APPENDIX B

Sample Scripts

We have written some useful shell scripts to maintain monitored data. These scripts call the Operating System commands or DB2 UDB Monitoring Tools, format the outputs, and save them into the directory structure that we discussed in Chapter4.

You can use these sample shell scripts for your environment, or modify for your own purpose.

Executing db2look

The `db2look.ksh` script can be used to execute `db2look` with default parameters and place the output in a `states` subdirectory under the your home directory. A sample output is included in Chapter 4.

Here is the syntax:

```
db2look.ksh -c "comment" -d dbname [-p "params"] [-o N] [-v] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -p:   db2look parameters (default="-m -l -a -x -e -f")
        -o:   Save output in dir N under $RESULTS
                (0=Current dir,default; -1=Not Valid)
        -v:   Verbose (default is NOT to display db2look script)
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of output filename.
```

Here is the script:

```
#!/bin/ksh
# db2look.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname [-p \"params\"] [-o N] [-v] [-b]
Execute db2look tool to extract database definitions into an executable script
which can be used to restore the definitions.
    -c:   Comment placed on first line of Summary output file (required)
    -d:   Database name (required)
    -p:   db2look parameters (default=\"${DB2LOOK_PARAMS}\")
    -o:   Save output in dir N under \$RESULTS
    (0=Current dir,default; -1=Not Valid)
    -v:   Verbose (default is NOT to display db2look script)
    -b:   DONT save prior (old) results to bak directory (default=save)
    (Not saved anyway unless -o is greater than 0)
    Notes:
    Value for -d is part of output filename.
    "
```

```
}

# Constants

RESULTS=~/states
RESULTS_FILE=`basename $0 .ksh`
RES_EXT=".sql"

# Defaults

QUIET=1
DB_NAME=""
RESULTS_DIR=0# 0 defaults to current dir
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

DB2LOOK_PARAMS="-m -l -a -x -e -f"

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-d") shift; DB_NAME=$1; shift;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-v") shift; QUIET=0;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ $RESULTS_DIR -ge 0 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

if [ "$PARSE_ERROR" != "" ]
then
    echo ""
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`
RES_EXT="_${DB_NAME}${RES_EXT}"

if [ $RESULTS_DIR -gt 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
 fi

if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
    mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
    cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
    fi
fi

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $RES_OUT" || Q_OUTPUT="| tee -a $RES_OUT"
```

```
rm $RES_OUT 2>/dev/null

echo "[Creating: $RES_OUT]"

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ----------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--              `date`"$Q_OUTPUT
eval echo "-- "$Q_OUTPUT

eval echo "-- db2look -d $DB_NAME $DB2LOOK_PARAMS"$Q_OUTPUT
eval echo "-- "$Q_OUTPUT
eval db2look -d $DB_NAME $DB2LOOK_PARAMS$Q_OUTPUT
```

Executing GET DBM CFG / GET DB CFG

This script executes a `GET DBM CFG` or `GET DB CFG` command and save the output, at the same time generating a script file which can return all of the parameters to their current value. The generated script can be executed by using the `db2 -tvf` command. In this way if you experiment with a number of different tunable parameters (not recommended in general, anyway), then you can always return to the values (state) which were saved in this script.

Here is the syntax:

```
sqlcache.ksh -c "comment" -d dbname [-o N] [-f] [-s N] [-sql]
             [-t] [-w] [-q] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -o:   Save output in dir N under $RESULTS
                (0=Current dir; -1=Not Saved,default)
        -f:   Save each SQL stmt to a different file in the $QUERIES dir
        -s:   Display N characters of SQL stmt (-1=all, default)
        -sql: Only display SQL statements, without statistics
        -t:   DONT display Timing stats
        -w:   DONT display Row Count stats
        -q:   Quiet (default is to display output)
        -r:   Don't get snapshot, reuse existing snapshot output file
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of snapshot output filename.
        Values for -d, -s, -f and -sql are part of Summary output filename.
        Timing and Row Count statistics won't be shown if "Not Collected".
        In most cases, best viewed in 80 or 132 column window.
```

