Building universes by the book
Standardizing your semantic development

Alan Mayer
Solid Ground Technologies

© Copyright 2014
Wellesley Information Services, Inc.
All rights reserved.

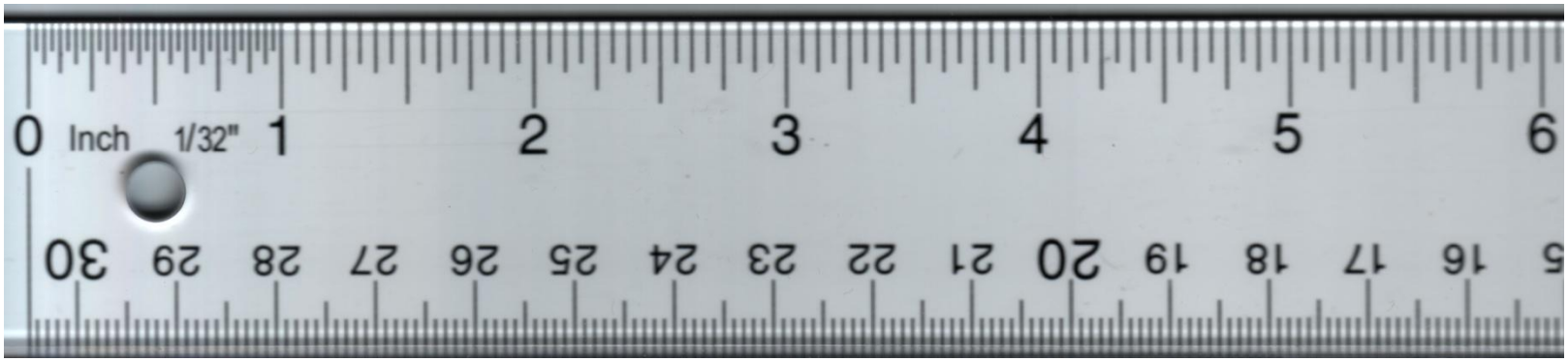
In This Session

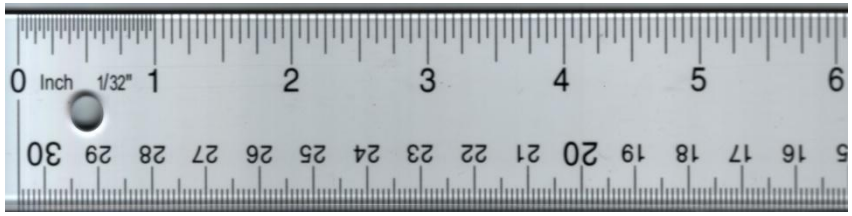
- Identify the areas best suited for universe semantic layers
- Understand the benefits of standards-driven design
- Learn best practices for designing universes
- Apply these guidelines to any current BusinessObjects environment (XI 3.1, BI 4.x)
- Realize the impact these design decisions have on your enterprise BI environment

What We'll Cover

- Introduction
- Focus
- General Best Practices
- Best Practices for Relational Databases
- Best Practices for HANA Models
- Wrap-up

Imagine a World with No Standards …





The Benefits of Standards-Driven Development

- Accuracy
- Functionality
- Maintainability
- Adoption
- Performance
- Reusability

What We'll Cover

- Introduction
- Focus
- General Best Practices
- Best Practices for Relational Databases
- Best Practices for HANA Models
- Wrap Up

Data Sources of Interest

- **Relational Databases**
  - Original target of BusinessObjects universes
  - All techniques and best practices apply
- **HANA-based Systems**
  - Appears like a relational target from a semantic perspective

Other Data Sources Not Covered

- Multi-Dimensional / OLAP Data Sources
  - Universes can be built against multi-dimensional cubes
  - May not be the best way of interacting with this data
  - Other tools can read OLAP data without a universe
    - Voyager (XI 3.1)
    - Analysis for OLAP (BI 4.x)

- SAP BW / BEx Queries
  - Universes can be built against these sources
  - Not the only interface available
    - Example: BICS for BEx Queries
  - Really merits a presentation of its own

BusinessObjects Versions

- Practices discussed will cover all current versions
  - BusinessObjects XI 3.1
    - Designer
  - BI 4.x
    - Universe Design Tool
    - Information Design Tool (IDT)
- Customers with earlier versions can still benefit
  - BusinessObjects XI R2
  - BusinessObjects 5.x / 6.x

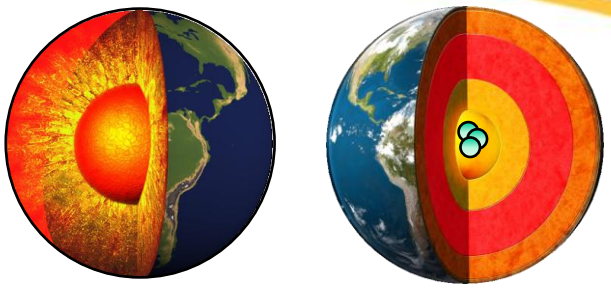
What We Won't Focus On

- Not a step by step tutorial on creating a universe from scratch
- Assumes beginner-level familiarity with the tools

What We'll Cover

- Introduction
- Focus
- General Best Practices
- Best Practices for Relational Databases
- Best Practices for HANA Models
- Wrap Up

UNV vs. UNX?



- XI 3.1 only allows .UNV universes
- BI 4.x allows either .UNV or .UNX
- Our advice:
  - Keep imported universes from XI 3.1 as .UNV
  - Create .UNX universes for:
    - Universes that require special .UNX features (federation)
    - New universe development
- Why not convert all universes to .UNX?
  - The technology has taken some time to mature
    - BI 4.0 advocates lived with semantic issues for 2 years
    - Data providers in every document must be changed

How Many Objects?

- Traditionally 700 - 800 objects
- This advice varies widely
  - OEM universes have many more
  - Those for end users may have less
- Larger universes take more Java runtime memory
  - This assumes you are using Java-based tools
    - Webi Rich Client
    - BI 4.1 Rich Internet Application (RIA)
- Objects inversely proportional to number of universes
  - Many universes allows fewer objects per universe
- Aim for the simplest universe possible

Single or Multiple Universes?

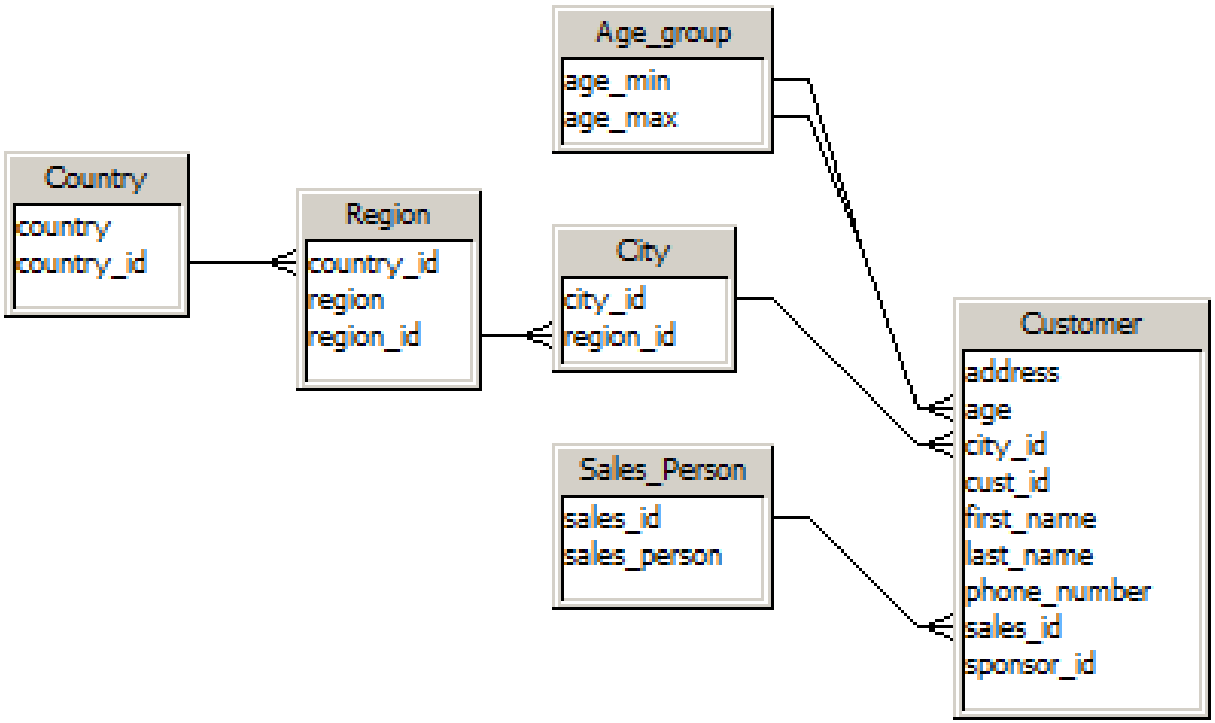
- Not an easy question
- Depends on how users interact with the universe
  - Are some business areas independent of others?
    - Single universe per area
  - If not, are there common terms used across areas?
    - Customer number, product number, …
    - Allows data from multiple universes to combine correctly in a report
  - How will users want to combine data?
    - Combinations (UNION / INTERSECT / MINUS) require the same universe
    - Subqueries require the same universe

Decision Point

What We'll Cover

- Introduction
- Focus
- General Best Practices
- **Best Practices for Relational Databases**
- Best Practices for HANA Models
- Wrap Up

Relational Database Overview



Views and tables appear as tables through Universe Designer, IDT

Building Blocks

Relational Universe Best Practices

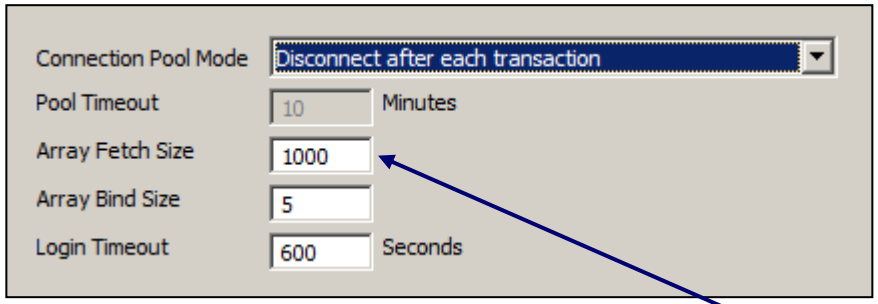
- **Guidelines will be given for:**
  - **Parameters**
  - **Classes**
  - **Objects**
  - **Joins**
  - **Hierarchies / Navigation Paths**
  - **Performance Techniques**

Universe Parameters

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

Database Connections

- Disconnecting after each transaction is safest
- Increase Array fetch size to accelerate data retrieval

