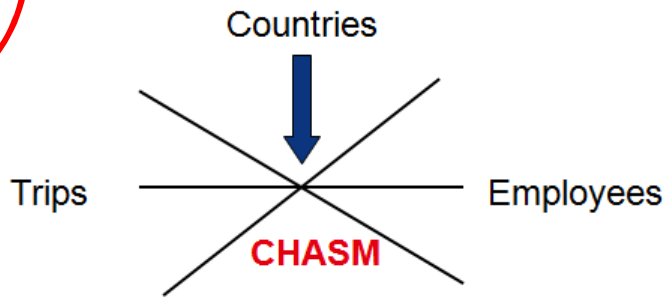
- Cardinalities
  - Determines the number of values joined between tables
    - One to one
    - One to many
    - Many to many
  - **ALWAYS** set the cardinalities for every join
  - **NEVER** depend on automatic cardinality detection
    - The algorithm used is not 100% accurate



Joins

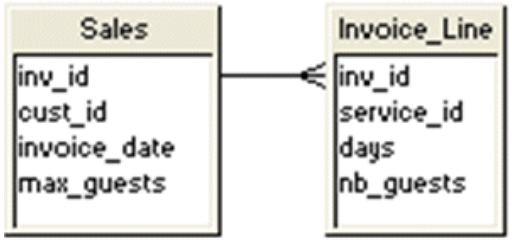
- Logical Trap #1 – Chasm Traps
  - Many to one to many relationship
  - No relationship from left to right
  - Usually resolved with aliases





Joins

- Logical Trap #2 – Fan Traps
  - One to many to many relationships
    - Also known as master-detail relationships
  - Trouble when aggregating on the master side
  - Several ways of resolving fan traps
    - Don't aggregate master columns
    - Use contexts to provide master and detail paths



Result Set

Invoice	Budgeted Guests	Actual Guests
23102	10	3
23102	10	4
Totals:	20	7

Agenda

- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Hierarchies

- Hierarchies allow Deski and Webi users to drill
  - Consist entirely of dimensions
  - Can reflect natural hierarchies
    - Time (Year > Quarter > Month > Week)
    - Organizational (Corporate > Region > Division > …)
- Two best practices for hierarchies
  - Create custom vs. default hierarchies
    - Much easier to control what users drill on
    - Avoids nonsensical drills (Last Name → First Name)
  - Order hierarchies from best to worst
    - If two hierarchies can be used to drill, the top-most hierarchy will be chosen

Agenda

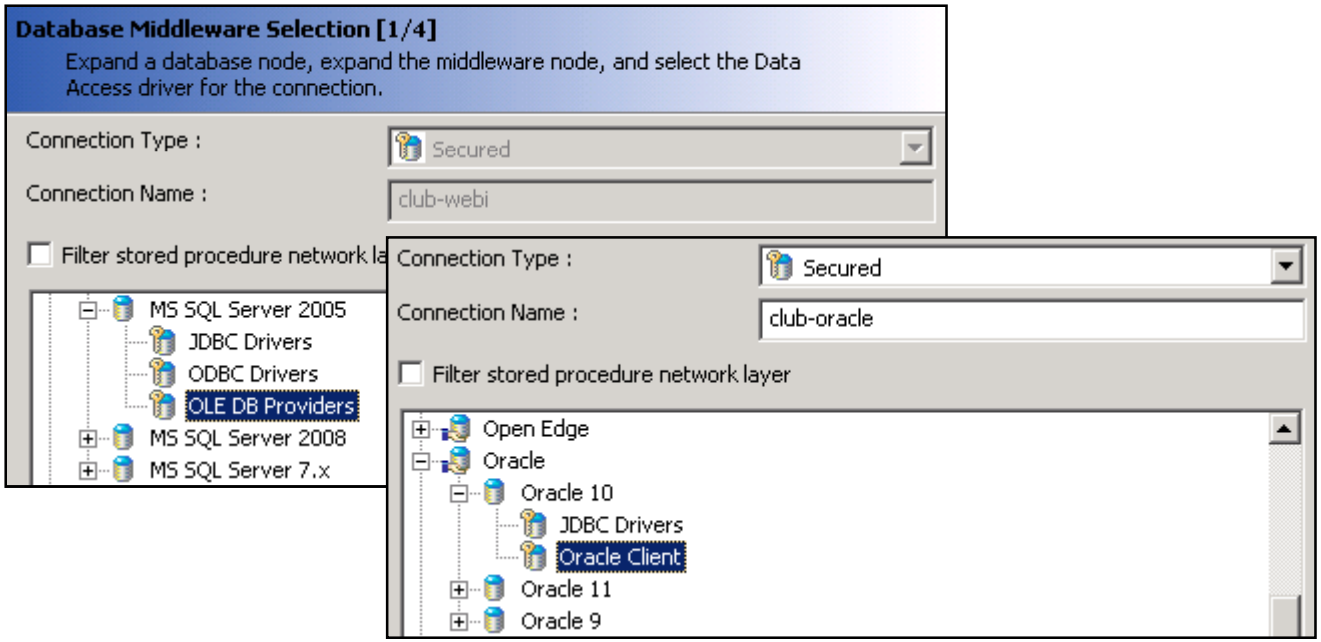
- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Universe Parameters