Here is the script:

```
#!/bin/ksh
# sqlcache.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" [-d dbname] [-o N] [-v] [-r] [-b]
Create an SQL script which can be used to restore the current settings of the
DBM CFG or DB CFG.
    -c:   Comment placed on first line of Summary output file (required)
    -d:   Database name, indicates to use DB CFG (default is DBM CFG)
    -o:   Save output in dir N under \$RESULTS
    (0=Current dir,default; -1=Not Valid)
    -v:   Verbose (default is NOT to display generated SQL script)
    -r:   Don't get DB/M CFG, reuse existing output file
    -b:   DONT save prior (old) results to bak directory (default=save)
    (Not saved anyway unless -o is greater than 0)
    Notes:
    Output of DB/M CFG also saved in -o directory.
    Value for -d is part of output filename and generated script filename.
```

```
    "
}

# Constants

RESULTS=~/states
RESULTS_FILE="dbm_cfg"
RES_EXT=".out"
SUM_EXT=".sql"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

QUIET=1
DB_NAME=""
RESULTS_DIR=0# 0 defaults to current dir
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-d") shift; DB_NAME=$1; shift;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-v") shift; QUIET=0;;
    "-r") shift; REUSE_OUT=1;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ $RESULTS_DIR -ge 0 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

if [ "$PARSE_ERROR" != "" ]
then
    echo ""
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

if [ "$DB_NAME" != "" ]
then
    DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`
    RESULTS_FILE="db_cfg"
    RES_EXT="_${DB_NAME}${RES_EXT}"
    SUM_EXT="_${DB_NAME}${SUM_EXT}"
fi

if [ $RESULTS_DIR -gt 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $REUSE_OUT -eq 1 -a ! -f $RES_OUT ]
then
    echo "Can't reuse $RES_OUT - Missing"
    exit
fi
```

```
if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
    mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
    [ $REUSE_OUT -eq 0 ] && \
    cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
    cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
fi

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

echo "[Creating: $SUM_OUT]"

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ----------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--              `date`"$Q_OUTPUT
eval echo "-- "$Q_OUTPUT

if [ $REUSE_OUT -eq 0 ]
then
    echo "[Creating: $RES_OUT]"
    echo ""
    if [ "$DB_NAME" = "" ]
    then
    eval echo "-- db2 get dbm cfg"$Q_OUTPUT
    db2 get dbm cfg > $RES_OUT
    else
    eval echo "-- db2 get db cfg for $DB_NAME" $Q_OUTPUT
    db2 get db cfg for $DB_NAME> $RES_OUT
    fi
else
    echo "[Reusing: $RES_OUT]"
fi

eval awk -f $AWKSCRIPT $RES_OUT $Q_OUTPUT
```

The upd_cfg.ksh script uses upd_cfg.awk file whose source is the following:

```
# run: db2 get dbm cfg | awk -f upd_cfg.awk > upd_dbm.sql
# run: db2 get db cfg for xx | awk -f upd_cfg.awk > upd_db_xx.sql

BEGIN{once = 0 ; set_NEWLOGPATH=0 }

/Database Manager Configuration/ && (once == 0){
    print "UPDATE DBM CFG USING";
    once = 1;
    FS=" = ";
}

/Database Configuration for Database/ && (once == 0){
    print "UPDATE DB CFG FOR " $NF " USING";
    once = 1;
    FS=" = ";
}

{# Print the original line as a comment
    print "--" $0;
}

# Look for configurable parameters in parenthases
/\([A-Z]+[A-Z_0-9]*\)/{
    match( $1, /\([A-Z]+[A-Z_0-9]*\)/ );

    # pull out parameter name and current value
    parm = substr( $1, RSTART+1, RLENGTH-2);
    val  = $2;

    # If the value is blank, set to empty string
    if (val ~ /^ *$/) {
    # Exception made for NEWLOGPATH, since empty string forces
```

```
    # value of a default path
    if (parm == "NEWLOGPATH") {
    set_NEWLOGPATH=1;
    next; # Skip the rest, don't print anything, see below
    } else {
    val = "''";
    }
    } else {
    # Remove part of value btwn paren's. Force recalc (-1) when:
    # 1) value contains "(caluculated)"; 2) value starts with "MAX";
    # 3) entire value was between paren's
    gsub( /\(.*\)/, "", $2);
    if ( (val ~ /\(calculated\)/) || (val ~ /^MAX/) ||
         ($2 ~ /^ *$/) ) {
    val = "-1";
    } else {
    val = $2;
    }
    }

    # Print the uncommented line
    printf "%57.57s    %s\n", parm, val;
}

# Exception for NEWLOGPATH: Set it to the current log path
/Path to log files/ && set_NEWLOGPATH{
    printf "%57.57s    %s\n", "NEWLOGPATH", $2;
}

END{ print ";" }
```