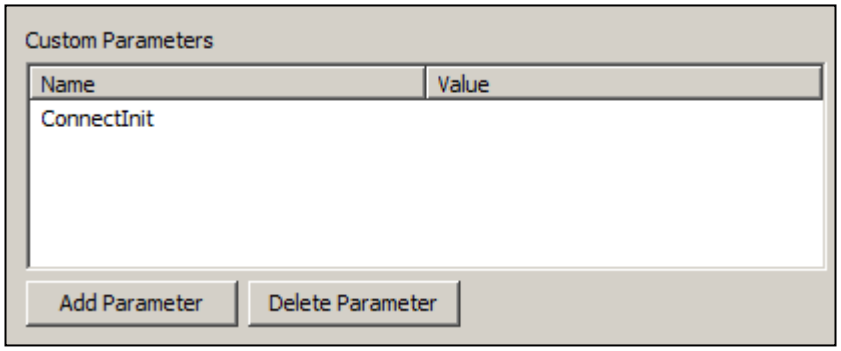


Tip

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

Custom Database Parameters

- Custom parameters can be selectively added
  - Highly dependent on database
- Hints can be added for certain databases (Oracle)
  - Especially desirable for data marts
  - Custom parameter = Hint
  - Value = `/*+ STAR_TRANSFORMATION */`



Query Banding

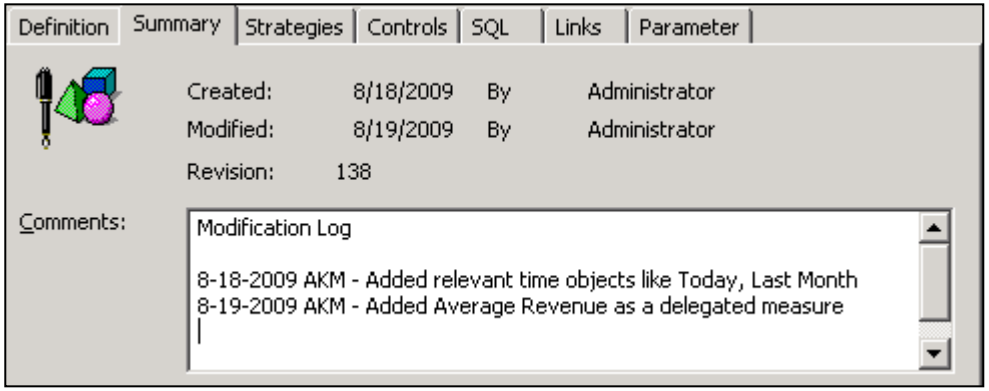
- The ConnectInit, Begin_SQL, and END_SQL custom parameter can be used to tag SQL statements
  - Allows Designers to send commands to the database after opening a connection
  - Identify SQL statements by document, query, universe, …
  - Teradata Example:

```
SET QUERY_BAND =
'DocName='@variable('DOCNAME')';
DPName='@variable('DPNAME')';
UserName='@variable('BOUSER')';
UnvName='@variable('UNVNAME')'; ' FOR SESSION;
```

Highly database dependant!

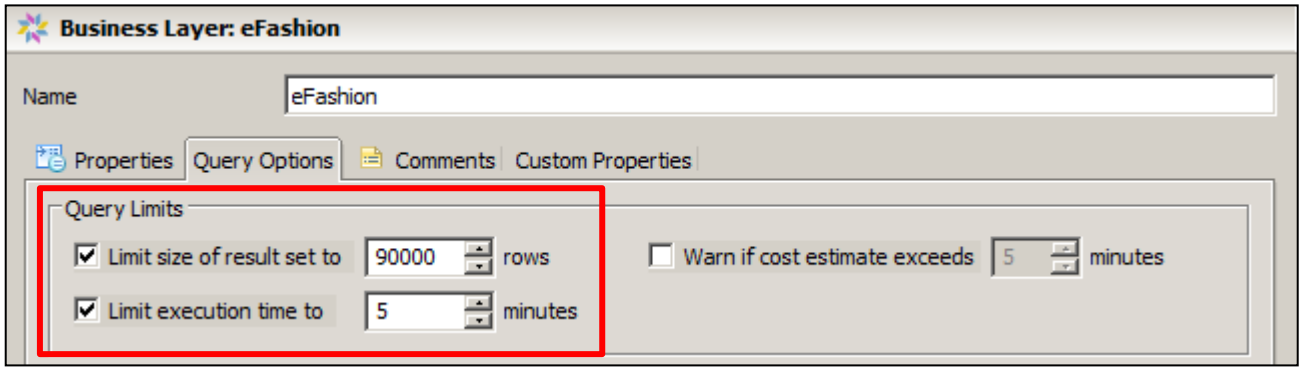
Summary Information

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



Query Limits

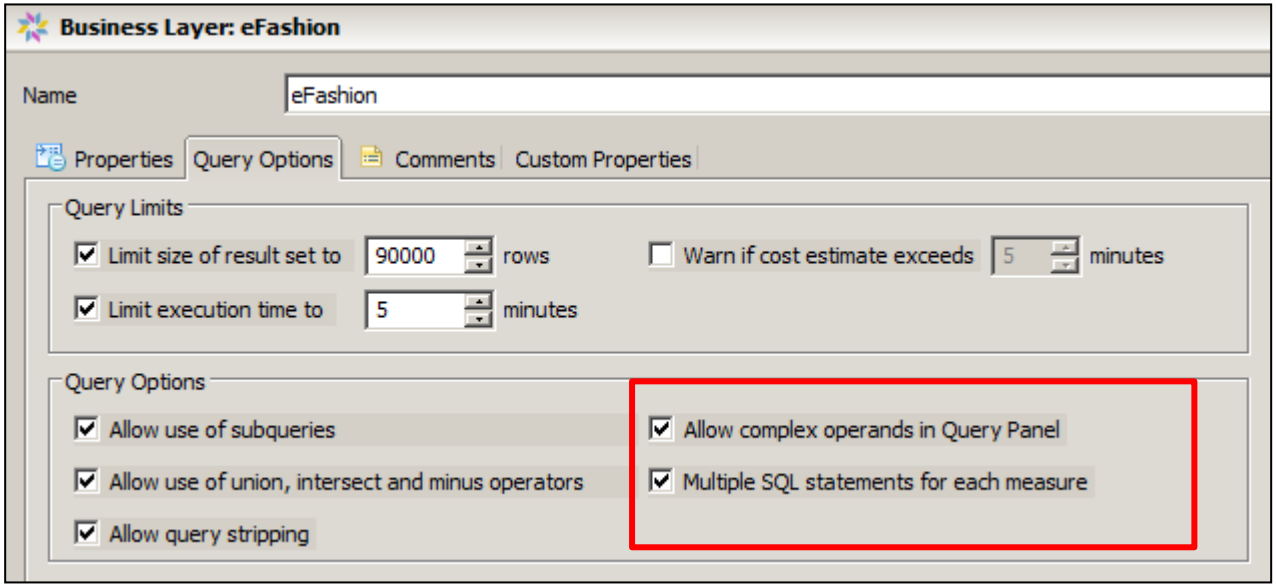
- These limits become default values for your universe
- The first two (rows, time) are the most important
- They prevent runaway queries by establishing maximum caps



Look in **Universe Parameters** > **Controls** for legacy .UNV tools

SQL Parameters

- **Multiple Path options are the most important**
  - **They control the creation of multiple SELECT statements**
  - **This will help with incorrect aggregation issues**

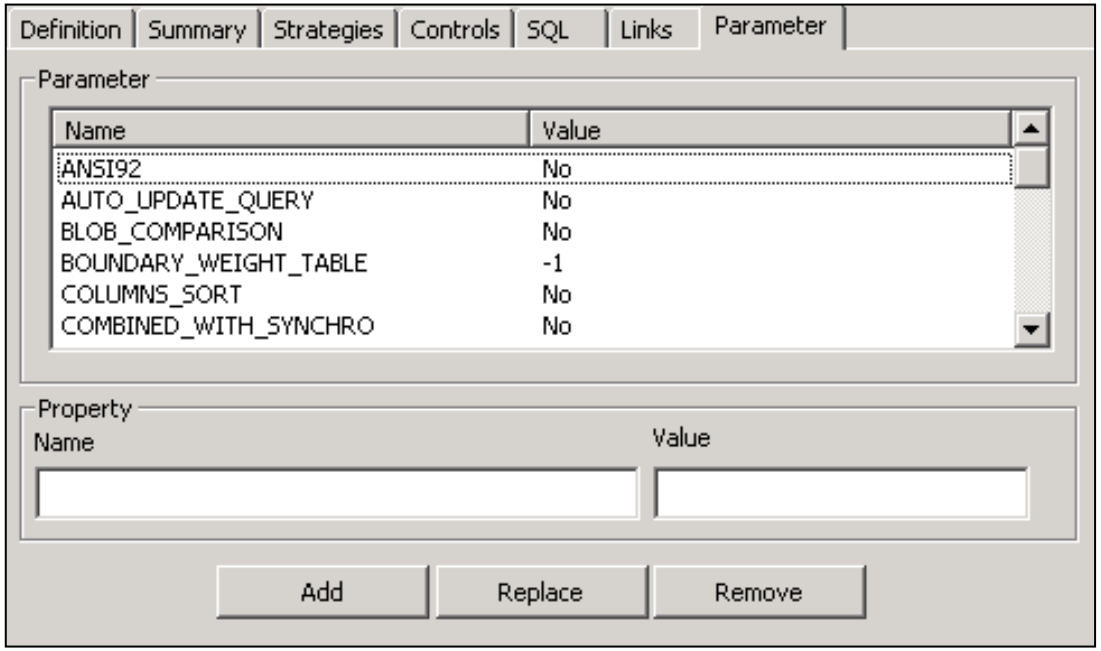


Note

Look in Universe Parameters > Controls for legacy .UNV tools

Dynamic Parameters

- These parameters can expand or limit a universe's functionality
  - IDT: Universe Properties > Parameters
  - Legacy (.UNV): Universe Parameters > Parameter



Universe Parameter Examples

- 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
    - Formats multi-pass SQL as a single statement
    - UNIONS the multiple SELECTS

