IBM TSM SQL Workshop
(A Practical Approach)

Robert Brilmayer, PLCS/STORServer
Laura Buckley, STORServer
Andy Raibeck, IBM

Agenda

- Brief SQL tutorial
- Anatomy of a SQL Database
- TSM’s SQL Interface
  - SELECT Syntax
  - Clauses, Operators and Functions
  - Basic Joins
- Challenge us…
A Brief SQL Tutorial

- Structured Query Language maintained by ANSI
- Used to communicate with databases
- Universal interface that can:
  - View (select)
  - Manipulate (fetch, update)
  - Add (insert) data to databases*

*TSM’s SQL Interface only supports viewing data (select)

Anatomy of a Relational Database

- In a relational database, one piece of information relates to another
- Data is stored in columns in tables
- The data in a column in one table can relate to data in a column in another table
- A row represents a single entry in a table
Example of a Table

**TABLE = ASSOCIATIONS**

**COLUMNS**

<table>
<thead>
<tr>
<th>DOMAIN_NAME</th>
<th>SCHEDULE_NAME</th>
<th>NODE_NAME</th>
<th>CHG_TIME</th>
<th>CHG_ADMIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCHANGE</td>
<td>DAILY_FULL</td>
<td>EXCH_STORMAIL</td>
<td>7/29/2003 13:21</td>
<td>ADMIN</td>
</tr>
<tr>
<td>MOBILE</td>
<td>DAILY_INCR</td>
<td>BEBO</td>
<td>5/7/2003 16:30</td>
<td>ADMIN</td>
</tr>
<tr>
<td>MOBILE</td>
<td>DAILY_INCR</td>
<td>ROSEBUD</td>
<td>5/7/2003 16:30</td>
<td>ADMIN</td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>DAILY_INCR</td>
<td>SSHQ</td>
<td>5/7/2003 17:12</td>
<td>ADMIN</td>
</tr>
</tbody>
</table>

**ROWS**

**RELATIONSHIPS BETWEEN TABLES**

**CLIENT SCHEDULES**

<table>
<thead>
<tr>
<th>DOMAIN_NAME</th>
<th>SCHEDULE_NAME</th>
<th>DESCRIPTION</th>
<th>ACTION</th>
<th>OBJECTS</th>
<th>STARTDATE</th>
<th>STARTTIME</th>
<th>DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCHANGE</td>
<td>DAILY_FULL</td>
<td>Daily increment</td>
<td>INCREMENTAL</td>
<td>c:\progra~</td>
<td>7/29/2003 23:10:00</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>MOBILE</td>
<td>DAILY_INCR</td>
<td>Daily increment</td>
<td>INCREMENTAL</td>
<td>11/14/2001 00:00:00</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOBILE</td>
<td>DAILY_INCR</td>
<td>Daily increment</td>
<td>INCREMENTAL</td>
<td>11/14/2001 00:00:00</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOBILE</td>
<td>DAILY_Incr</td>
<td>Daily increment</td>
<td>INCREMENTAL</td>
<td>7/11/2003 16:27:48</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>DAILY_INCR</td>
<td>Daily increment</td>
<td>INCREMENTAL</td>
<td>11/14/2001 00:00:00</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>TEST</td>
<td>Testing options</td>
<td>INCREMENTAL</td>
<td>7/11/2003 00:00:00</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>DAILY_INCR</td>
<td>MSSQL agent</td>
<td>COMMAND</td>
<td>c:\progra~</td>
<td>7/3/2003 23:30:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>STORBASEFULL</td>
<td>MSSQL agent</td>
<td>COMMAND</td>
<td>c:\progra~</td>
<td>6/2/2003 22:00:00</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>STANDARD</td>
<td>ARCH-TEST</td>
<td>Archive</td>
<td>INCREMENTAL</td>
<td>5/30/2003 16:17:00</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STANDARD</td>
<td>DAILY_INCR</td>
<td>Daily increment</td>
<td>INCREMENTAL</td>
<td>11/14/2001 00:00:00</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ASSOCIATIONS**

<table>
<thead>
<tr>
<th>DOMAIN_NAME</th>
<th>SCHEDULE_NAME</th>
<th>NODE_NAME</th>
<th>CHG_TIME</th>
<th>CHG_ADMIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCHANGE</td>
<td>DAILY_FULL</td>
<td>EXCH_STORMAIL</td>
<td>7/29/2003 13:21</td>
<td>ADMIN</td>
</tr>
<tr>
<td>MOBILE</td>
<td>DAILY_INCR</td>
<td>BEBO</td>
<td>5/7/2003 16:30</td>
<td>ADMIN</td>
</tr>
<tr>
<td>MOBILE</td>
<td>DAILY_INCR</td>
<td>ROSEBUD</td>
<td>5/7/2003 16:30</td>
<td>ADMIN</td>
</tr>
<tr>
<td>MOBILE</td>
<td>DAILY_INCR</td>
<td>SSHQ</td>
<td>5/7/2003 17:12</td>
<td>ADMIN</td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>DAILY_INCR</td>
<td>SSHQ</td>
<td>5/7/2003 17:12</td>
<td>ADMIN</td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>DAILY_INCR</td>
<td>SSEXT</td>
<td>5/7/2003 17:12</td>
<td>ADMIN</td>
</tr>
</tbody>
</table>
TSM SQL Interface

