UNDERSTANDING SQL AS A FOUNDATION FOR HYPERION QUERYING
What is SQL Anyway?

- SQL stands for “Structured Query Language”
- It is used by all reporting tools that access standard relational databases (RDB’s)
- It communicates the definition of the desired output to the database
- It deals in “clauses” (i.e. SELECT, FROM, etc.)
- It varies slightly depending on database type (DB2, SQL Server, Oracle, etc.)
- Its basic commands/clauses/expressions conform to ANSI (American National Standards Institute) standards
Why do We Care What SQL is?

- Hyperion and all other reporting apps use SQL to communicate with databases.
- Knowing how SQL behaves is the key to knowing how your Hyperion query will behave.
- Knowing how to read SQL will help tremendously in troubleshooting your Hyperion queries.
- Understanding SQL is the only way to fully comprehend the logic of more advanced Hyperion features (i.e. sub-queries, derivable queries, etc.).
What don’t We Need to Know?

- You do not need to know how to write SQL because you have no platform/application to run SQL directly.
- You do not need to know about SET or similar query definition statements.
- You do not need to learn types of SQL statements other than SELECT because these are not used in reporting.
- You do not need to memorize all SQL functions as long as you have access to a functions reference guide (many are available online for free).
- You do not need to know about additional concepts that do not apply in Hyperion (i.e. stored procedures, using asterisk to return all columns, etc.).
What is a SELECT Statement?

- SELECT statements are blocks of SQL code that are initiated with a SELECT clause and therefore start with the word “SELECT”
- SELECT statements define output to be returned to the application from the database
- SELECT statements can also be referred to as “SQL Queries” or just “queries”
What is a Clause?

Clauses are used to separate SQL statements into sections, each with different functionalities. They are declared with the keyword/clausename at the beginning of the clause. The two required clauses in any SQL statement are...

- **SELECT** – Defines columns/fields to be returned
- **FROM** – Defines tables (etc.) where the fields in the other clauses can be found and defines some outer joins
What does a SELECT statement look like?

The most basic of select statements simply selects one or more fields from a table defined in the FROM clause as follows…

```
SELECT sysdate FROM dual;
```

…returns:

<table>
<thead>
<tr>
<th>sysdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-JAN-11</td>
</tr>
</tbody>
</table>

Note: SYSDATE is the formula for “current date” and DUAL is a table designed for developers to pull a single row of data (for Oracle Databases Only)
What Items Can Go Into the SELECT Clause?

- Commas (used to separate items listed in the SELECT clause)
- Database fields/columns from tables (etc.)
- Calculated fields such as ‘Hello’ for the text “Hello” (use single quotes) or 1+5 for the number “6”
- Acceptable keyword references (i.e. sysdate) and function references (i.e. sum)
- Any combination of the items listed above
- Alias column headers for any items listed (use keyword AS and single quotes)
What do SELECT Clause Items Look Like?

SELECT sum(sales) AS ‘Sales’, sysdate – 1 AS ‘Yesterday’ FROM dual;

...returns:

<table>
<thead>
<tr>
<th>Sales</th>
<th>Yesterday</th>
</tr>
</thead>
<tbody>
<tr>
<td>61,923</td>
<td>31-DEC-10</td>
</tr>
</tbody>
</table>
What Items Can Go Into the FROM Clause?

- Source Tables (including views, synonyms and DB aliases)
- Table aliases (separate from table name with a space)
- Information that creates outer join logic (declarations and definitions)

Note: Tables in the FROM clause must be preceded by the table owner name and a period.
What do FROM Clause Items Look Like?

SELECT
  SUM(AL1.Amount),
  AL2.Department
FROM
  DBO.My_Fact AL1
  LEFT OUTER JOIN
  DBO.My_Dim AL2
ON (AL1.Main_Sid=AL2.Main_Sid);

Note: Joins and table alias referencing will be explained further in the next section
What Have We Learned So Far?

Think about it…

- Why does SQL have clauses?
- What defines the column count of a query?
- What can be found in a FROM clause besides source table references and aliases?