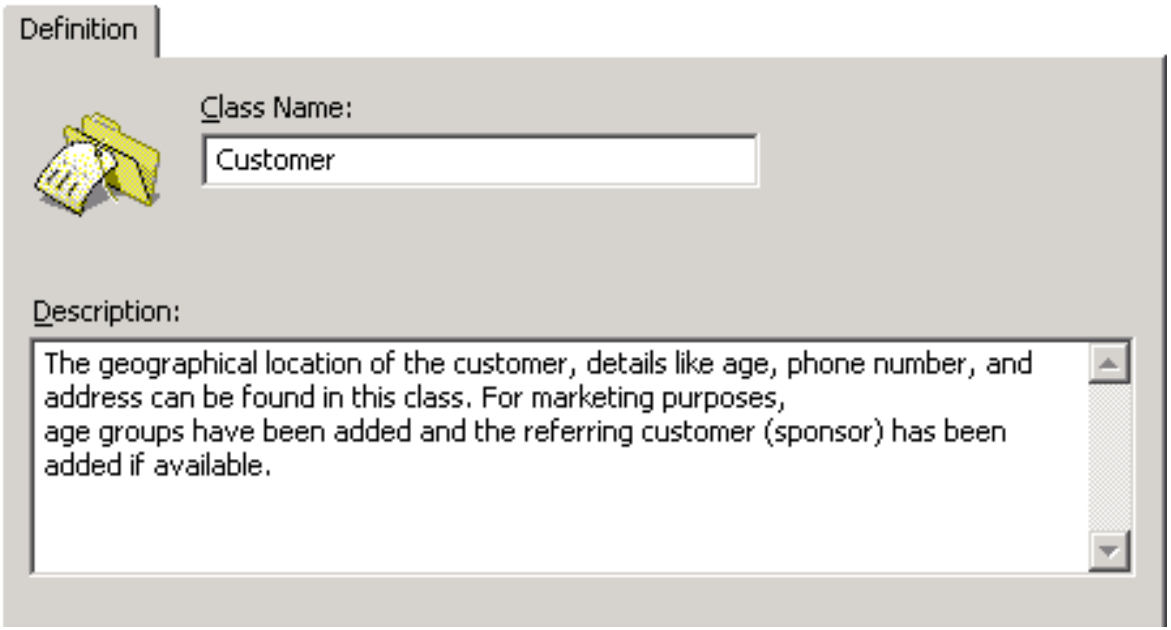
JOIN_BY_SQL will help with datasources and tools that have trouble with multiple SELECTs – Crystal Reports, HANA

Classes

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

Class Naming Conventions

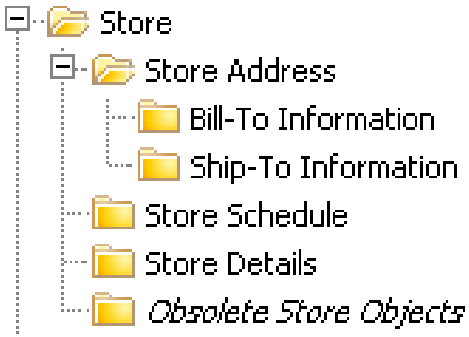
- Stick to a reasonable limit for the name (60 chars)
- Descriptions can be long – be as descriptive as possible
  - How objects can be used
  - Any special filters on this particular class



Best Practice

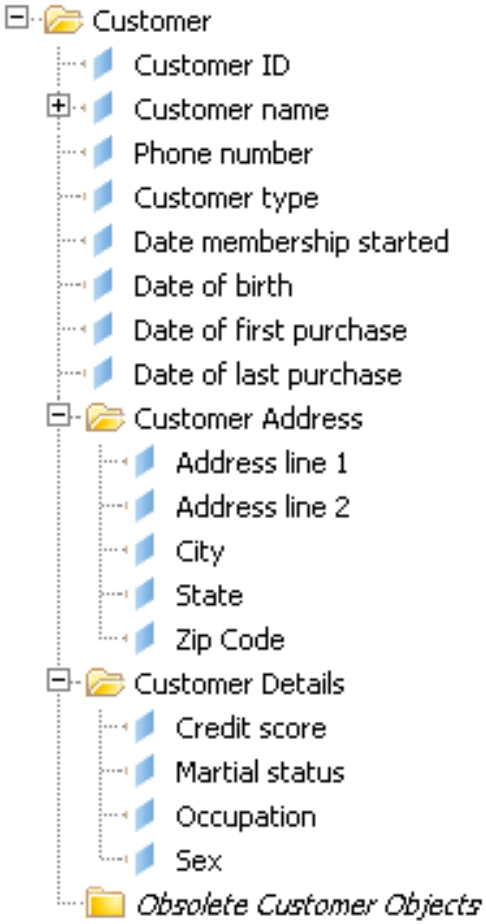
Class Layout

- 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
  - 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



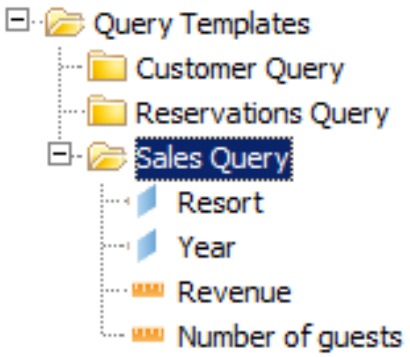
Class Contents

- 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
  - Alternatives:
    - Alphanumeric
    - Order by type (dates, calculations, …)
    - Hierarchically (general to specific)
    - Fastest to execute when placed in conditions



Query Templates

- **Offer common queries as classes**
  - Helps novice users start a query
  - Easy to use – simply drag the class
  - Especially handy for universes with large number of objects



Tip

This is a great way of accelerating the adoption of your universes. Users can create a valid query instantly!

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

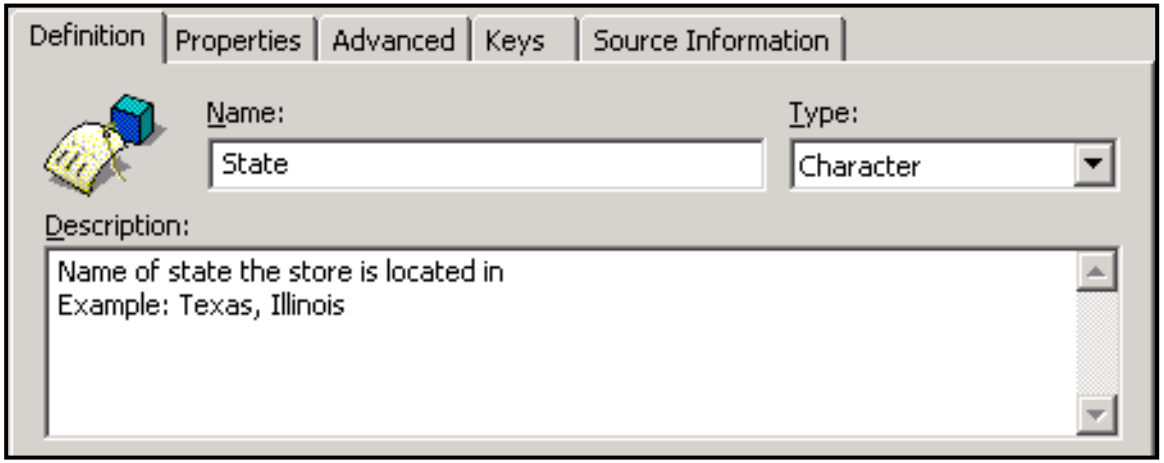
Object Naming Conventions

- 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

Object Descriptions

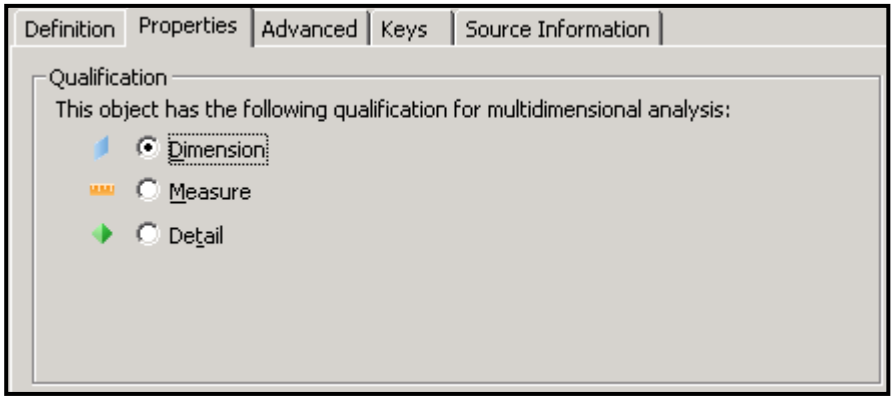
- **Add help text for EVERY object**
  - **Add a description then several examples**
  - **Add format masks (MM/DD/YY) on the first line**



Best Practice

Object Type

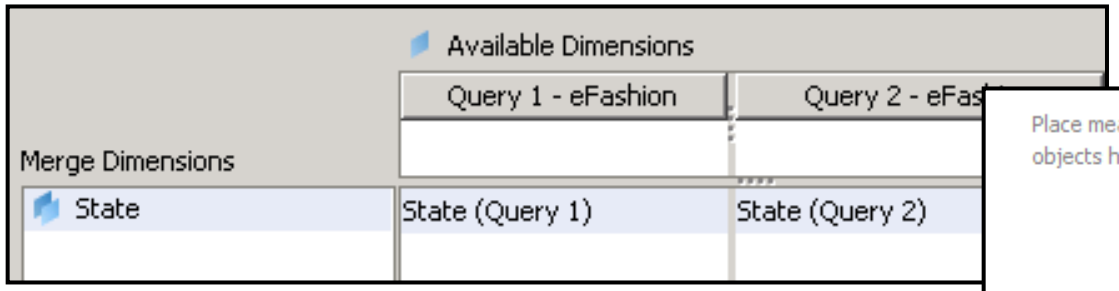
- Should the object be a dimension, detail, or measure?
  - Dimension: Key fact that drives the remainder of the query
  - Detail: Additional information that depends on existing dimension
  - Measure: Calculation
- Biggest point of confusion: Dimension or detail?
  - More on this in a moment …



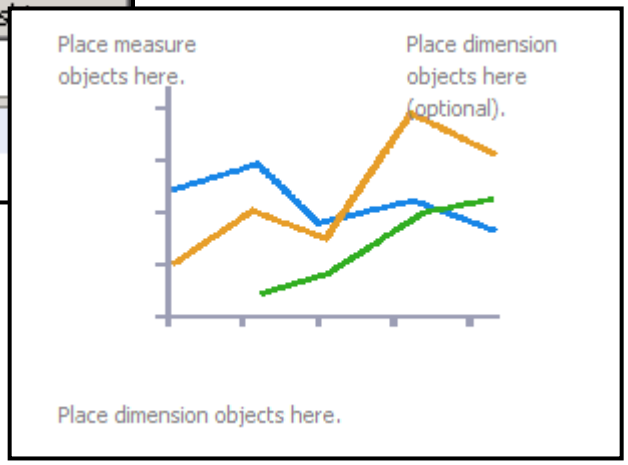
Object Type by Function

- **Report functionality depends on object type**
  - **Hierarchies consist of dimension objects only**
  - **Query linking (merged dimensions) depend on linked dimensions**
  - **Report writers like Web Intelligence require measures**