- These are controls set once per universe
  - Database connection
  - Summary information
  - Strategies
  - Query Limits
  - SQL Limits
  - Dynamic parameters

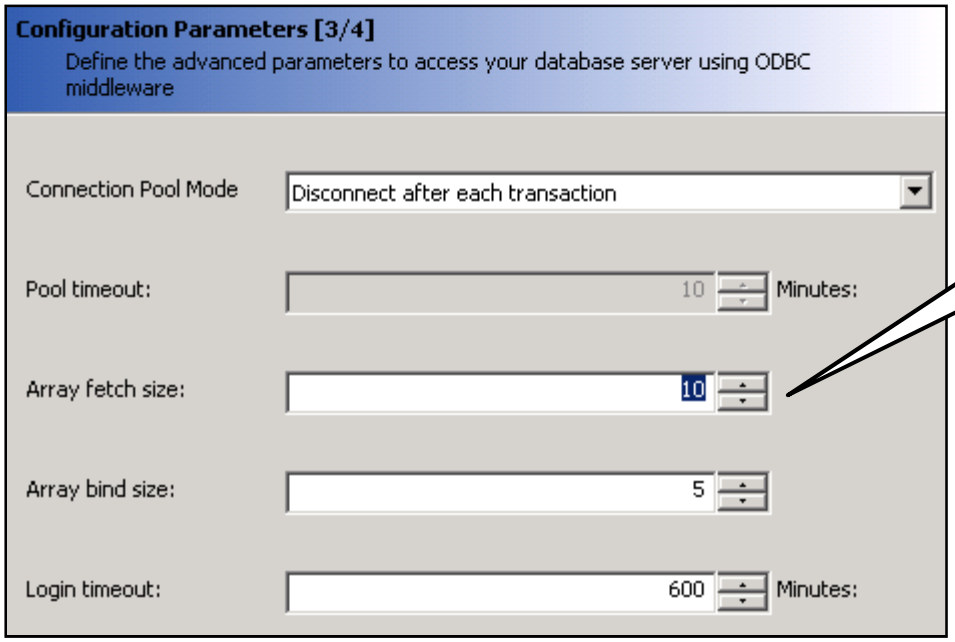
Universe Parameters

- Database Connection
  - Avoid ODBC connections where possible
    - Opt for OLE-DB or superior technologies like native drivers