Display Statements in the dynamic SQL Cache

The sample script sqlcache.ksh displays the SQL statements and selected statistics currently in the dynamic SQL cache.

Here is the syntax:

```
sqlcache.ksh -c "comment" -d dbname [-o N] [-f] [-s N] [-sql]
             [-t] [-w] [-q] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -o:   Save output in dir N under $RESULTS
                (0=Current dir; -1=Not Saved,default)
        -f:   Save each SQL stmt to a different file in the $QUERIES dir
        -s:   Display N characters of SQL stmt (-1=all, default)
        -sql: Only display SQL statements, without statistics
        -t:   DONT display Timing stats
        -w:   DONT display Row Count stats
        -q:   Quiet (default is to display output)
        -r:   Don't get snapshot, reuse existing snapshot output file
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of snapshot output filename.
        Values for -d, -s, -f and -sql are part of Summary output filename.
        Timing and Row Count statistics won't be shown if "Not Collected".
        In most cases, best viewed in 80 or 132 column window.
```

The source is the following:

```
#!/bin/ksh
# sqlcache.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname [-o N] [-f] [-s N] [-sql]
          [-t] [-w] [-q] [-r] [-b]
Summerize the Statement statistics captured by the Dynamic SQL Snapshot.
    -c:   Comment placed on first line of Summary output file (required)
    -d:   Database name (required)
```

```
    -o:   Save output in dir N under \$RESULTS
    (0=Current dir; -1=Not Saved,default)
    -f:   Save each SQL stmt to a different file in the \$QUERIES dir
    -s:   Display N characters of SQL stmt (-1=all, default)
    -sql: Only display SQL statements, without statistics
    -t:   DONT display Timing stats
    -w:   DONT display Row Count stats
    -q:   Quiet (default is to display output)
    -r:   Don't get snapshot, reuse existing snapshot output file
    -b:   DONT save prior (old) results to bak directory (default=save)
    (Not saved anyway unless -o is greater than 0)
    Notes:
    Value for -d is part of snapshot output filename.
    Values for -d, -s, -f and -sql are part of Summary output filename.
    Timing and Row Count statistics won't be shown if \"Not Collected\".
    In most cases, best viewed in 80 or 132 column window.
"
}

# Constants

QUERIES=~/queries
RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`"
RES_EXT=".out"
SUM_EXT=".sum"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

QUIET=0
DISPLAY_SQL=-1
SAVE_SQL=0
SQL_ONLY=0
RESULTS_DIR=-1# -1 defaults to not saved
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

NO_TIMING=0;
NO_ROWS=0;
WINCOLS=`stty size | awk '{print $2}'`

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-d") shift; DB_NAME=$1; shift;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-f") shift; SAVE_SQL=1;;