Merged Dimensions

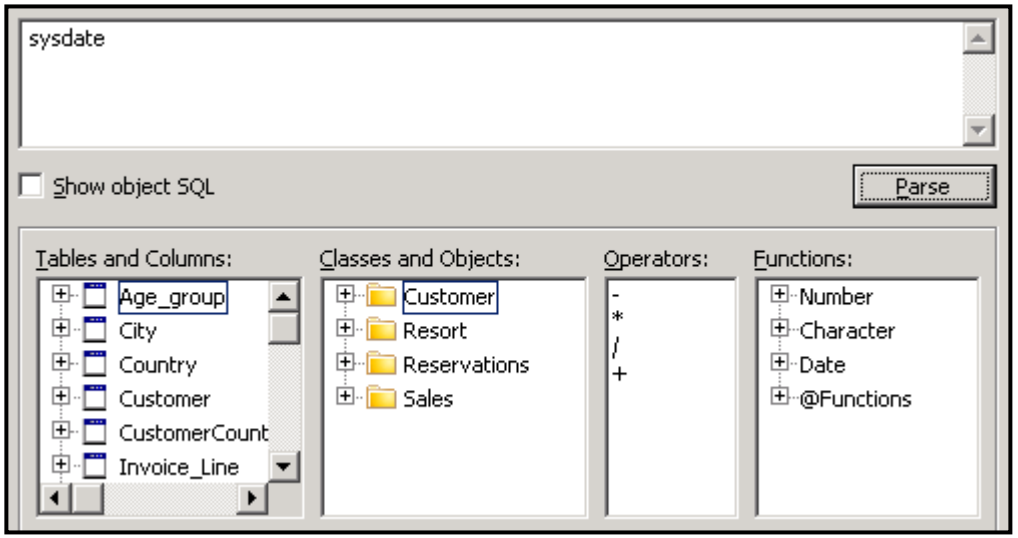


Graphs



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.
    - Example: any object not based on a table ('Today')



Objects based on pseudo-columns or system functions will not parse

Objects with 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 )
```

Object 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'

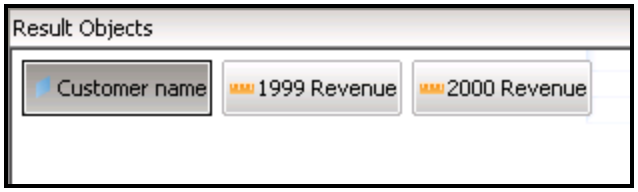
1999 Revenue

```
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'
```

WHERE Clause Alternatives

- 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 )
```

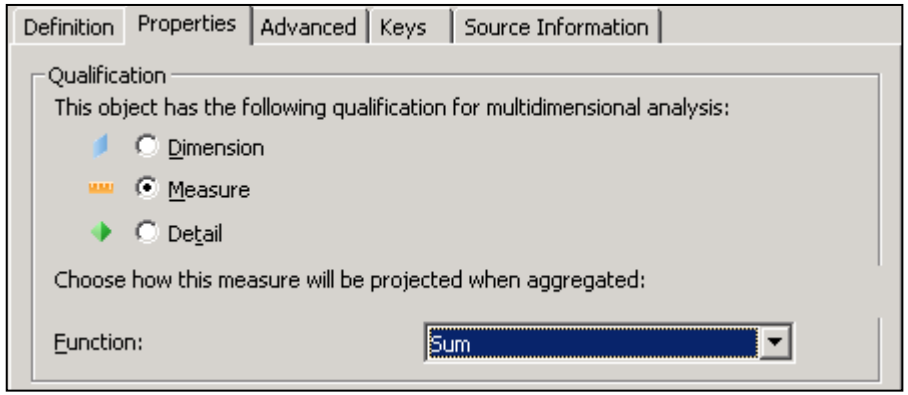
- Condition objects could also be used
  - Users can change AND to OR in the query panel

Object Calculations

- **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>)

Calculation Projections

- **Projections control how Webi works with measures**
  - **Specifies how measures will be aggregated AFTER data is returned**
- **The projection for COUNT is usually SUM**

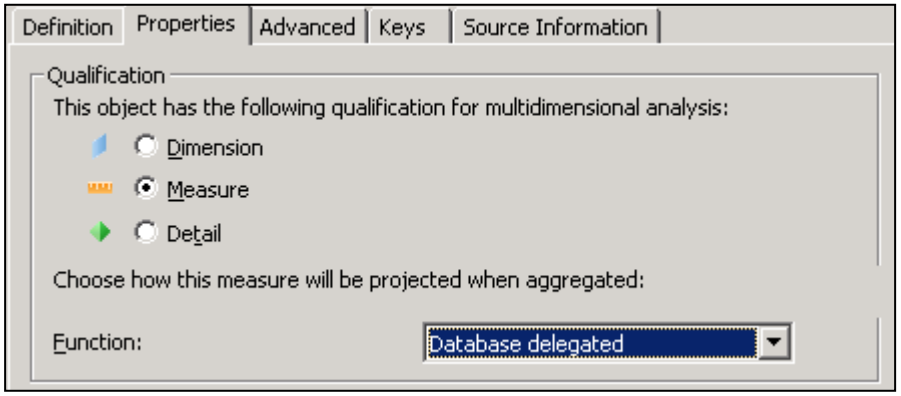


Country	Resort	Revenue
France	French Riviera	835,420
US	Bahamas Beach	971,444
US	Hawaiian Club	1,479,660

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

Delegated Projections

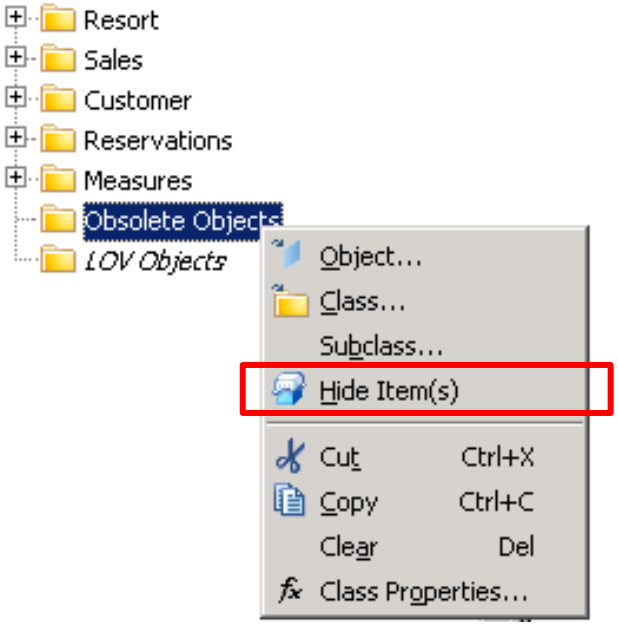
- **Use the Delegated Measure feature for AVG, %**
  - **This forces the report writer to re-run SQL every time dimensions or details within the block change**
  - **This prevents incorrect calculations**
  - **Can't automatically calculate the average of an average**



Best Practice

Hidden Objects and Classes

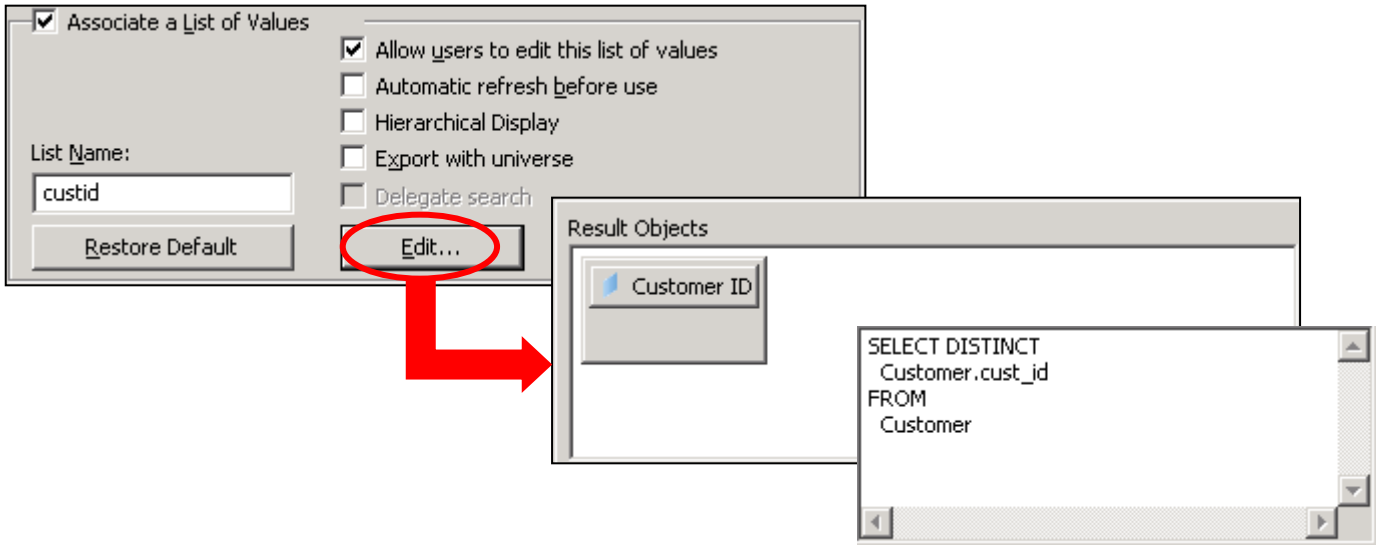
- Hide objects / classes that are obsolete
- Extremely useful technique for creating more complicated objects
  - Can also be used to accelerate List of Value queries



Tip

List of Values (LOV)

- 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



Extended List of Values

- **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**

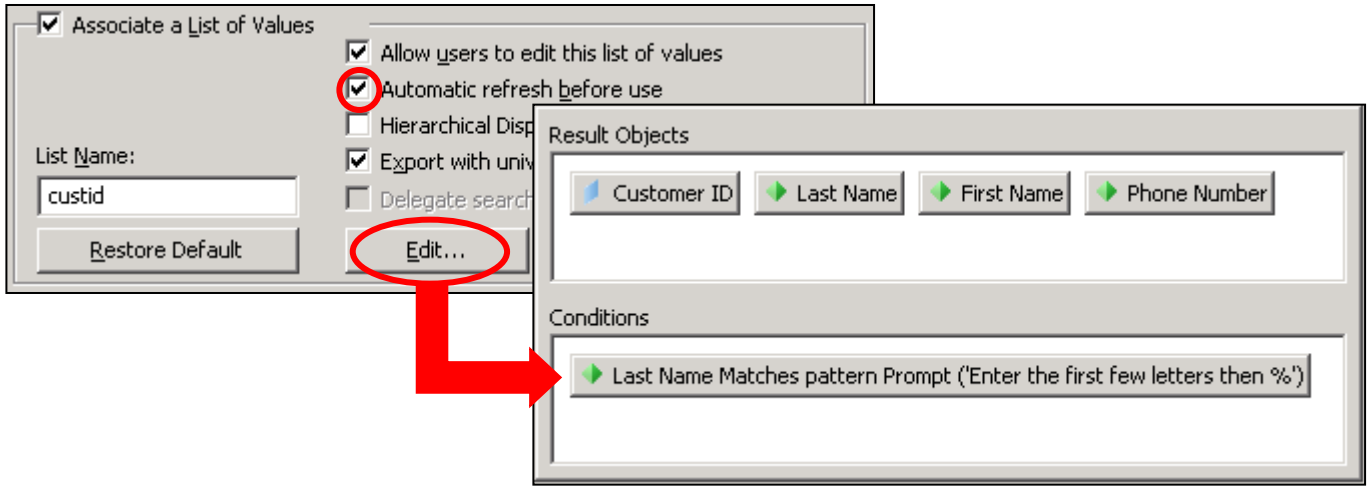
Customer ID	Last Name	First Name	Phone Number
101	Brendt	Paul	(212) 555 2146
102	McCarthy	Robin	(214) 555 3075
103	Travis	Peter	(510) 555 4448
104	Larson	Joe	(213) 555 5095
105	Goldschmidt	Tony	(619) 555 6529