**BONUS:** What defines the row count of a query?
How does SQL Handle Multiple Tables?

- Use joins to associate data from one table to another based on correlated fields (SID’s or business keys) called “join fields”
- All tables used in the FROM clause must be joined
- Use `table_name.field_name` (or `table_alias.field_name`) to reference fields from specific tables in any clause
- There are inner joins (aka “simple joins”) and outer joins (aka “left joins” or “right joins”) that are available depending upon your logical needs
How does a Join Work?

- Joins associate two tables based on one or more join fields to consolidate (or “look up”) information.

<table>
<thead>
<tr>
<th>Store_Sid</th>
<th>Day_Sid</th>
<th>Product_Sid</th>
<th>Amount_Sales</th>
<th>Unit_Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>4353</td>
<td>3636</td>
<td>7164</td>
<td>25789.35</td>
<td>1578</td>
</tr>
<tr>
<td>6823</td>
<td>9453</td>
<td>6294</td>
<td>45263.08</td>
<td>2822</td>
</tr>
<tr>
<td>9467</td>
<td>2847</td>
<td>4721</td>
<td>33714.96</td>
<td>1984</td>
</tr>
<tr>
<td>3954</td>
<td>3947</td>
<td>8352</td>
<td>35601.13</td>
<td>2118</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Store_Sid</th>
<th>Region_Sid</th>
<th>Store_Name</th>
<th>Store_Mgr</th>
</tr>
</thead>
<tbody>
<tr>
<td>3954</td>
<td>3865</td>
<td>Mercury</td>
<td>Smith</td>
</tr>
<tr>
<td>4125</td>
<td>7537</td>
<td>Venus</td>
<td>Tennant</td>
</tr>
<tr>
<td>4353</td>
<td>8315</td>
<td>Earth</td>
<td>Eccleson</td>
</tr>
<tr>
<td>6823</td>
<td>4831</td>
<td>Mars</td>
<td>Davidson</td>
</tr>
<tr>
<td>7215</td>
<td>7537</td>
<td>Jupiter</td>
<td>Baker</td>
</tr>
<tr>
<td>7846</td>
<td>3865</td>
<td>Uranus</td>
<td>Pertwee</td>
</tr>
<tr>
<td>9467</td>
<td>7537</td>
<td>Neptune</td>
<td>Hartnell</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Store_Name</th>
<th>Store_Mgr</th>
<th>Amount_Sales</th>
<th>Unit_Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth</td>
<td>Eccleson</td>
<td>25789.35</td>
<td>1578</td>
</tr>
<tr>
<td>Mars</td>
<td>Baker</td>
<td>45263.08</td>
<td>2822</td>
</tr>
<tr>
<td>Neptune</td>
<td>Hartnell</td>
<td>33714.96</td>
<td>1984</td>
</tr>
<tr>
<td>Mercury</td>
<td>Smith</td>
<td>35601.13</td>
<td>2118</td>
</tr>
</tbody>
</table>
What do Simple Joins Look Like?

Simple joins are added to a new-to-us clause called the WHERE clause which follows the FROM clause and sets the field from one table equal to a field from another.

```sql
SELECT AL2.Store_Name, AL2.Store_Mgr,
       AL1.Amount_Sales, AL2.Unit_Sales
FROM DBO.Sales_Fact AL1, DBO.Stores_Dim AL2
WHERE AL1.Store_Sid = AL2.Store_Sid;
```

...returns the reporting data from the previous slide
How does an Outer Join Work?

Outer joins perform the same functionality as inner joins but allow for null values to be returned in cases where no match is found for a join field value in one table or the other.

### Sales_Fact (Table)

<table>
<thead>
<tr>
<th>Store_Sid</th>
<th>Day_Sid</th>
<th>Product_Sid</th>
<th>Amount_Sales</th>
<th>Unit_Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>4353</td>
<td>3636</td>
<td>7164</td>
<td>25789.35</td>
<td>1578</td>
</tr>
<tr>
<td>6823</td>
<td>9453</td>
<td>6294</td>
<td>45263.08</td>
<td>2822</td>
</tr>
<tr>
<td>9467</td>
<td>2847</td>
<td>4721</td>
<td>33714.96</td>
<td>1984</td>
</tr>
<tr>
<td>3954</td>
<td>3947</td>
<td>8352</td>
<td>35601.13</td>
<td>2118</td>
</tr>
</tbody>
</table>

