Hyperion Querying (b)
Advanced Queries
What are the Advanced Query Topics?

- **Appended Queries**: Queries that merge or stack SELECT statements to produce combined results.
- **SubQueries**: Embedding SELECT statements into the WHERE clause.
- **Derivable Queries**: Embedding SELECT statements into the FROM clause.
- **Local Queries**: Using local data sections (results, tables, etc.) as topics on a datamodel.
- **Additional Features**: Meta Topics, Master Datamodels, Sync With Database, etc.
What is an Appended Query?

Appended queries are queries that use a special operator between two SQL select statements to combine the results:

- **UNION ALL** – Returns all rows from both select statements (stacks)
- **UNION** – Returns all rows from both select statements that are unique/distinct over the compounded results (stacks then parses the duplicates)
- **INTERSECT*** – Returns only rows that would be retrieved identically by both select statements (stacks then returns only the duplications)
- **MINUS* (or EXCEPT**)** – Returns rows found in the results of the first select statement that are not also found in the second (returns only rows from the first select statement then removes any rows that match a row from the second statement)

* - Our versions of DB2 and MSAccess do not offer INTERSECT or MINUS/EXCEPT
** - Use MINUS or EXCEPT depending upon your database type
### What does an Appended Query Do?

#### Raw Data

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaheim</td>
<td>1</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2</td>
</tr>
<tr>
<td>Cologne</td>
<td>3</td>
</tr>
<tr>
<td>Dublin</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cologne</td>
<td>3</td>
</tr>
<tr>
<td>Dublin</td>
<td>4</td>
</tr>
<tr>
<td>London</td>
<td>5</td>
</tr>
<tr>
<td>Amsterdam</td>
<td>6</td>
</tr>
</tbody>
</table>

#### TABLE-A

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaheim</td>
<td>1</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2</td>
</tr>
<tr>
<td>Cologne</td>
<td>3</td>
</tr>
<tr>
<td>Dublin</td>
<td>4</td>
</tr>
</tbody>
</table>

#### TABLE-B

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cologne</td>
<td>3</td>
</tr>
<tr>
<td>Dublin</td>
<td>4</td>
</tr>
<tr>
<td>London</td>
<td>5</td>
</tr>
<tr>
<td>Amsterdam</td>
<td>6</td>
</tr>
</tbody>
</table>

#### WITH UNION ALL

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaheim</td>
<td>1</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2</td>
</tr>
<tr>
<td>Cologne</td>
<td>3</td>
</tr>
<tr>
<td>Dublin</td>
<td>4</td>
</tr>
</tbody>
</table>

#### WITH UNION

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaheim</td>
<td>1</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2</td>
</tr>
<tr>
<td>Cologne</td>
<td>3</td>
</tr>
<tr>
<td>Dublin</td>
<td>4</td>
</tr>
<tr>
<td>London</td>
<td>5</td>
</tr>
<tr>
<td>Amsterdam</td>
<td>6</td>
</tr>
</tbody>
</table>

#### WITH INTERSECT

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cologne</td>
<td>3</td>
</tr>
<tr>
<td>Dublin</td>
<td>4</td>
</tr>
</tbody>
</table>

#### WITH MINUS

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaheim</td>
<td>1</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2</td>
</tr>
</tbody>
</table>

All rows which are present in both TABLE-A and TABLE-B are highlighted in yellow.
How can We Understand Appendations?

VENN LEGEND
1. Green = Query-A w/o Rows Duplicated in Query-B
2. Red = Query-A Rows Duplicated in Query-B
3. Yellow = Query-B Rows Duplicated in Query-A
4. Blue = Query-B w/o Rows Duplicated in Query-B

SELECT * FROM Query-A
UNION
SELECT * FROM Query-B

Returns 1+2+3+4

SELECT * FROM Query-A
INTERSECT
SELECT * FROM Query-B

Returns 2

SELECT * FROM Query-A
UNION ALL
SELECT * FROM Query-B

Returns 1+2+4

SELECT * FROM Query-A
MINUS
SELECT * FROM Query-B

Returns 1
How do We Create Appendations?

From the menu bar use “Query > Append Query”:

Tabs are then shown for each statement in the appendation along with an appendation control for choosing from the available operators:

**NOTE:** Click the tabs to navigate between select statements or right-click to remove or rename
How do We Define Appendations?

- Click the appendation operator to change between UNION, INTERSECT, MINUS, etc.
- Create a query in each tab to represent a part of the appendation (first tab is primary for EXCEPT/MINUS) by using the existing datamodel to add topic items to the request and filter lines*
- Because data will be “stacked” there must be an equal amount of request items in each tab and their datatypes must be consistent
- Remove and rename tabs by using the right-click menu

* - Must use a single datamodel for all tabs (or call reporting services)
What is a SubQuery?