    "-s") shift; DISPLAY_SQL=$1; shift;;
    "-sql") shift; SQL_ONLY=1;;
    "-t") shift; NO_TIMING=1;;
    "-w") shift; NO_ROWS=1;;
    "-q") shift; QUIET=1;;
    "-r") shift; REUSE_OUT=1;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ $DISPLAY_SQL -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -s param"

[ $RESULTS_DIR -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"
```

```
echo "$PARAMS" | awk '/-o -1/ && /-r/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -r with -o -1"

echo "$PARAMS" | awk '/-o -1/ && /-q/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -q with -o -1"

[ $SQL_ONLY -eq 1 ] && NO_TIMING=0 && NO_TIMING=0

echo "$PARAMS" | awk '/-sql/ && (/-t/ || /-w/){exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -sql with -t or -w"

if [ "$PARSE_ERROR" != "" ]
then
    echo ""
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

if [ $SAVE_SQL -eq 1 ]
then
    # Get last query file number
    mkdir $QUERIES 2>/dev/null
    LAST_QUERY=`ls $QUERIES | sort -n | tail -1`
    LAST_QUERY=`basename $LAST_QUERY .sql`
fi

DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`

RES_EXT="_${DB_NAME}${RES_EXT}"
SUM_EXT="_${DB_NAME}_${DISPLAY_SQL}${SAVE_SQL}${SQL_ONLY}${NO_TIMING}${NO_ROWS}${SUM
_EXT}"

if [ $RESULTS_DIR -gt 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $REUSE_OUT -eq 1 -a ! -f $RES_OUT ]
then
    echo "Can't reuse $RES_OUT - Missing"
    exit
fi

if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
    mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
    [ $REUSE_OUT -eq 0 ] && \
    cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
    cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
fi

TMP_SQL="${QUERIES}/sql.$$.tmp"

# Clean up previous aborts (trap didn't work :(
rm ${QUERIES}/sql.[0-9]*.tmp 2>/dev/null

export QUERIES DISPLAY_SQL SAVE_SQL SQL_ONLY LAST_QUERY TMP_SQL
export NO_TIMING NO_ROWS WINCOLS

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

if [ $RESULTS_DIR -ge 0 ]
```

```
then
    echo "[Creating: $SUM_OUT]"
else
    Q_OUTPUT=""
    echo "[No Output Saved]"
    fi

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ----------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--              `date`"$Q_OUTPUT
eval echo "-- "$Q_OUTPUT

if [ $RESULTS_DIR -eq -1 ]
then
    echo db2 get snapshot for dynamic sql on $DB_NAME
    echo ""
    db2 get snapshot for dynamic sql on $DB_NAME | awk -f $AWKSCRIPT
else
    if [ $REUSE_OUT -eq 0 ]
    then
    eval echo db2 get snapshot for dynamic sql on $DB_NAME $Q_OUTPUT
    eval echo ""  $Q_OUTPUT
    echo "[Creating: $RES_OUT]"
    db2 get snapshot for dynamic sql on $DB_NAME > $RES_OUT
    else
    echo ""
    echo "[Reusing: $RES_OUT]"
    fi

    eval echo ""  $Q_OUTPUT
    eval awk -f $AWKSCRIPT $RES_OUT $Q_OUTPUT
fi
```