### Stores_Dim (Table)

<table>
<thead>
<tr>
<th>Store_Sid</th>
<th>Region_Sid</th>
<th>Store_Name</th>
<th>Store_Mgr</th>
</tr>
</thead>
<tbody>
<tr>
<td>3954</td>
<td>3865</td>
<td>Mercury</td>
<td>Smith</td>
</tr>
<tr>
<td>4125</td>
<td>7537</td>
<td>Venus</td>
<td>Tennant</td>
</tr>
<tr>
<td>4353</td>
<td>8315</td>
<td>Earth</td>
<td>Eccleson</td>
</tr>
<tr>
<td>6823</td>
<td>4831</td>
<td>Mars</td>
<td>Davidson</td>
</tr>
<tr>
<td>7215</td>
<td>7537</td>
<td>Jupiter</td>
<td>Baker</td>
</tr>
<tr>
<td>7846</td>
<td>3865</td>
<td>Uranus</td>
<td>Pertwee</td>
</tr>
<tr>
<td>9467</td>
<td>7537</td>
<td>Neptune</td>
<td>Hartnell</td>
</tr>
</tbody>
</table>

### Output (Query Data)

<table>
<thead>
<tr>
<th>Store_Name</th>
<th>Store_Mgr</th>
<th>Amount_Sales</th>
<th>Unit_Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth</td>
<td>Eccleson</td>
<td>25789.35</td>
<td>1578</td>
</tr>
<tr>
<td>Mars</td>
<td>Baker</td>
<td>45263.08</td>
<td>2822</td>
</tr>
<tr>
<td>Neptune</td>
<td>Hartnell</td>
<td>33714.96</td>
<td>1984</td>
</tr>
<tr>
<td>Mercury</td>
<td>Smith</td>
<td>35601.13</td>
<td>2118</td>
</tr>
<tr>
<td>Venus</td>
<td>Tennant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jupiter</td>
<td>Baker</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uranus</td>
<td>Pertwee</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Joined (Right/Outer)**
What do Outer Joins Look Like?

As we saw when we looked at the FROM clause we can define there any outer joins using the following syntax in place of listing the items as comma-separated values.

```sql
...Table_Owner.Table_1
  LEFT OUTER JOIN
  Table_Owner.Table_2
  ON (Table_1.Join_Field = Table_2.Join_Field)...
```
What does an Outer Joined Query Look Like?

```
SELECT
    AL2.Store_Name, AL2.Store_Mgr,
    AL1.Amount_Sales, AL2.Unit_Sales
FROM
    DBO.Sales_Fact AL1
RIGHT OUTER JOIN
    DBO.Stores_Dim AL2
ON (AL1.Store_Sid = AL2.Store_Sid);
```

...returns the reporting data from the slide before last
Is There More to Know About Joins?

- Multiple join fields can be used by separating the individual conditions with the logical operator keywords like ‘AND’ & ‘OR’ (discussed on next slide)
- Use parenthesis around outer joins to then outer-join them to additional tables
- To tell “left from right” remember that a left join retrieves all rows from the first table that is specified and a right join retrieves all rows from the second table that is specified
- Additional logical options and functionality are available through joins (i.e. joining with other operators that are discussed later)
Is There More that the WHERE Clause Can do for Us?

- Filters can be inserted into the WHERE clause to limit data output.
- Multiple filters are separated by the logical operator keywords “**AND**” & “**OR**” (no quotes).
- Different operators can be used to filter on equals, greater than, etc.
- Parenthesis can be used to logically group filters together (be careful).

*Note: Hyperion uses the placeholder “VarLimit” (no quotes) to denote areas of the WHERE clause that are to be filled in at process-time with user-supplied values.*
What does the WHERE Clause Look Like?

```
SELECT
    AL2.Store_Name, AL2.Store_Mgr,
    AL1.Amount_Sales, AL2.Unit_Sales
FROM
    DBO.Sales_Fact AL1, DBO.Stores_Dim AL2
WHERE
    AL1.Store_Sid = AL2.Store_Sid AND
    (AL2.Amount_Sales > 35000 OR
    AL1.Store_Name = 'Earth');
```

...returns:

<table>
<thead>
<tr>
<th>Store_Name</th>
<th>Store_Mgr</th>
<th>Amount_Sales</th>
<th>Unit_Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth</td>
<td>Eccleson</td>
<td>25789.35</td>
<td>1578</td>
</tr>
<tr>
<td>Mars</td>
<td>Baker</td>
<td>45263.08</td>
<td>2822</td>
</tr>
<tr>
<td>Mercury</td>
<td>Smith</td>
<td>35601.13</td>
<td>2118</td>
</tr>
</tbody>
</table>
What Operators Can We Use in Our Filters?