- SubQueries are used to dynamically define filter values based upon a separate yet related query.
- In SQL a SubQuery is represented by a full SELECT statement being embedded within parenthesis and used as the value for a filter in the WHERE clause.
- SubQueries can only use one item in their SELECT clause and consequently only one request item in Hyperion.
- SubQueries are added in Hyperion via the advanced options in the filter dialog box.
- Once added a subquery is displayed within the Hyperion section catalog as a separate sub-section underneath its parent query section.
- SubQueries can be either simple or correlated.
How Can We Understand SubQueries?

Introduce SubQuery:

```
...WHERE Sale_Date = (SELECT MAX(Sale_Date) FROM MyTable)
```

Against This Raw Data:

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Sale Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaheim</td>
<td>2/3</td>
<td>12,258.66</td>
</tr>
<tr>
<td>Anaheim</td>
<td>2/5</td>
<td>14,805</td>
</tr>
<tr>
<td>Anaheim</td>
<td>2/6</td>
<td>94,136.10</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2/1</td>
<td>182,107.50</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2/2</td>
<td>65,393.46</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2/6</td>
<td>54,729.30</td>
</tr>
<tr>
<td>Cologne</td>
<td>2/2</td>
<td>10,408.50</td>
</tr>
<tr>
<td>Cologne</td>
<td>2/4</td>
<td>15,973.32</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/1</td>
<td>7,222.10</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/2</td>
<td>66,880.58</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/3</td>
<td>13,312.30</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/4</td>
<td>28,259.42</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/5</td>
<td>38,811.26</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/6</td>
<td>19,305.47</td>
</tr>
</tbody>
</table>

To See Only These Rows:

...because the result of the subquery is 2/6

**NOTE:** No rows are displayed for “Cologne” because it has no data that passes the filter criteria
**What is a Correlated SubQuery?**

Correlated subqueries introduce a join from the subquery to the “outer” query in order to apply logic within a grouping:

<table>
<thead>
<tr>
<th>Store Name</th>
<th>Sale Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaheim</td>
<td>2/3</td>
<td>12,258.66</td>
</tr>
<tr>
<td>Anaheim</td>
<td>2/5</td>
<td>14,805</td>
</tr>
<tr>
<td>Anaheim</td>
<td>2/6</td>
<td>94,136.10</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2/1</td>
<td>182,107.50</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2/2</td>
<td>65,393.46</td>
</tr>
<tr>
<td>Brooklyn</td>
<td>2/6</td>
<td>54,729.30</td>
</tr>
<tr>
<td>Cologne</td>
<td>2/2</td>
<td>10,408.50</td>
</tr>
<tr>
<td>Cologne</td>
<td>2/4</td>
<td>15,973.32</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/1</td>
<td>7,222.10</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/2</td>
<td>66,880.58</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/3</td>
<td>13,312.30</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/4</td>
<td>28,259.42</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/5</td>
<td>38,811.26</td>
</tr>
<tr>
<td>Dublin</td>
<td>2/6</td>
<td>19,305.47</td>
</tr>
</tbody>
</table>

**Correlate on “Store Name”:**

**To See Only These Rows:**

**NOTE:** A row is now displayed for “Cologne” because the MAX() calculation is applied within each store.
What is a Correlated SubQuery’s SQL?

```sql
SELECT
    AL1.Store_Name, AL1.Sale_Date, AL1.Amount
FROM
    Raw_Data AL1
WHERE
    AL1.Sale_Date IN
    (  
        SELECT
            MAX ( AL2.Sale_Date )
        FROM
            Raw_Data AL2
        WHERE
            AL2.Store_Name=AL1.Store_Name
    )
```
How do We Create SubQueries?

SubQueries in Hyperion are created by way of the advanced options in the filter dialog box:

...this inserts a new SubQuery Section:

NOTE: Remove subqueries by removing the associated advanced filter item in the “outer” query section.
How do We Define Our SubQuery?

- Build only one request item for feeding the value(s) to the filter in the outer-query.
- Use any topic(s) on the datamodel to build your request item and to add required filters.
How do We Correlate Our SubQuery?

Within the SubQuery:

**STEP 1**
Drag your query section onto the datamodel as a topic from the topic catalog.

**STEP 2**
Create one or more joins by dragging topic items from the datamodel topics to the new query topic.

**STEP 3**
Specify the correlated field using the prompt that is displayed and click OK.

**NOTE:**
You will virtually always want to correlate an inner-query item to the exact same item within the outer-query.
How do We Un-Correlate a SubQuery?

To un-correlate a subquery go to the subquery section and remove the associated filter item and the query topic from the datamodel:
How do We Process a SubQuery?

SubQuery sections are processed only when their “outer” query is processed and cannot be processed by themselves:
What is a Derivable Query?

- Also referred to as an “inline view”
- Represents “virtual” topics as defined by other SQL statements (other query sections in the document)
- Embeds one select statement (the derivable query) into the FROM clause of another select statement (the “outer” query)
- Used on a datamodel just like any other topic (no properties/set in the derivable query)
- Can be safely combined with other database topics (unlike local results per later slides)
- Can only be used if there are other query sections within the document that are connected to the same database
- Requires processing only the outer query itself (not the derivable queries) to return the specified results
How do We use Derivable Queries?