- Supports the SQL SELECT query only
  - Requires a “minimum” of 4MB of free space in the database
  - Complicated queries may take a long time to complete and can interfere with server operations
  - You cannot issue SELECT queries from the server console (admin command line only)

---

TSM SQL Interface

- “Mostly” conforms to standard SQL
  - Subset of the SQL92 and SQL93 ANSI standards
- Does NOT support:
  - UNION
  - INTERSECT
  - EXCEPT
  - Correlated subqueries (returning multiple values)
  - Semicolon cannot be used as a command terminator
SELECT Syntax

SELECT column|expression [,n..]  
FROM tablename {,n...}

- column refers to a column in a table (  
  - * is allowed as a wildcard to select all columns in a table  
- expression refers to functions that allow you  
  manipulate the data being returned  
- [,n...] indicates that you may specify one or more  
  columns or expressions  
- FROM clause indicates which table to search  
- You can specify one or more tablename

Simple SELECT Example

SELECT STARTTIME  
FROM CLIENT_SCHEDULES

- Display the contents of the STARTTIME column from  
  every row in the CLIENT_SCHEDULES table  
- Column and table names cannot be abbreviated  
- Column names are displayed in the order they are  
  entered on the SELECT statement  
- Much of the data in TSM is stored in uppercase and  
  must be entered in uppercase in SELECT statements
TSM Database Catalog

- TSM has three system catalog tables so that you can view the tables, columns and enumerated data types available
  - SYSCAT.TABLES
  - SYSCAT.COLUMNS
  - SYSCAT.ENUMTYPES

Viewing the TSM System Catalog

- To view available table names:
  `SELECT * FROM SYSCAT.TABLES`
- To view all column names within tables
  `SELECT tabname, colname FROM SYSCAT.COLUMNS`
- To view the valid values and order for enumerated types
  `SELECT * FROM SYSCAT.ENUMTYPES`
Manipulating the Results

- You may not want all of the data in all of the columns all of the time so:
  - SQL provides *clauses, operators, and functions*
  - These allow you to sort, order, filter, and compute the data on a select command

### Clauses, Operators, and Functions

<table>
<thead>
<tr>
<th>ALL</th>
<th>CURRENT_DATE</th>
<th>GROUP BY</th>
<th>ORDER BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY</td>
<td>CURRENT_TIME</td>
<td>HAVING</td>
<td>POSITION</td>
</tr>
<tr>
<td>AVG</td>
<td>CURRENT_TIMESTAMP</td>
<td>IN</td>
<td>SOME</td>
</tr>
<tr>
<td>AS</td>
<td>CURRENT_USER</td>
<td>JOIN</td>
<td>SUBSTRING</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>DISTINCT</td>
<td>LIKE</td>
<td>SUM</td>
</tr>
<tr>
<td>CASE</td>
<td>EXISTS</td>
<td>MAX</td>
<td>TRIM</td>
</tr>
<tr>
<td>CAST</td>
<td>EXTRACT</td>
<td>MIN</td>
<td>WHERE</td>
</tr>
<tr>
<td>COUNT</td>
<td>FROM</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

*Supported by TSM SELECT*
Clauses (not the Santa kind..)

- A **clause** is a part of a SQL statement (i.e. SELECT column1,column2)
- Clauses combine to form an entire SQL statement
- For example, you can combine the SELECT clause and FROM clause to form a statement
  - SELECT also refers to the statement itself

WHERE Clause

- The **WHERE** clause allows you to filter out rows from the results
  - I want this, this and this, but only where this condition is true
  - I want to see all the volumes on which the client called CARROLL has data:

```sql
SELECT NODE_NAME,VOLUME_NAME
FROM VOLUMEUSAGE
WHERE NODE_NAME='CARROLL'
```
Comparison Operators (use with WHERE)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example (WHERE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Equal (used with NULL)</td>
<td>NODE_NAME IS NULL</td>
</tr>
<tr>
<td>IS NOT</td>
<td>Not equal (used with NULL)</td>
<td>VERSION IS NOT NULL</td>
</tr>
<tr>
<td>=</td>
<td>Equal</td>
<td>NODE_NAME='CARROLL'</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not Equal</td>
<td>NODE_NAME&lt;&gt;'CARROLL'</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>LOGICAL_MB &lt; 5000</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>LOGICAL_MB &gt; 5000</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>LOGICAL_MB &lt;= 5000</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>LOGICAL_MB &gt;= 5000</td>
</tr>
</tbody>
</table>