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Simple equals</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Simple not-equals</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equals</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equals</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>Between (non-inclusive)</td>
</tr>
<tr>
<td>LIKE</td>
<td>Uses % as wildcard in text</td>
</tr>
<tr>
<td>IS NULL</td>
<td>Equals null/no-value (no second argument used)</td>
</tr>
</tbody>
</table>
What Have We Learned So Far?

Think about it…

- What are the two primary differences between outer joins and inner joins?
- How can we tell our left from our right when working with outer joins?
- What can be found in a WHERE clause besides simple joins?

**BONUS:** Why do we use AL1, AL2, etc. for table aliases in this class?
What About Functions?

- Functions can be used to alter and/or calculate values wherever field references are acceptable.
- Some functions cannot be used in some clauses (i.e. aggregate functions cannot be in the WHERE clause).
- Functions often take arguments provided within parenthesis.
- Functions will often be specific to your database type (DB2, Oracle, SQL Server, etc.).
What do Standard Functions Look Like?

**SUBSTR()** – DB2 function to return part of a text value

**Expression/Syntax:**
SUBSTR(MyTable.MyField, 3, 5)

**Returns:**
Characters from field “MyField” starting at the third character and running for a length of 5 characters

**DECODE()** – Oracle function to replace values with alternatives

**Expression/Syntax:**

**Returns:**
The value ‘Alpha’ if “MyField” equals ‘A’ or ‘Beta’ if it equals ‘B’, and ‘Unknown’ in all other cases

**FLOOR()** – SQL Server function to return numbers as rounded down

**Expression/Syntax:**
FLOOR(MyTable.MyField)

**Returns:**
Value of “MyField” as rounded down to the nearest whole number

**DATEDIFF()** – MySQL function to return the difference of 2 dates

**Expression/Syntax:**
DATEDIFF(MyTable.StartDt, MyTable.EndDt)

**Returns:**
The difference between the date value of “StartDt” and “EndDt” as whole days
What About Aggregate Functions?

Aggregate functions are used to summarize fact data (measures, counts, etc.) to the level of defined dimensional grouping fields ("by" items)

<table>
<thead>
<tr>
<th>Prod_Sales (Table)</th>
<th>Store_Name</th>
<th>Product_Line</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth</td>
<td>Books</td>
<td>543.26</td>
<td></td>
</tr>
<tr>
<td>Earth</td>
<td>Audio</td>
<td>826.98</td>
<td></td>
</tr>
<tr>
<td>Earth</td>
<td>Wares</td>
<td>1264.54</td>
<td></td>
</tr>
<tr>
<td>Mars</td>
<td>Audio</td>
<td>432.19</td>
<td></td>
</tr>
<tr>
<td>Mars</td>
<td>Wares</td>
<td>892.78</td>
<td></td>
</tr>
</tbody>
</table>

SELECT Store_Name, SUM(Sales) FROM Prod_Sales;

<table>
<thead>
<tr>
<th>Output (Query Data)</th>
<th>Store_Name</th>
<th>SUM(Sales)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth</td>
<td>2634.78</td>
<td></td>
</tr>
<tr>
<td>Mars</td>
<td>1324.97</td>
<td></td>
</tr>
</tbody>
</table>
What are the Basic Aggregate Functions in ANSI SQL?