Tip

Additional objects can be any type (dimensions, details, …)

List of Values Conditions

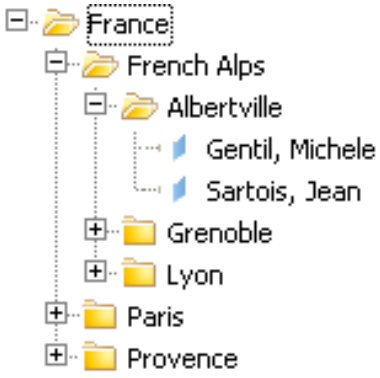
- **Conditions can also be added to further refine possible values**
  - **Embedded prompts can reduce long lists (1000 or more)**
  - **Pattern matching can be used to reduce the list further**
  - **Make sure to automatically refresh LOV queries with prompts**



Tip

Hierarchical List of Values

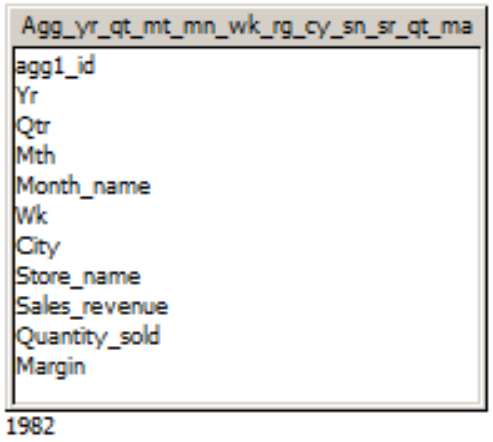
- LOV results can be displayed in list or hierarchical format
- If the latter is desired, arrange LOV objects in drilled order
  - Left-most object is returned as final value
  - Next object would represent the top of the hierarchy
  - Third object would server as the second hierarchical level
  - Second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

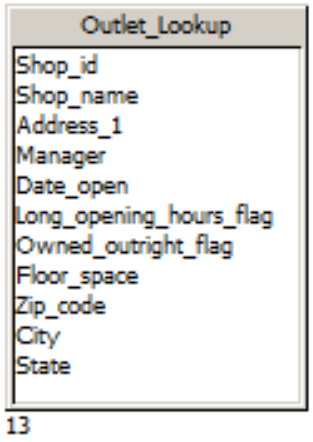


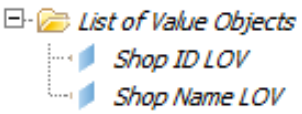
Tip

List of Values on Lookup Tables

- Create special LOV objects from small lookup tables
- Hide these objects
  - Only use them in a LOV query
- Performance gains can be tremendous!





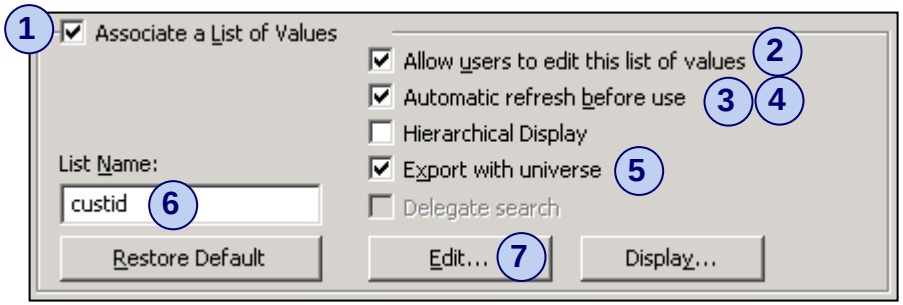


Tip

This technique will come in handy for HANA-based universes

Other LOV Best Practices

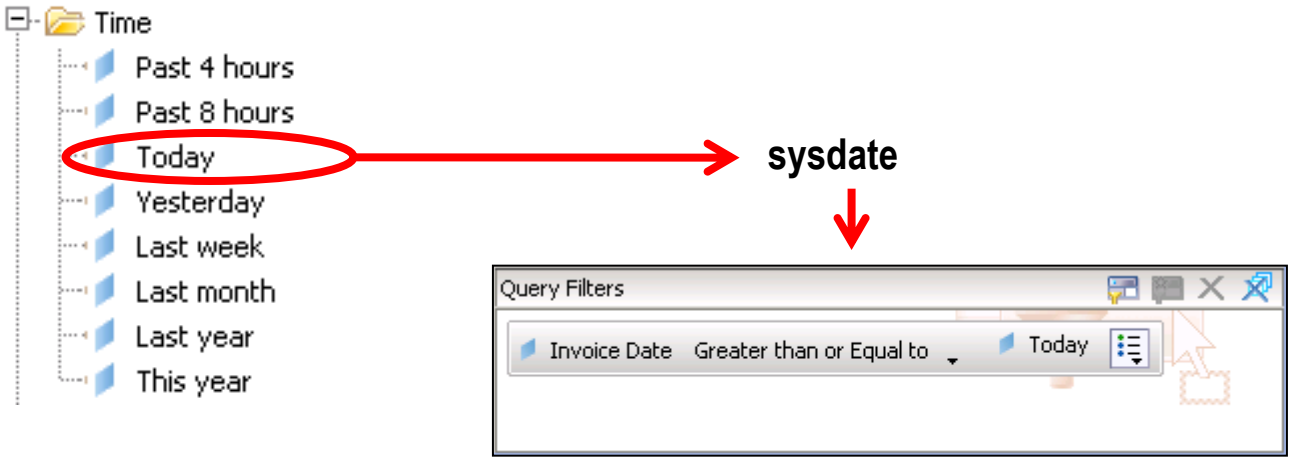
1. Don't maintain list of values for dates, calculations
2. Most users are not allowed to edit their List of Values
3. Always refresh a list that includes a prompted condition
4. Don't refresh a list that is relatively static
5. Always export customized list of values
6. Name a customized LOV query (other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



Best Practice

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



Tip

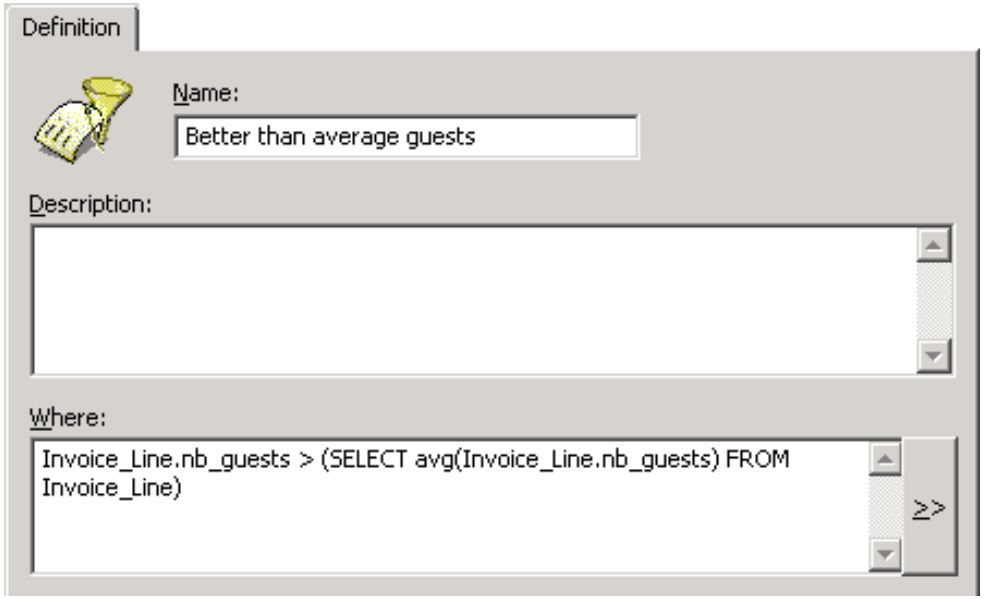
Object Formatting

- **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

Condition Objects

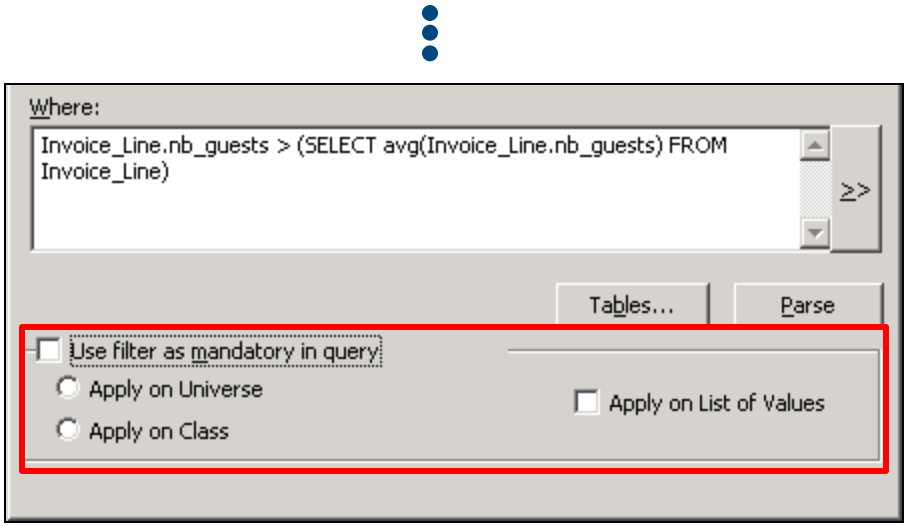
- Condition objects act as pre-programmed query filters
- Great for frequently used and difficult conditions
  - Subqueries, correlated subqueries
- Once created, users can combine in a query using AND, OR



Knowing what conditions are used most often comes in handy …

Conditions on Classes

- 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



Tip

Query Linking

- Queries can be combined in 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

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

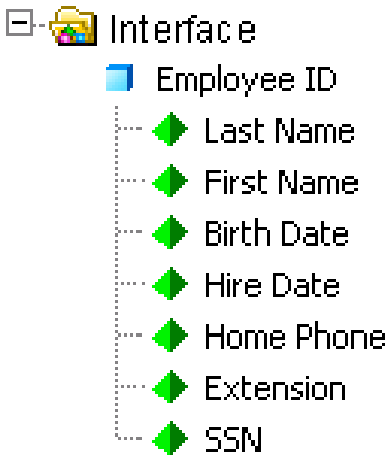
Country	Number of guests	Future guests
France	446	46
US	1,105	56