The `sqlcache.ksh` file uses the following `sqlcache.awk` file:

```
BEGIN{
    OS    = "AIX";
    QUERIES    = ENVIRON[ "QUERIES"     ];
    DISPLAY_SQL = ENVIRON[ "DISPLAY_SQL" ];
    SAVE_SQL    = ENVIRON[ "SAVE_SQL"    ];
    SQL_ONLY    = ENVIRON[ "SQL_ONLY"    ];
    NO_TIMING   = ENVIRON[ "NO_TIMING"   ];
    NO_ROWS     = ENVIRON[ "NO_ROWS"     ];
    LAST_QUERY  = ENVIRON[ "LAST_QUERY"  ];
    TMP_SQL     = ENVIRON[ "TMP_SQL"     ];
    WINCOLS     = ENVIRON[ "WINCOLS"     ];
    header    = 0;
    COLLECTED   = 1;
    STMT_NUM    = 0;
    MIL         = 1000000;

#print  QUERIES    "=QUERIES"     ;
#print  DISPLAY_SQL "=DISPLAY_SQL" ;
#print  SAVE_SQL    "=SAVE_SQL"    ;
#print  SQL_ONLY    "=SQL_ONLY"    ;
#print  NO_TIMING   "=NO_TIMING"   ;
#print  NO_ROWS     "=NO_ROWS"     ;
#print  LAST_QUERY  "=LAST_QUERY"  ;
#print  TMP_SQL     "=TMP_SQL"     ;
#print  WINCOLS     "=WINCOLS"     ;
}

/SQL1611W/{ print }

/ Number of executions/{
        N_EXECUTIONS= "";
        N_COMPILES= "";
        T_W_PREP= "";
        T_B_PREP= "";
        R_DELETED= "";
        R_INSERTED= "";
        R_READ= "";
        R_UPDATED= "";
        R_WRITTEN= "";
        S_SORTS= "";
        T_T_EXECUTION= "";
        T_T_USER= "";
```

```
        T_T_SYSTEM= "";
}
/ Number of executions/{ N_EXECUTIONS=$NF}
/ Number of compilations/{ N_COMPILES=$NF}
/ Worst preparation time \(ms\)/{ T_W_PREP=$NF}
/ Best preparation time \(ms\)/{ T_B_PREP=$NF}
/ Rows deleted/{ R_DELETED=$NF}
/ Rows inserted/{ R_INSERTED=$NF}
/ Rows read/{ R_READ=$NF}
/ Rows updated/{ R_UPDATED=$NF}
/ Rows written/{ R_WRITTEN=$NF}
/ Statement sorts/{ S_SORTS=$NF}
/ Total execution time \(sec.ms\)/{ T_T_EXECUTION=$NF}
/ Total user cpu time \(sec.ms\)/{ T_T_USER=$NF}
/ Total system cpu time \(sec.ms\)/{ T_T_SYSTEM=$NF}

/ Statement text/{  # Begin Display

    STMT_NUM=STMT_NUM + 1;
    S_TEXT=substr( $0, index( $0, "=")+2 );
    SQL_OUT="";

    if (R_READ == "Collected")
    COLLECTED=0;

    if (DISPLAY_SQL == -1)
    SQL_LEN = length( S_TEXT);
    else
    SQL_LEN = DISPLAY_SQL;

    # Save SQL Text to a file, or get name of existing (duplicate) file
    if ( SAVE_SQL )  Save_Sql();

    headline = "";
    dataline = "";

    if ( SQL_ONLY )
    {
    if (SAVE_SQL)S_Save_Info();
    }
    else
    {
    T_A_EXEC= T_T_EXECUTION / N_EXECUTIONS;
    T_A_USER= T_T_USER/ N_EXECUTIONS;
    T_A_SYSTEM= T_T_SYSTEM / N_EXECUTIONS;
    A_R_READ= R_READ/ N_EXECUTIONS;
    A_R_WRITTEN= R_WRITTEN/ N_EXECUTIONS;
    A_R_INSERTED= R_INSERTED/ N_EXECUTIONS;
    A_R_UPDATED= R_UPDATED/ N_EXECUTIONS;
    A_R_DELETED= R_DELETED/ N_EXECUTIONS;

    N_EXECUTIONS= round_MIL( N_EXECUTIONS );
    N_COMPILES= round_MIL( N_COMPILES   );
    A_R_READ= round_MIL( A_R_READ     );
    A_R_WRITTEN= round_MIL( A_R_WRITTEN  );
    A_R_INSERTED= round_MIL( A_R_INSERTED );
    A_R_UPDATED= round_MIL( A_R_UPDATED  );
    A_R_DELETED= round_MIL( A_R_DELETED  );

    S_Exec_Info();

    if (COLLECTED && ! NO_TIMING) S_Timing_Info();

#if ( SAVE_SQL ||
#     ((! SAVE_SQL) && ! (NO_TIMING || NO_ROWS || ! COLLECTED)) )
    if ( SAVE_SQL )
    S_Save_Info();

    if (COLLECTED && ! NO_ROWS) S_Rows_Info();
    }

    S_SQL_Text();

    rm_Trailing();

    if ( headline && ! header)
    {
    for (i=1; i<=length( headline); i++)
```

```
    underline = underline "-";