<table>
<thead>
<tr>
<th>Function</th>
<th>Applies To</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM()</td>
<td>Numbers, Text*</td>
<td>Totals the values</td>
</tr>
<tr>
<td>AVG()</td>
<td>Numbers</td>
<td>Averages the values</td>
</tr>
<tr>
<td>MIN()</td>
<td>Numbers, Date/Time, Text*</td>
<td>Returns Minimum Value</td>
</tr>
<tr>
<td>MAX()</td>
<td>Numbers, Date/Time, Text*</td>
<td>Returns Maximum Value</td>
</tr>
<tr>
<td>COUNT(*)</td>
<td>All</td>
<td>Returns Count of Values (rows)</td>
</tr>
</tbody>
</table>

*Additional aggregate functions are available as both standard ANSI and database-specific.
What is a GROUP BY Clause?

- The GROUP BY clause is used to define the dimensional grouping fields (“by” items).
- The GROUP BY clause is required when using aggregate functions if more than one dimensional grouping field is selected.
- The GROUP BY clause must contain comma-separated references to all non-aggregated items from the SELECT clause except for static values (i.e. ‘Hello’, 10, sysdate, etc.).
- Non-aggregate functions can be used as GROUP BY items.
What does a GROUP BY Clause Look Like?

SELECT
AL2.Store_Name, AL2.Store_Mgr,
SUM(AL1.Amount_Sales), sysdate
FROM
DBO.Sales_Fact AL1, DBO.Stores_Dim AL2
WHERE
AL1.Store_Sid = AL2.Store_Sid
GROUP BY
AL2.Store_Name, AL2.Store_Mgr

Dimensional Grouping Fields

GROUP BY Clause Declaration

GROUP BY Items

Static Field
What Have We Learned So Far?

Think about it…

- What is the difference between standard and aggregate functions?
- What is the key determining factor that dictates which functions will be available to you and what their proper syntax should be?
- What are the only items that can/must go into a GROUP BY clause?

**BONUS:** From where within a Hyperion query can we manipulate the GROUP BY clause?
What is a HAVING Clause?

The HAVING clause comes after the GROUP BY clause or after the WHERE clause if there is no GROUP BY. It is used to establish limits on aggregated fields since these limits cannot go directly into the WHERE clause…

```
(SELECT...FROM...WHERE...GROUP BY...)  
...HAVING
SUM(AL1.Sales) > 35000...
```

…returns aggregated rows where the total of sales is greater than 35000.
What is an ORDER BY Clause?

The ORDER BY clause is used to sort data and comes at the end of a SELECT statement. It references only items from the SELECT clause and does so as comma-separated and by alias or index number (1 for the first SELECT item, 2 for the second, etc.). Optionally the keyword “DESC” (no quotes) can follow any ORDER BY item to change the sort order from ascending (default) to descending. Sorting is applied in the order in which the sort items are listed in the ORDER BY clause.
What do All of Our Clauses Look Like Together?

```
SELECT
    AL2.Store_Name, AL2.Store_Mgr,
    SUM(AL1.Amount_Sales), sysdate
FROM
    DBO.Sales_Fact AL1
RIGHT OUTER JOIN
    DBO.Stores_Dim AL2
ON (AL1.Store_Sid=AL2.Store_Sid)
WHERE
    AL1.Store_Name <> 'Earth'
GROUP BY
    AL2.Store_Name, AL2.Store_Mgr
HAVING
    SUM(AL1.Amount_Sales) > 0
ORDER BY
    3 DESC, 1
```
Can We Embed SELECT Statements into Our Clauses?

- **Sub-query**: Embedding a SELECT statement into the WHERE clause that returns a value for use in filtering
- **Derivable Query (Inline View)**: Embedding a SELECT statement into the FROM clause (in parenthesis and with an alias) to have its output treated as a virtual dataset
- **Sub-Select**: Embedding a SELECT statement into a SELECT clause to return values outside of the scope of the original query (not recommended/used in Hyperion)
What Have We Learned So Far?

Think about it…

- What items should not go into a HAVING clause?
- How are items referenced within an ORDER BY clause?
- What clauses are available to us in SQL?

**BONUS:** Why doesn’t Hyperion generally support sub-select functionality?
How Can We Format SQL for Easy Reading?

Go to…

http://www.dpriver.com/pp/sqlformat.htm?ref=g_wangz

…on the web
Is this Thing Over Yet?

This concludes this presentation of...

Hyperion IR Reporting 101

UNDERSTANDING SQL AS A FOUNDATION FOR HYPERION QUERYING

Questions?