Incorrect

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

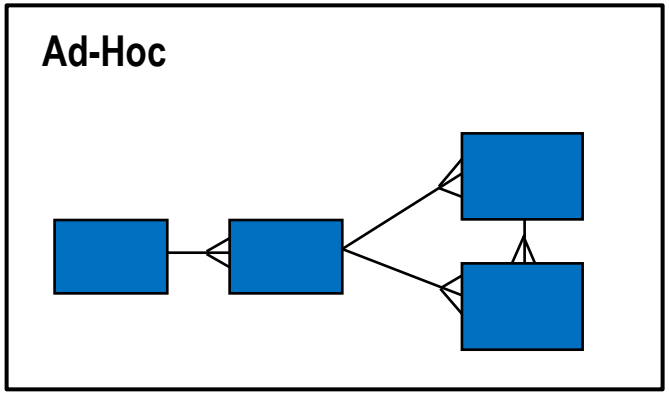
Interface Classes and Objects

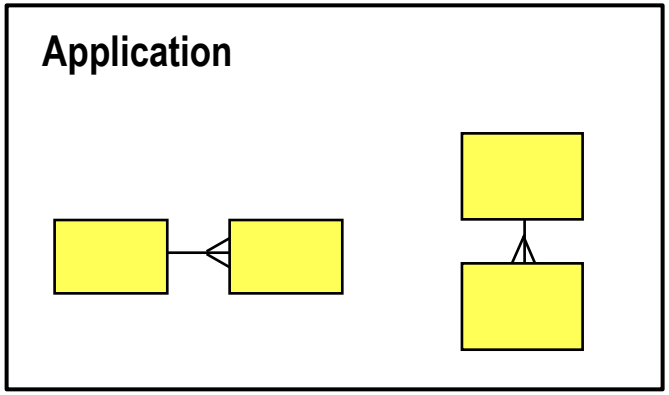
- Add interface classes to your universe to simplify linking
  - Users quickly adapt to looking for these classes
  - Results are accurate and reliable
- This will also drive your object type decisions
  - Dimension vs. detail becomes much clearer



Join Strategy

- 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

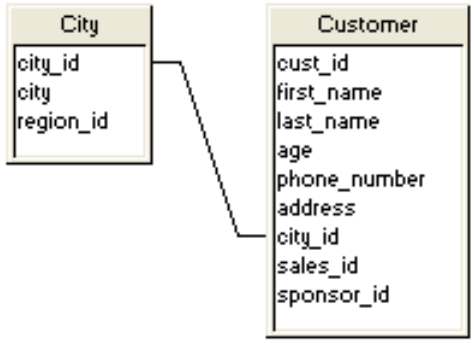




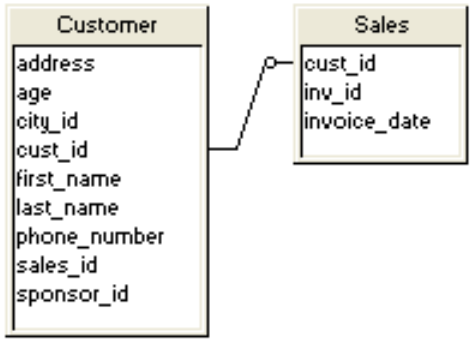
Join Types

- **Many different types of joins are available**

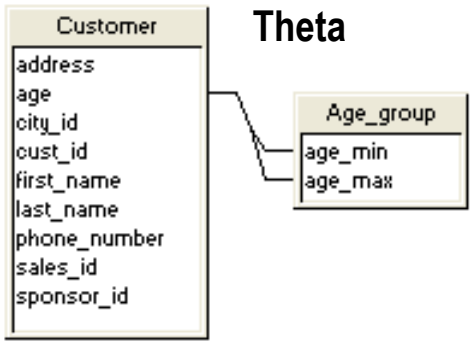
Inner



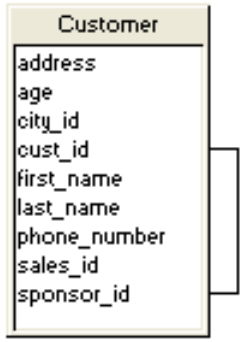
Outer



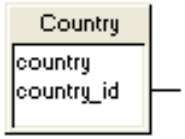
Theta



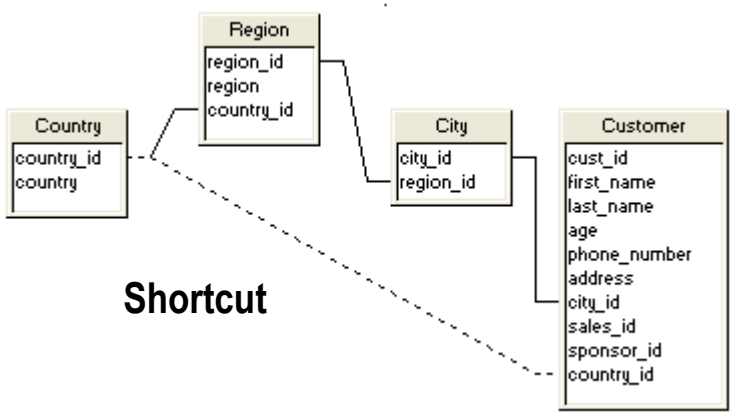
Recursive



Self

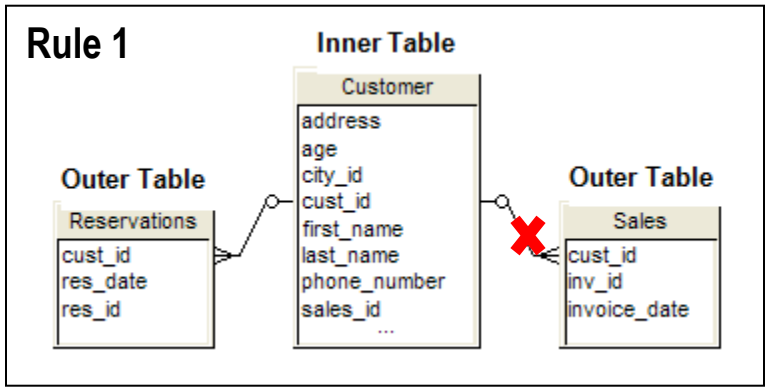


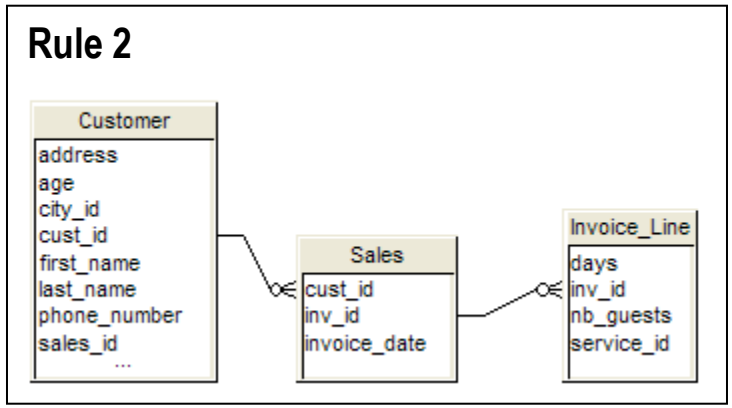
Shortcut



Outer Joins

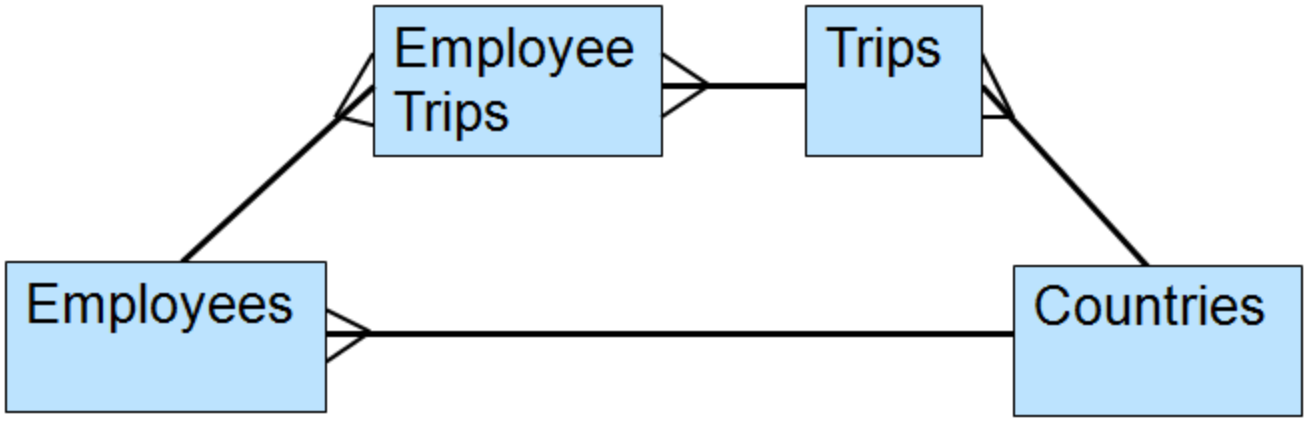
- **Outer joins have special considerations**
  - Not the best performing join
  - Two rules that are forced by SQL:
    1. Inner table of an outer join cannot be used as the inner table of another outer join
    2. Outer joins must cascade





Loops

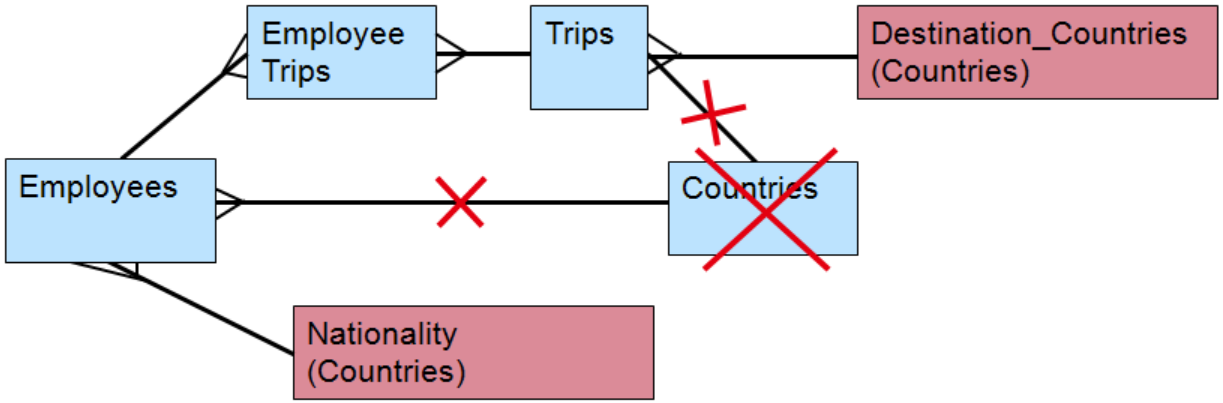
- Two or more paths between tables
- Developers MUST resolve loops to allow users full query access



Unresolved loops will prevent users from creating queries!