Universe Parameters

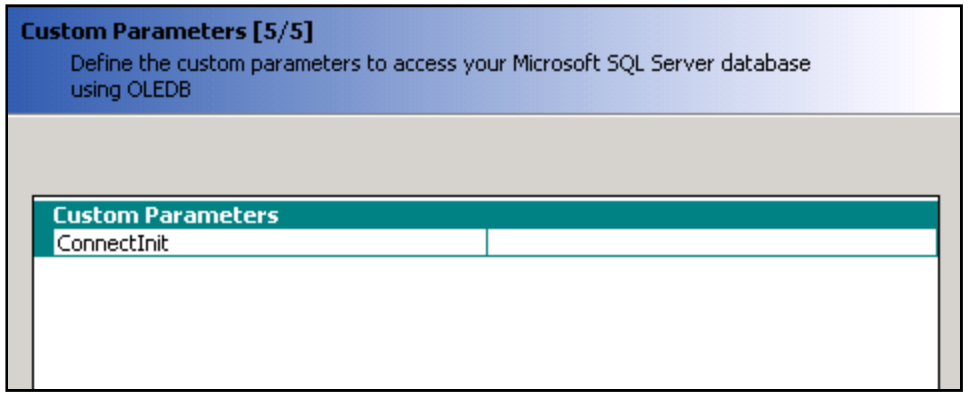
- Database Connection, cont'd
  - Disconnecting after each transaction is safest
  - Increase **Array fetch size** to accelerate data retrieval



Default is 10, but this can be increased up to 1000 rows per fetch

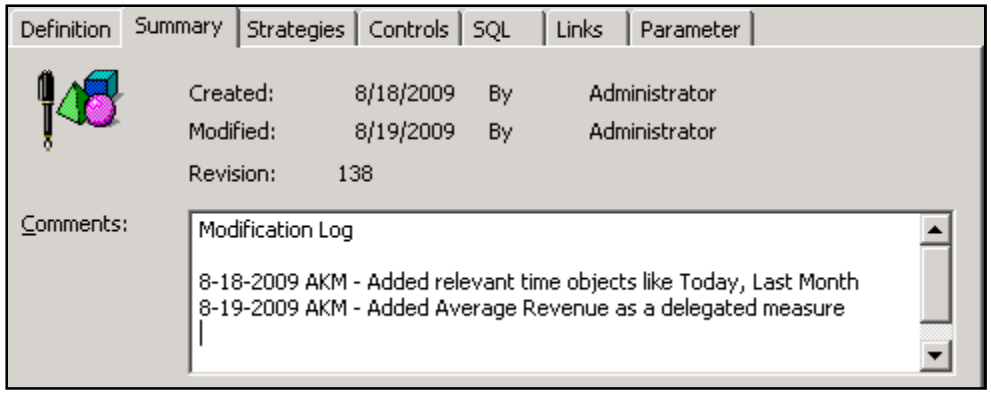
Universe Parameters

- Database Connection, cont'd
  - Custom parameters can be selectively added
    - Highly dependent on database
  - Hints can be added for certain databases (Oracle)
    - Especially desireable for data marts
    - Custom parameter = Hint
    - Value = /*+ STAR_TRANSFORMATION */



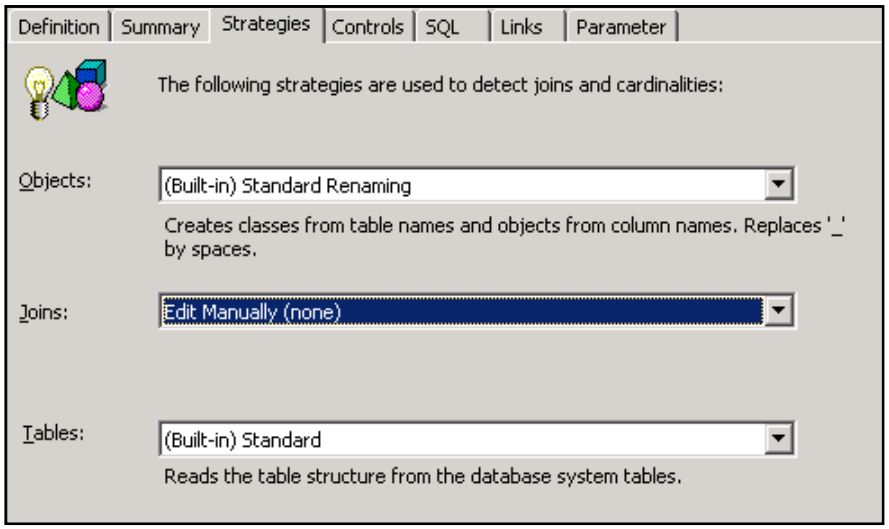
Universe Parameters

- Summary information
  - Use the Comments section to add designer notes
    - Just like a programmer's header block
    - Can also use as an incremental modification log