Logical Operators

- Logical operators separate two or more conditions in the WHERE clause
  - **LIKE** is used with the wildcard % to match all occurrences
    `SELECT * FROM NODES
    WHERE NODE_NAME LIKE 'C%'`
  - **AND** means that the expressions on both sides must be true to return TRUE
    `SELECT * FROM NODES WHERE NODE_NAME='CARROLL' AND
     PLATFORM_NAME='WinNT'`
More Logical Operators

- You can use OR to sum up a series of conditions. If any of the comparisons is true, OR returns TRUE.

  ```sql
  SELECT * FROM NODES
  WHERE NODE_NAME='CARROLL' OR 'DODSON'
  ```

- Use IN to replace multiple OR's.

  ```sql
  SELECT * FROM NODES WHERE NODE_NAME IN ('CARROL','DODSON','LEWIS','CHARLES')
  ```

- Use BETWEEN to get a range.

  ```sql
  SELECT NODE_NAME FROM OCCUPANCY
  WHERE LOGICAL_MB BETWEEN 5000 AND 10000
  ```

ORDER BY Clause

- The ORDER BY clause is used to sort the rows prior to displaying them:

  ```sql
  SELECT NODE_NAME, PLATFORM_NAME
  FROM NODES
  ORDER BY PLATFORM_NAME
  ```

- You can specify that the results be sorted in ascending or descending order:

  ```sql
  SELECT NODE_NAME, TYPE, FILESPACE_NAME, LOGICAL_MB FROM OCCUPANCY
  ORDER BY LOGICAL Mb DESC
  ```
Functions

- Functions allow you to aggregate data and operate on strings, numeric and date and time values.
- Aggregate functions perform operations on values from selected rows to produce a single value.
  - They include COUNT(*), SUM, AVG, MAX, and MIN.
  - COUNT(*) is useful for finding the number of rows that match a query.

Timestamp and CAST Functions

- Example of date/time and cast function – displays nodes that have not accessed the server in a specified ($1) number of days.
- CAST’s the timestamp as decimal for processing:

```
SELECT NODE_NAME, LASTACC_TIME FROM NODES
WHERE CAST((CURRENT_TIMESTAMP - LASTACC_TIME) DAYS AS DECIMAL) >= $1
```
GROUP BY Clause

- The GROUP BY clause allows you to combine the rows being selected into logical groups
- Normally used with aggregate functions

```
SELECT NODE_NAME, SUM(NUM_FILES) AS #_OF_FILES, SUM(LOGICAL_MB) AS TOTAL_MB FROM OCCUPANCY GROUP BY NODE_NAME
```

- When using aggregate functions, you need to name the columns (i.e. AS #_OF_FILES)

HAVING Clause

- HAVING always follows the GROUP BY clause
- Use the HAVING clause to filter the results of the GROUP BY clause

```
SELECT NODE_NAME, SUM(NUM_FILES) AS #_OF_FILES, SUM(LOGICAL_MB) AS TOTAL_MB FROM OCCUPANCY GROUP BY NODE_NAME HAVING SUM(LOGICAL_MB)>1000
```
Joining Tables

- Helps you see how data relates between tables
- There are different types of joins, depending on the data you are trying to relate
- Use an alias to specify which column you want to display when joining tables with columns of the same name
- The keyword DISTINCT specifies only unique rows will be retrieved and prevent duplicates

JOIN Example

- To see which schedules a node is associated with join ASSOCIATIONS with CLIENT_SCHEDULES
- DOMAIN_NAME and SCHEDULE_NAME are common columns
- We’ll use the alias C.SCHEDULE_NAME, to indicate which SCHEDULE_NAME to return
Join Example – continued

SELECT DISTINCT NODE_NAME, C.SCHEDULE_NAME
FROM ASSOCIATIONS A, CLIENT_SCHEDULES C
WHERE A.SCHEDULE_NAME = C.SCHEDULE_NAME

- This is an example of an inner join or equi-join
- The goal is to match the values of a column in one table to the corresponding values in the second table (schedule_name)

Challenge Us

- That’s was just an overview of TSM SQL and some basic examples
- There is a lot more you can do with the tool
- We are now going to present some of the solutions to questions you provided earlier in the week
SQL References

- TSM SQL Guide
  - More info on clauses, operators, and functions
  - More examples
  - Using ODBC
- ADSM.ORG
- SQL Books and self-study courses