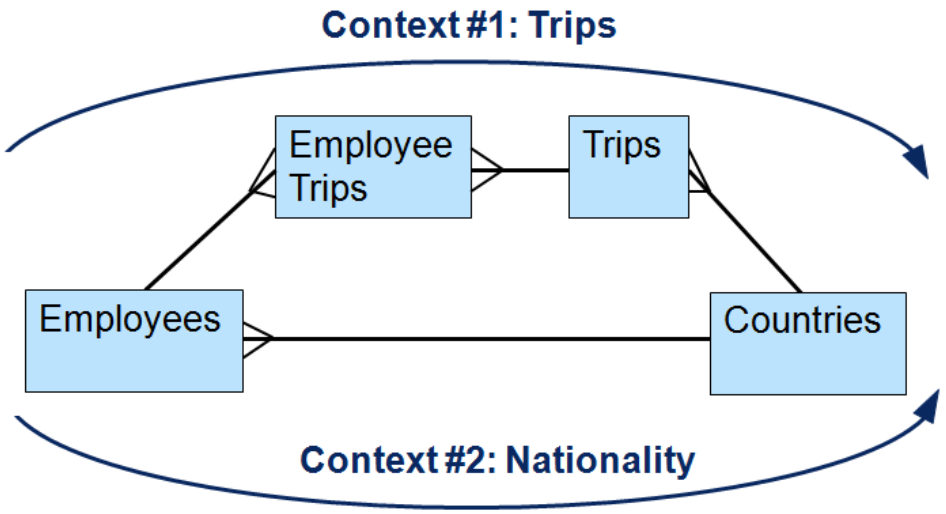
Joins

- **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



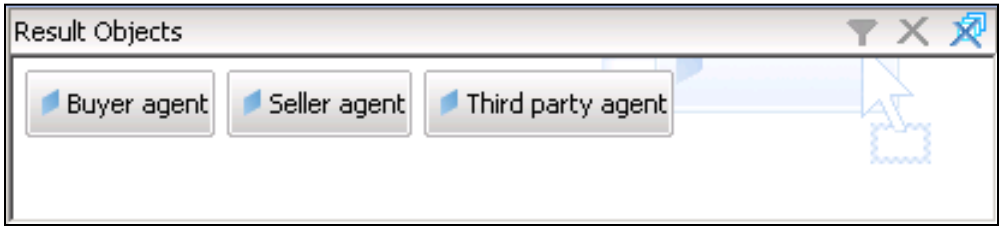
Comparing Aliases to Contexts

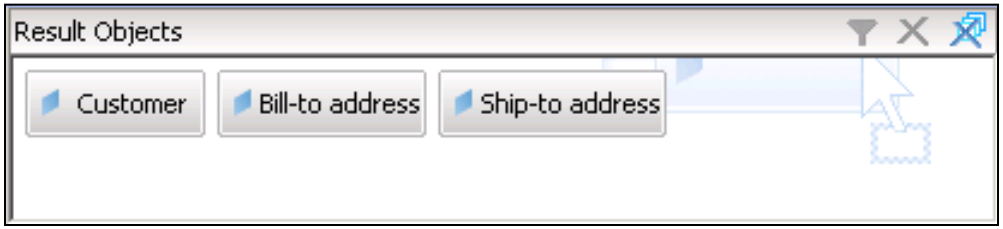
Aliases	Contexts
Resolves loop at design time	Resolves loop when query is run
Creates more objects	No additional objects added
Aliases cascade	Context selection may be forced on user
	Every join must be part of the context (XI 3.1 and previous versions)

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

Choosing Aliases

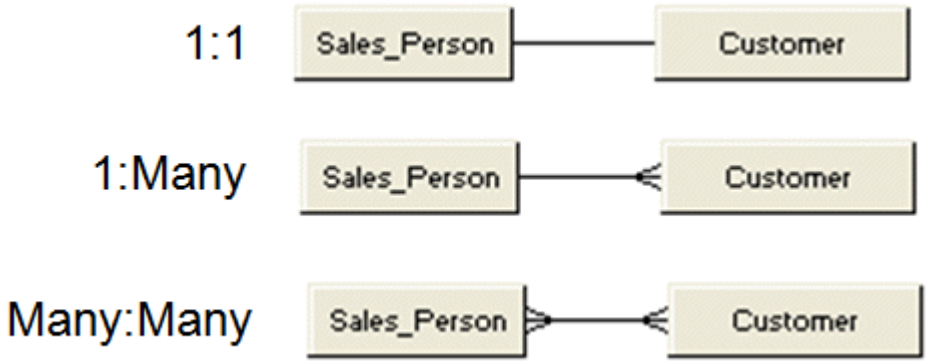
- **ACID Test for Aliases**
  - Place all objects created from aliases in a query
  - Would this make sense to a user?
    - If so, aliases must be used to simultaneously represent values
  - Aliases used to resolve chasm traps, lookup tables





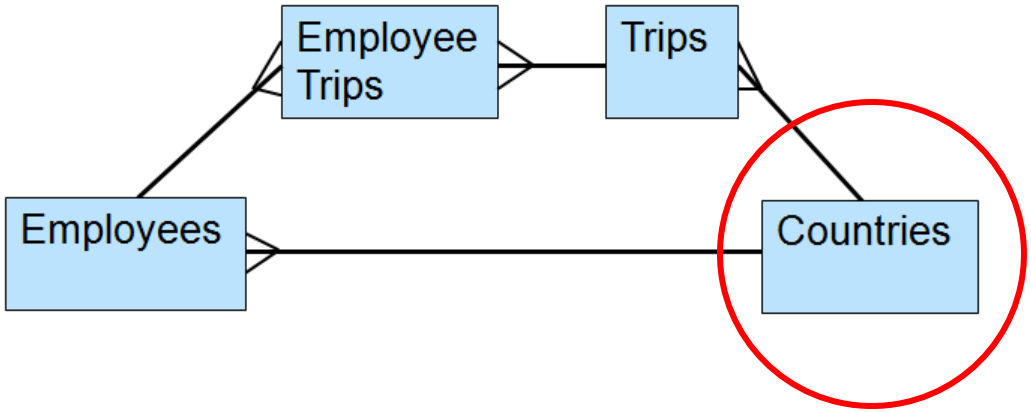
Join Cardinality

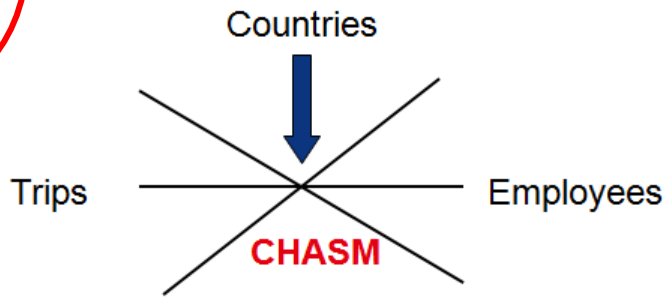
- Determines the number of values joined between tables
  - One to one
  - One to many
  - Many to many
- ALWAYS set the cardinalities for every join
- NEVER depend on automatic cardinality detection
  - The algorithm used is not 100% accurate



Chasm Traps

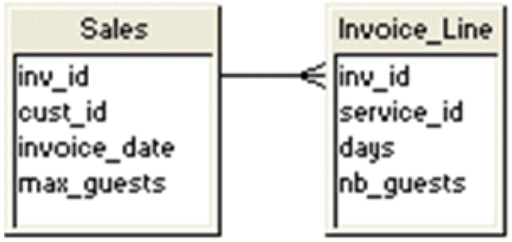
- Many to one to many relationship
- No relationship from left to right
- Usually resolved with aliases





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



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

Hierarchies / Navigation Paths

- **Navigation Paths allow Webi users to drill**
  - Consist entirely of dimensions
  - Can reflect natural hierarchies
    - Time (Year > Quarter > Month > Week)
    - Organizational (Corporate > Region > Division > …)
- **Two best practices**
  - 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

Hierarchies (XI 3.1 and earlier) are known as Navigation Paths in 4.x

Relational Performance Techniques

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

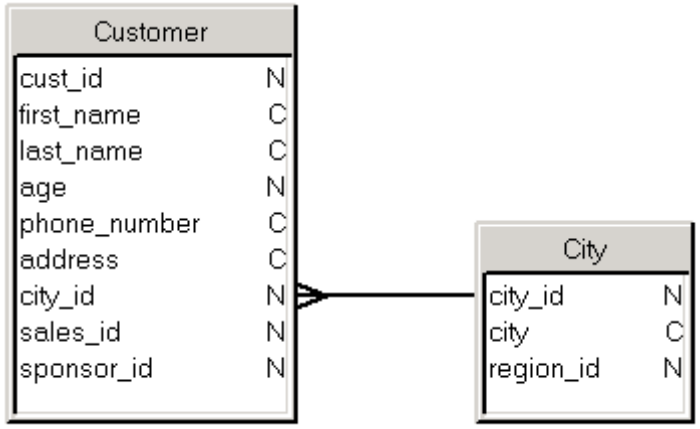
Shortcut Joins

- Eliminate additional joins in the query if not needed
  - The dashed line below represents a shortcut join
    - If the query contains objects from Country and Customer only, the shortcut will be taken

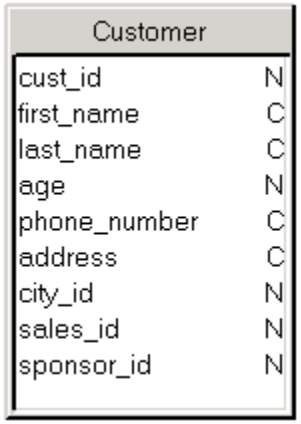
Be careful! Only represent true shortcuts this way. Never use to intentionally break a loop.