Use the right-click menu within the topic catalog at the lower left of a query to see the option to list derivable queries:

All query sections within the document that are actively connected to the same database as the current query (including any hidden ones) will now be available in the topic catalog:

Drag and drop any derivable queries from the catalog to the datamodel to use them as you would any other topic:
What is a Local Query?

- Also referred to as a “local results query”
- Allows the use of local results and other internal data sections of the document as topics on a query’s datamodel
- Restricts the ability to edit things like request item properties, topic properties, topic item properties, etc.
- Processes using the resources on your local machine instead of the database server (not preferable)
- Should **NEVER** be used in conjunction with database topics
- Avoid mixing local topics and database topics by retrieving all data from the database first then query the output separately as local results
How do We use Local Queries?

Use the right-click menu within the topic catalog at the lower left of a query to see the option to list local queries:

All results/data sections within the document (including any hidden ones) will now be available in the topic catalog:

Drag and drop any local results/data sections from the catalog to the datamodel to use them to build a local query:
Why use Derivable vs. Local Queries?

Derivable queries let the database do the work which will almost always be far more efficient than your local computer:
What is a Meta Topic?

A meta topic is a “virtual” topic built on a datamodel to house references to topic items from other topics:

**BUILDING META TOPICS**

- Create empty meta topics via the query menu on the menu bar at the top
- Create meta topics from existing topics via the right-click menu
- Add topics items to meta topics by dragging and dropping from existing topics
- Remove topic items or meta topics via the right-click menu
What are the Facts on Meta Topics?

- Create your own dimension, fact or other topics to make query manipulation easy for end users.
- They function in the same manner as standard database topics except are more malleable and require no joins.
- Set the “Datamodel View” option in the datamodel menu on the menu bar at the top to “Meta” in order to hide all non-meta topics.
- Add computed items to meta topics via the datamodel menu on the menu bar at the top (includes local items).
- Not available for use within local queries.
What is a Master Datamodel?

- Creates a separate datamodel section within the document that can be associated to multiple queries
- Allows changes in a single datamodel to be populated to all associated queries
- Allows newly inserted queries to be associated with the pre-built master datamodel instead of directly with the database
- Locks the datamodel and when combined with the hiding of the datamodel section it can then discourage users from editing datamodels
- Any existing datamodel can be promoted to a master datamodel by the reporting services team upon request
What does a Master Datamodel Look Like?

**Master Datamodel Section**

[Diagram showing a section of a master datamodel]

**Master Datamodel Query**

[Table showing query prompts]

**Insert Query Prompt**

[Dialog box for creating new queries]

**NOTE:** Options to use existing connections when inserting a new query will always be available.
What is a Sync With Database?

- Performed via the datamodel menu on the top menu bar.
- Used primarily to synchronize the datamodel with any changes that have been made to the database (added/removed fields or tables, renamed items, etc.).
- Can be used to “re-point” one or more topics to an altered topic physical name so as to preserve requests/filters/joins.
- Treats any renaming of an item as if the item was instead dropped and a new item was then added in its place.
- Be careful of section dependencies and **ALWAYS** save first.
What does Sync With Database Look Like?

NOTE: Using this option will completely destroy virtually all items (joins, requests, filters, etc.) that are associated with topics or topic items that no longer exist by their original name on the datamodel. All but the request items are simple to replace but requests can tie deeply into section dependencies. To combat this preserve your requests by altering their formula to static or aggregated values. Once the datamodel is synchronized these items can then be re-pointed to any appropriate references.
How do We use Facts & Dimensions?

**FACTS**
- Represent items to be measured and/or aggregated (sales, student counts, etc.)
- Usually found on designated fact tables
- Should be added to datamodels first
- Should be at the center of datamodels whenever possible

**DIMENSIONS**
- Represent grouping/category items to measure by (by dept., by term, etc.)
- Usually found on designated dimension tables
- Should be added to datamodels last
- Should not be at the center of datamodels whenever possible

**NOTE:** When aggregating you will always see fact values only for each distinct combination of dimensional values
What does a Star Schema Look Like?

Star schemas have facts in the middle, dimensions as the outer points of the “star”. They can also potentially have “snowflake” dimensions stemming from the primary dimensions or multiple facts at their center:
How can We See our SQL?

NOTE: Displayed “Custom SQL” is masked (not 100% true SQL)
What Other Features can be Used?

- Process
- Ordering
- Customize
- Filters
What Other Features cannot be Used?

**NOTE:** Contact the Reporting Services Team if you wish to incorporate these features in your reports.
IS THIS THING OVER YET?

This concludes this presentation of:

Hyperion Querying (b)
Advanced Queries

Questions?