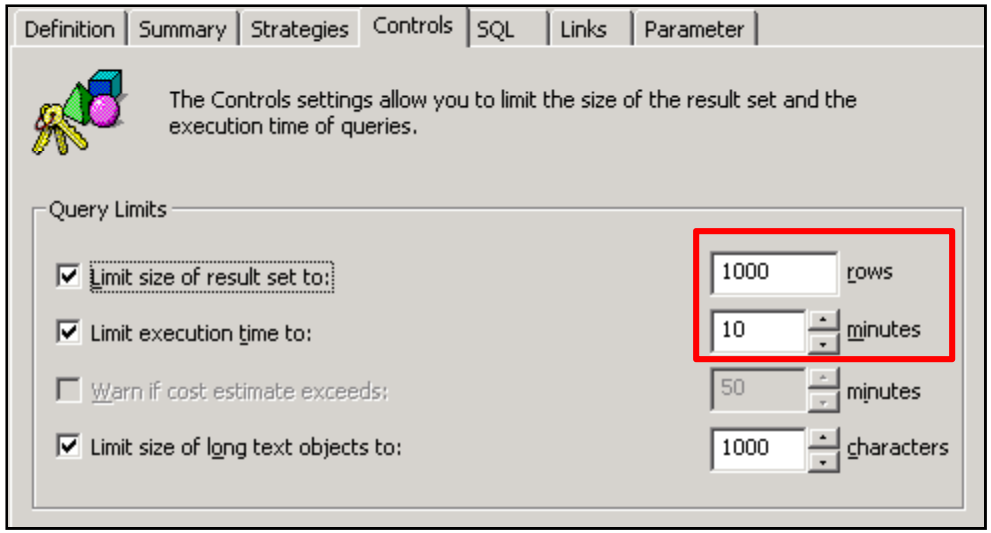
Universe Parameters

- Strategies
  - These are scripts that determine how tables, objects, and joins are automatically selected
  - Creating your own strategies for tasks like:
    - Selecting a smaller sample of tables in the Table Browser
    - Represent certain database objects (public synonyms)



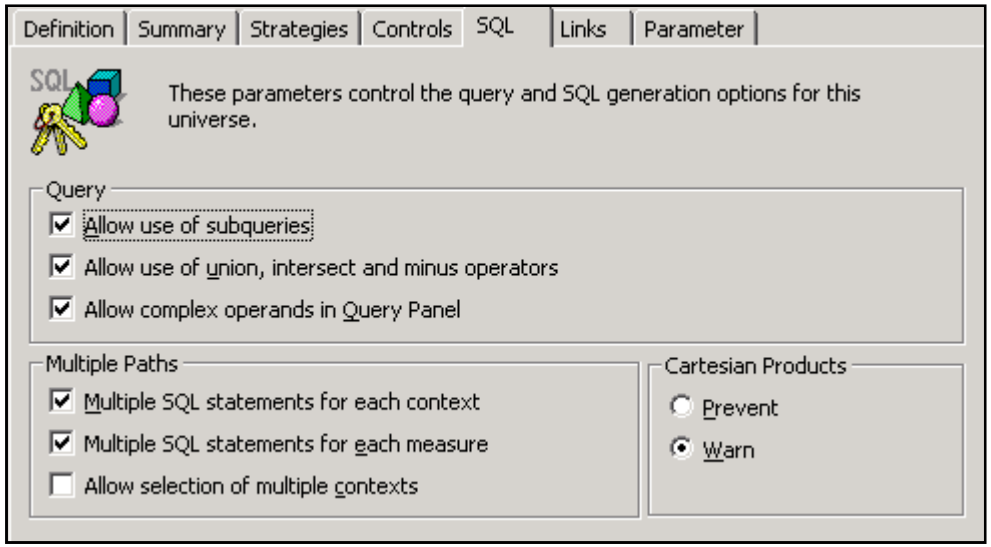
Universe Parameters

- Query Limits
  - These limits become default values for your universe
  - The first two (rows, time) are the most important