Index Awareness

- **Which is faster?**



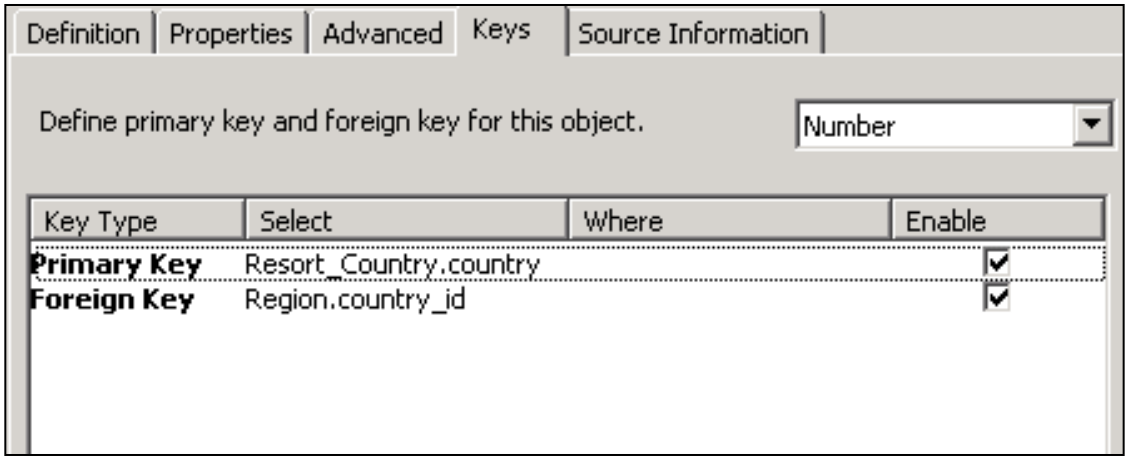
Customer.city_id = City.city_id
and
City.city in ('Dallas', 'Chicago')



Customer.city_id in (11, 15)

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"

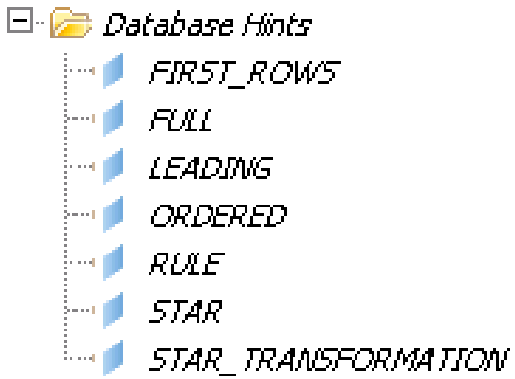


Performance Database Techniques

- 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
- Only create joins on indexed columns
  - Primary and foreign keys are usually indexed

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



Aggregate Awareness

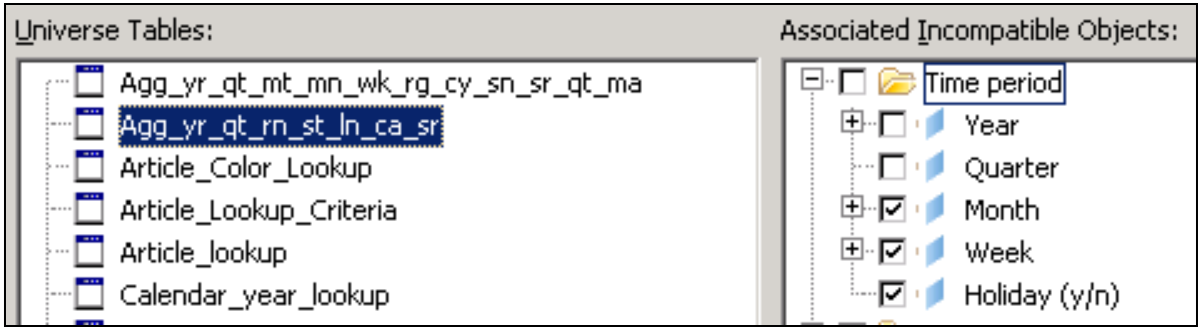
- The only technique where a single object reacts to other objects within the same query
- Used to select the fastest / optimal table to retrieve the data from
- Originally meant for measures
  - Can be used to consolidate dimensions as well
- Steps involved in using Aggregate Awareness:
  - Define the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

Making a Universe Aggregate Aware

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



Recognizing Incompatibilities

- **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
…		

What We'll Cover

- Introduction
- Focus
- General Best Practices
- Best Practices for Relational Databases
- Best Practices for HANA Models
- Wrap Up

HANA Best Practices

- **Semantic guidelines will be given for:**
  - **HANA tables**
  - **HANA models**

HANA Database Overview

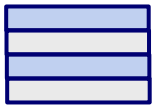
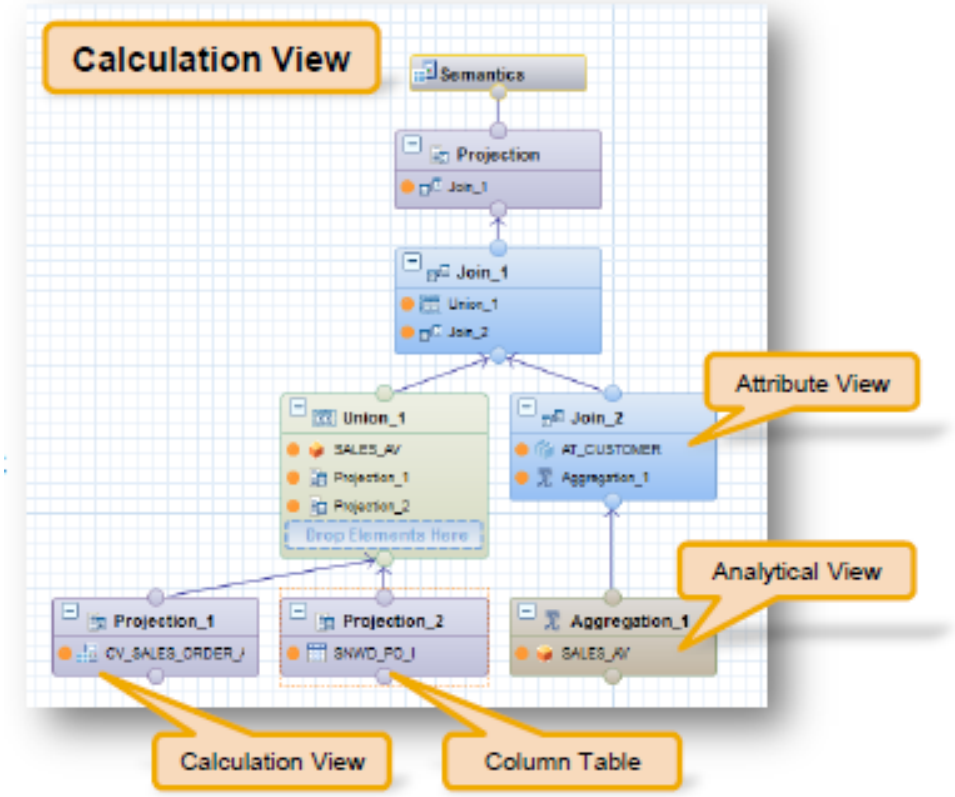


Table (Row-based)

Table (Column-based)

Views / Models

- **Calculation View** (Calculation Flow)
- **Analytical View** (Aggregations)
- **Attribute View** (Master Data)



Building Blocks

HANA Universe Building Blocks

- Universes can be built from HANA tables or models
- Tables
  - Many of the previous relational best practices apply
  - Provides for a very flexible ad-hoc environment
  - Can't take advantage of complexity or speed that models offer
- Models
  - Universes become quite simple
    - One model per universe
  - In general, models should not be joined to each other
  - A universe can mix models with tables for certain purposes
    - List of values
    - Aggregate navigation

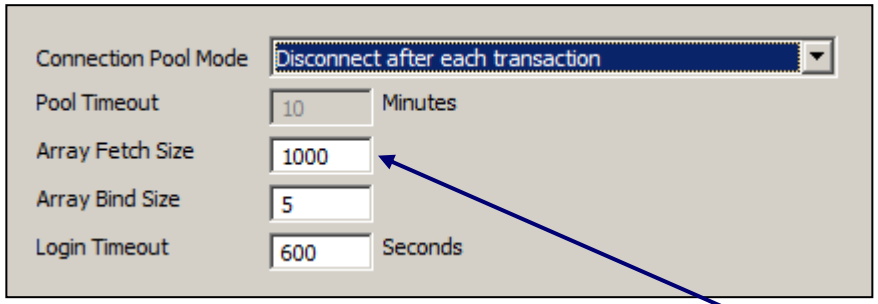
HANA Universes Against Models

- Creating universe queries against a model executes its flow
  - Intense processing could be started per query
  - "Boil the ocean"
  - Need to minimize the number of times this occurs
- Also need to consider where to add semantic logic
  - Two places: Universe and HANA Model
  - Pushing logical calculations to the model may be more efficient
    - Leaving calculations in the universe may take MORE resources

Best Practice

HANA Database Connections

- Same advice as relational universes
- Increase Array fetch size to accelerate data retrieval
  - More rows per fetch means fewer HANA requests



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

HANA Dynamic Parameters

- Set the JOIN_BY_SQL parameter to **Yes**

```
SELECT
  NVL( F__1.Axis__1,F__2.Axis__1  ),
  F__1.M__20,
  F__2.M__37
FROM
      (
      SELECT
        Resort_Country.country AS Axis__1,
        sum(INVOICE_LINE.nb_guests) AS M__20
      FROM
      …
      F__1
       FULL OUTER JOIN
      (
      SELECT
        Resort_Country.country AS Axis__1,
        sum(RESERVATION_LINE.future_guests) AS M__37
      FROM
      …
       F__2
      ON ( F__1.Axis__1=F__2.Axis__1 )
```

Best Practice

Converts queries with multiple SELECTS to one statement

Universes against HANA Models

- Do not join models to anything else
- Stick with one model per universe
  - Possible exception for aggregate awareness
- Use straight dimensions and measures from the model
  - Do not add additional functions or logic
  - Push that added complexity to the HANA model
- Same advice for joins and condition objects
  - No added complexity
- Avoid multi-source universes (federation)
- Avoid Index Awareness
  - Most universes will be based on one model anyway

Universes against HANA Tables

- Use HANA columnar tables for best performance
- Avoid added complexity in the WHERE or GROUP BY clauses
  - Short version: Avoid additional complexity
  - WHERE clause
    - Condition objects or WHERE clauses of other objects
  - GROUP BY clause
    - Created by BusinessObjects automatically
    - Influenced by object SELECT clauses
- Always change data types manually in conditions
  - Don't rely on HANA to automatically change data types

Best Practice

What We'll Cover

- Introduction
- Focus
- General Best Practices
- Best Practices for Relational Databases
- Best Practices for HANA Models
- Wrap Up

7 Key Points to Take Home

- Create your own semantic standards from the topics presented
- Understand the benefits of standards and the risks in veering from them
- Focus on those areas that yielded the greatest reward
- Know which semantic standards apply to HANA-based projects
- Confidently apply these techniques over a wide variety of BusinessObjects versions
- Plan to retrofit those universes that aren't standards-driven
- Enjoy the peace of mind that comes with well-designed semantic solutions!

Where to Find More Information

- Didier Mazoue, “Creating Relational Universes: Best Practices”, http://scn.sap.com/docs/DOC-23256
- Alan Mayer, “Better Universes by Design”, ASUG SAP BusinessObjects User Conference 2010
- Verossi, Bihan, Mazoue, “Creating a universe on SAP HANA: Best Practices”, http://scn.sap.com/docs/DOC-20569
- Alan Mayer, “Preparing for Life on Planet UNX”, ASUG SAP BusinessObjects User Conference 2012

Your Turn!

Questions?

Alan Mayer
214-295-6250 (office)
214-755-5771 (mobile)
alan.mayer@solidgrounded.com
Twitter: @solidgrounded

Please remember to complete your session evaluation

Disclaimer

SAP, R/3, mySAP, mySAP.com, SAP NetWeaver®, Duet™®, PartnerEdge, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Wellesley Information Services is neither owned nor controlled by SAP.