SQL Query Formatting Standards

JFORCES is a data-driven simulation built upon a relational database. As such SQL is ubiquitous, especially in scenario generation and data analysis. To improve maintenance it's essential that JFORCES programmers follow certain guidelines when constructing SQL statements. These standards must be maintained whether the SQL is in SQL-specific scripts or embedded in other languages (e.g. C, TCL, sh).

First, when a statement is too long to place on one line, put a break in it in an intelligent place. Don't rely on the editor's wrapping to put everything on a page. If you do it will be impossible to use any included indentation to help the next programmer understand the query.

Second, use carriage returns and indentation to format the SQL statement according to its major components. For example use the format:

```
select beam.antenna_uid, count(*) as count
   from DA_SCRM_BEAMS beam, DA_SCRM_ANTENNAS ant
   where beam.run_id=$run_id  and
         beam.sub_run_id=1 and beam.antenna_uid=ant.uid
   group by beam.SCRM_antenna_id
```

The indentation logic is:

- The major “select” (or equivalent SQL operation) is leftmost. This helps distinguish multiple SQL statements in the same file
- The secondary components (e.g. “from”, “where”, “group”, “order by” when used) are at the first level of indentation relative to the principle SQL command (e.g. “select”)
- The continued components (e.g. The continued where substatements) are indented at after the secondary component (e.g. After the “where “ in the initial line).
- If there are sub-statements (e.g. Multiple selects in a union) indent each sub-statement using the above logic (i.e. Indented right-ward from the main statement) and indent within this sub-statement using the above logic, as done in this example:

```
select      carriertype, carriernum, objname, objnumber, launchpt, launchtime
              from      CRSMSLOBJECT
              where     scenario = '%s'
              union \n"
                (select  a.objname, a.objnumber, a.assocname, a.assocnumber, 0, m.takeofftime
                   from ASSOCIATEOBJ a, MOBILEDATA m
                   where a.scenario = '%s' and a.scenario = m.scenario and
                         m.objname=a.assocname and m.objnumber = a.assocnumber and
                         taskflag in (%d) and m.takeofftime>0 \n"
                   except all
                      select c.carriertype, c.carriernum, c.objname,
                             c.objnumber, 0, m.takeofftime
                          from CRSMSLOBJECT c, MOBILEDATA m
                          where c.scenario = '%s' and c.scenario = m.scenario and
```

```
                                                m.objname=c.objname and m.objnumber = c.objnumber)
```

It is recommended that “and”s be placed on the end of the prior line. The logic is that the SQL statement will throw an error if you inadvertently delete the final line. It's easier to catch and fix these than it is to try to figure out why the results are wrong when there's no error. Also, it's acceptable to put multiple filters (e.g. “ beam.sub_run_id=1 and beam.antenna_uid=ant.uid” on the same line. In fact it's encouraged if these filters are logically related.

When multiple tables are referenced in a statement define the source of every field using either the source table name or a reasonable mnemonic. The following two examples show the original SQL statement using non-intuitive table mnemonics and then showing reasonable mnemonics. I hope you agree the second one in easier to trace:

Original query:

```
     select a.SCRM_antenna_id, count(*) from DA_SCRM_BEAMS a,
     DA_SCRM_ANTENNAS b where a.run_id=$run_id and a.sub_run_id=1 and
     a.SCRM_antenna_id=b.uid group by a.SCRM_antenna_id
```

Improved query:

```
     select beam.antenna_uid, count(*) as count
        from DA_SCRM_BEAMS beam, DA_SCRM_ANTENNAS ant
        where beam.run_id=$run_id  and
              beam.sub_run_id=1 and beam.antenna_uid=ant.uid
        group by beam.SCRM_antenna_id
```

This was a particularly ugly example because table “a” in the first query referred to DA_SCRM_BEAMS and “b” referred to DA_SCRM_ANTENNAS. Talk about confusing! Remember that the next programmer might have to look through 1000's of lines of code while reviewing data analysis queries (as was the case here). Have some consideration people!

It's recommended that the source table be identified for each field. While SQL doesn't require this you're forcing the next programmer (or yourself a year later) to review the structure of all of the referenced tables to identify the source if you don't provide it in-line.

Field Naming Conventions:

We're starting to add unique identifiers to database tables based upon sequence draws. This is typically done in the data collection tables, where adding a unique ID can eliminate repeated compound joins to retrieve data on a specific entity (asset, comm link, beam, etc.) For example, we used to use a run-specific ID to identify assets and require that all queries for this data have the format:

```
      (sql select) …    where da_objdef.run_id = (x) and da_objdef.sub_run_id =
    (y) and  da_objdef.asset_id = (z).
```

This was wasteful and error-prone. So we added a unique ID (UID) column to this table so once you had this UID you could access the information with a simple query, as follows:

```
 (sql select) …     where da_objdef.uid = (z).
```

But it's often hard to distinguish the run IDs from these database unique IDs. So the following conventions shall be maintained:

1. The unique ID for the current table will always be named “UID”
2. A reference to a UID in a different table will use a source descriptor and “_UID”. For example, the DA_SCRM_ANTENNAS table references the DA_OBJDEF UID to specify the asset an antenna is mounted on. The field name for that is “ASSET_UID”. This tells us that the field refers to the host asset and already points to the the UID in that table (to simplify access).
3. Often it's necessary to collect the run ID for these referential tables during runtime and then assign the correct UID as part of post-processing. See routine SCRM_prep_da_data in file da_table_prep.tcl for some examples. In this case the same field naming logic will be used except the suffix will be “_ID”. Thus the field “ASSET_ID” in DA_SCRM_ANTENNAS is used to collect the runtime asset ID during runtime and the “ASSET_UID” field is initially set to “0” (used to indicate the data is uninitialized for this run). During post processing the presence of 0's in the ASSET_UID field is checked, and if any are found the “ASSET_UID” field is set from the DA_OBJDEF table by matching the run_id, sub_run_id, and asset_id fields. Again, the “_UID” suffix is used for the database unique ID's, the “_ID” suffix is used for run identifiers.

<u>Language-Specific comments:</u>
These comments are specific to certain languages to help readability of either the original queries or the invoked queries stored in log files. They do NOT supersede the overall standards.

C/C++:
Use multiple lines in the source code and make sure it will be echoed out with the appropriate indentation. Here's an example:

```
sprintf( sql_string, " \n"
      "select      m.objname, m.objnumber \n"
      "   from     MOBILEDATA m, PLATFORMCHAR p,  MEDCROSSREF med \n"
      "   where    homebase='%s' and p.name=m.objname and med.mednum=%d and \n"
      "            p.medium=med.medval and \n"
      "            scenario='%s' \n"
      "   except all \n"
      "      select assocname, assocnumber from ASSOCIATEOBJ \n"
      "          where scenario='%s' and taskflag in (%d, %d)",
       base->name, airmednum, sim_controls->scenario,
       sim_controls->scenario, ASS_ONBOARD, ASS_TRANSPORT );
```

The SQL logic is apparent both in the original format and in the output (with real value inserted) saved in the log files.

Remember we output SQL statements both in C and TCL. So try to make sure this output is both readable and can easily be run in an interactive SQL session by cutting and pasting from these logs.