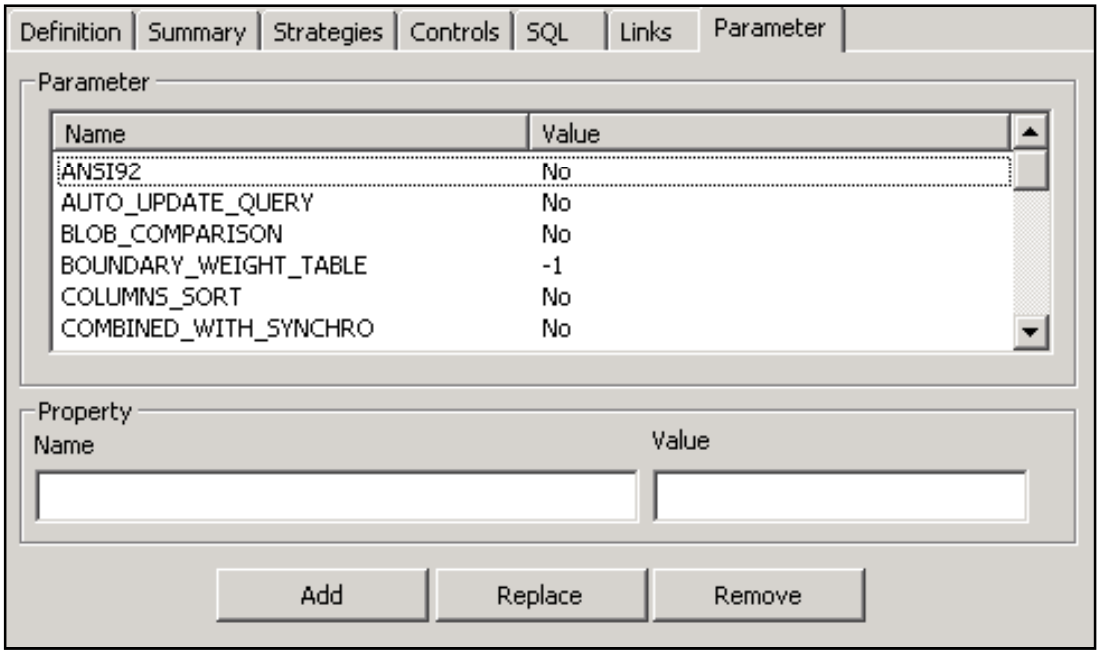
Universe Parameters

- SQL Parameters
  - Multiple Path options are the most important
    - They control the creation of multiple SELECT statements
    - This will help with incorrect aggregation issues
  - Cartesian Products should be set to Prevent for ad-hoc universes



Universe Parameters

- Dynamic Parameters
  - These parameters can expand or limit a universe's functionality



Universe Parameters

- Dynamic Parameters, cont'd
  - Some of the more important candidates:
    - **ANSI92** Follows the ANSI-92 convention for joins in the FROM clause. Allows full outer joins.
    - **JOIN_BY_SQL** Allows multi-pass SQL processing to work in tools like Crystal Reports. UNIONS the multiple SELECTS
    - **END_SQL** Allows comments to be added at the end of every SELECT statement. DBAs can use to find the associated universe and user

Agenda

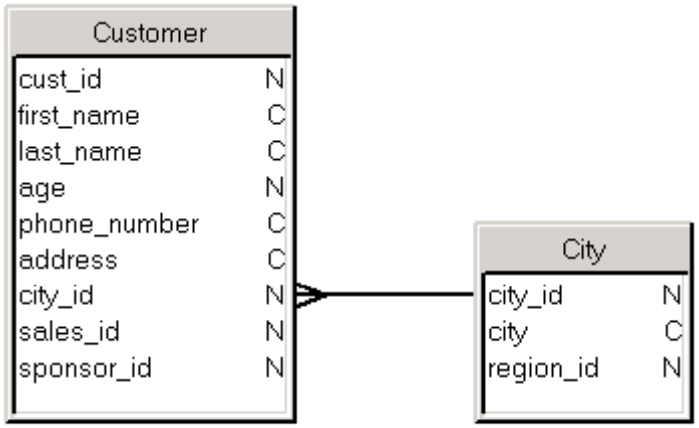
- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Performance

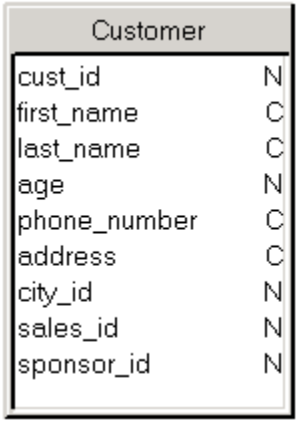
- There are several techniques available for accelerating query performance:
  - Index Awareness
  - Database Techniques
  - Object-based Hints
  - Aggregate Awareness

Performance

- Index Awareness
  - Which is faster?



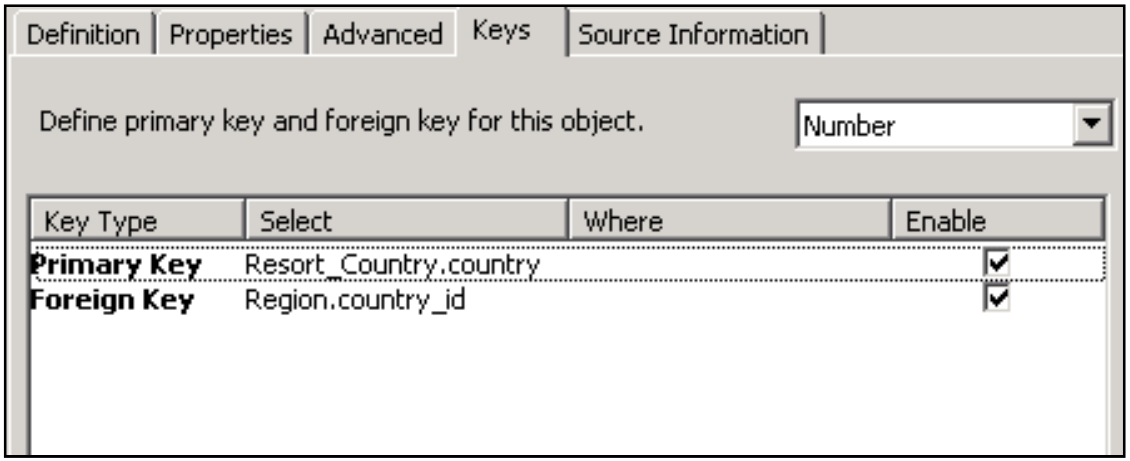
Customer.city_id = City.city_id
and
City.city in ('Dallas', 'Chicago')



Customer.city_id in (11, 15)

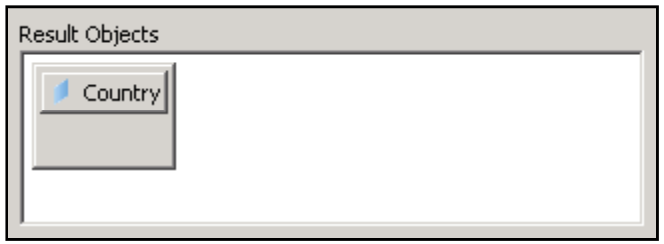
Performance

- Index Awareness, cont'd
  - The universe can substitute IDs for descriptions on the fly
    - Eliminates a join AND uses the foreign key index
  - Primary and foreign keys must be programmed
    - Must be done for every object to be made "index aware"



Performance

- Index Awareness, cont'd
  - Downside:
    - Uses the object's List of Values query for this purpose
    - Not a recommended technique for slowly changing dimensions



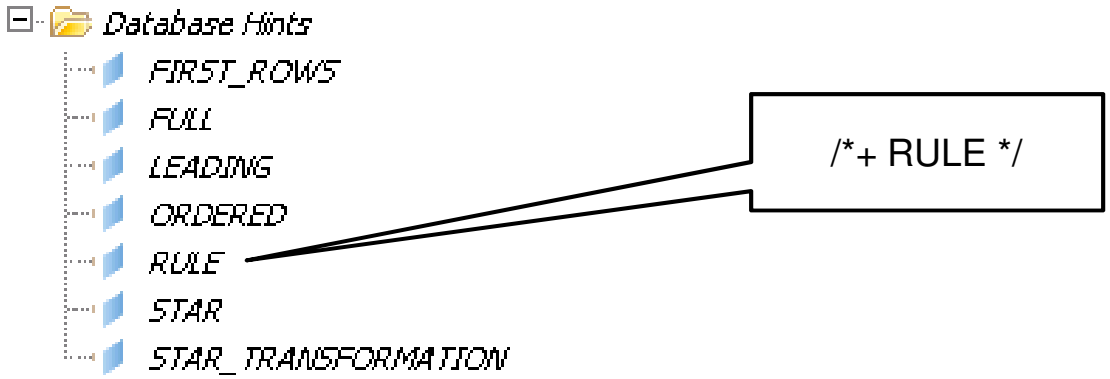
```
SELECT DISTINCT
  Resort_Country.country,
  Resort_Country.country_id
FROM
  Country  Resort_Country
```

Performance

- Database Techniques
  - Reduce the number of joins where possible
  - Identify performance potholes in your universe structure
    - May be a particular table or view
  - Work with your DBA to optimize data retrieval
    - Refresh statistics on a regular basis
    - Add indexes based on DB optimizer strategy (EXPLAIN PLAN)
    - Replace views with materialized views if possible

Performance

- Object-based Hints
  - NOT meant for ad-hoc universes in general
    - Objects *could* be hidden from public view
  - Applicable for databases that use hints (Oracle)
  - Objects are created that introduce the database hint
  - Must be the first object added to a query



Performance

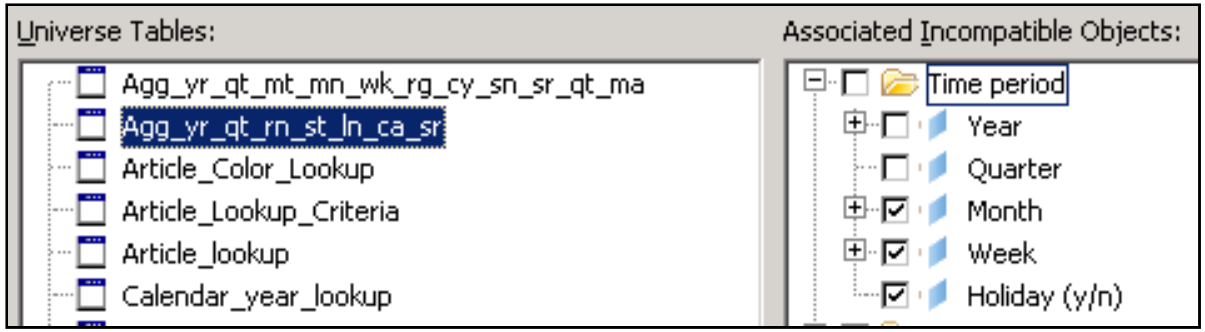
- Aggregate Awareness
  - The only technique where a single object reacts to other objects within the same query
  - Used to select the fastest / optimal table to retrieve the data from
  - Originally meant for measures
    - Can be used to consolidate dimensions as well
  - Steps involved in using Aggregate Awareness:
    - Define the AggregateAware object
    - Define classes/objects incompatible with that object

Performance

- Aggregate Awareness, cont'd
  - Steps involved in using Aggregate Awareness:
    1. Define the AggregateAware object, fastest first

```
@Aggregate_Aware(
      sum(Agg_yr_qt_mt_mn_wk_rg_cy_sn_sr_qt_ma.Sales_revenue),
      sum(Agg_yr_qt_rn_st_ln_ca_sr.Sales_revenue),
      sum(Shop_facts.Amount_sold))
```

    2. Define incompatibilities



Performance

- Aggregate Awareness, cont'd
  - Incompatibility is determined by the grain of the table

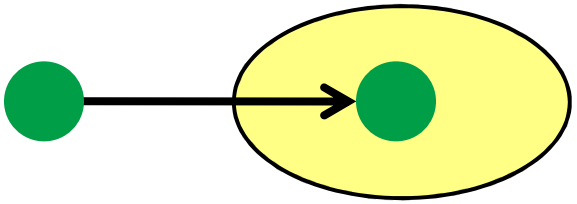
Class	Object	Incompatible ?
Time Period	Year	
	Quarter	
	Month	x
	Week	x
	Holiday (y/n)	x
Store	State	
	City	x
	Store Name	x
…		

Agenda

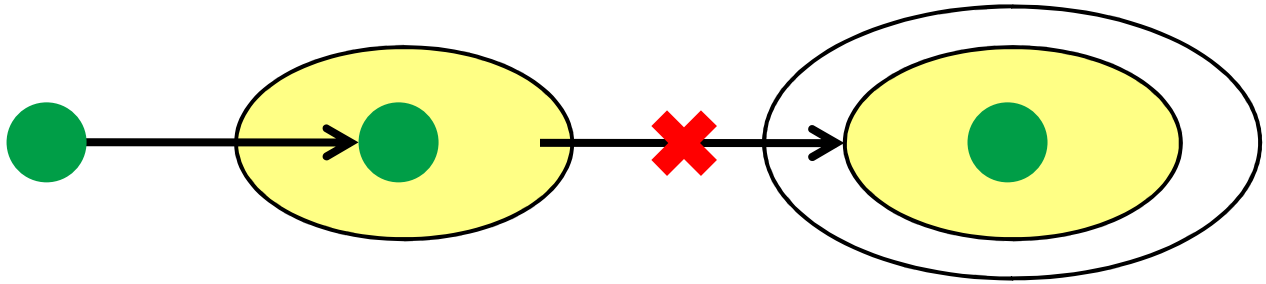
- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Linking

- Don’t re-invent the wheel!
  - Existing universes can be reused to form a new universe



  - Only one level of linking is allowed



Linking

- What you inherit
  - All existing tables and joins
  - All classes and objects
- What you don't
  - Contexts
  - Aggregate incompatibilities
  - Customized List of Values
    - This is changed slightly since XIR2
    - If both source and target universes are in the same universe folder …
    - … those lists can be reused

Agenda

- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Security

- XIR2 introduced the idea of restriction sets
  - Specific security rules that override default settings
  - Replaced universe override feature in v5/6 Supervisor
  - Unlike Supervisor, these sets can be named and reused
  - Rules applied when the queries are run
- Use these sets to:
  - Change the database connection by user
  - Alter the amount of time queries run for a group
  - Apply row and column-level security
- Restriction sets can interface with existing security rules

Agenda

- Introduction
- Ground Rules
- Classes and Objects
- Joins
- Hierarchies
- Parameters
- Performance
- Linking
- Security
- Conclusion

Conclusion

- Building good universes is not a trivial process
- These best practices will guide you in that effort

Questions?

- Alan Mayer

  214-295-6250

  alan.mayer@solidgrounded.com

SolidGround Technologies

SESSION CODE: 806

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: 806