    print headline;
    print underline;
    header = 1;
    }

    if (dataline) print dataline;
}

###############################################################################
function round_MIL( val ) {

    if ( val > MIL ) val = int( val / MIL ) "M";
    return val;
}

###############################################################################
function S_Exec_Info() {

    headline = headline sprintf( \
    "%4.4s | %6.6s %6.6s | %10.10s | ",
    "Qnum",
    "Exec's",
    "Comp's",
    "BestPrepMS");
    dataline = dataline sprintf( \
    "|%3s | %6s %6s | %10s | ",
    STMT_NUM,
    N_EXECUTIONS,
    N_COMPILES,
    T_B_PREP);
}

###############################################################################
function S_Timing_Info() {

    headline = headline sprintf( \
    "%10.10s %10.10s %10.10s | ",
    "ExecSEC.MS",
    "UserSEC.MS",
    "SystSEC.MS");
    dataline = dataline sprintf( \
    "%10.10s %10.10s %10.10s | ",
    T_A_EXEC,
    T_A_USER,
    T_A_SYSTEM);
}

###############################################################################
function S_Rows_Info() {

    SPACERHEAD="";
    SPACER="";
    WINCOL_Spacing();

    headline = headline sprintf( \
    "%6.6s %6.6s %s%10.10s   %10.10s %10.10s | ",
    "R_Read",
    "Writtn",
    SPACERHEAD,
    "Inserted",
    "Updated",
    "Deleted");
    dataline = dataline sprintf( \
    "%6.6s %6.6s %s%10.10s   %10.10s %10.10s | ",
    A_R_READ,
    A_R_WRITTEN,
    SPACER,
    A_R_INSERTED,
    A_R_UPDATED,
    A_R_DELETED);
}

###############################################################################
function S_Save_Info() {

headline = headline sprintf( \
```

```
    "%8.8s | ",
    "SQL-File");
#if ( ( ! SQL_ONLY ) || ( SQL_ONLY && S_TEXT ) )
    dataline = dataline sprintf( \
    "%8s | ",
    SQL_OUT);
}

###############################################################################
function S_SQL_Text() {

    if ( DISPLAY_SQL )
    headline = headline sprintf( \
    "%-*s | ",
    SQL_LEN, "SQL-Text");
    if (S_TEXT && SQL_LEN)
    dataline = dataline sprintf( \
    "%-*.*s | ",
    SQL_LEN, SQL_LEN, S_TEXT);
}

###############################################################################
function WINCOL_Spacing() {

    if ( COLLECTED && ! ( NO_TIMING || NO_ROWS ) )
    {
    if (SAVE_SQL)
    {
    if (WINCOLS == 80)
    {
    headline = headline "   | ";
    dataline = dataline "   | ";
    SPACERHEAD="| ";
    SPACER="| ";
    }
    else if (WINCOLS == 132)
    {
    SPACERHEAD="|";
    SPACER="|";
    }
    }
    else # No SAVE_SQL info
    {
    if (WINCOLS == 80)
    {
    headline = headline "          |    | ";
    dataline = dataline "          |    | ";
    SPACERHEAD="| ";
    SPACER="| ";
    }
    else if (WINCOLS == 132)
    {
    headline = headline "        | ";
    dataline = dataline "        | ";
    SPACERHEAD="| ";
    SPACER="| ";
    }
    }
    }
}

###############################################################################
function rm_Trailing() {
# Get rid of trailing separator

    if ( substr( headline, (len=length( headline))-2) == " | " )
    {
    headline = substr( headline, 1, (len - 3) );
    dataline = substr( dataline, 1, (len - 3) );
    }
}

###############################################################################
function Save_Sql() {
# Save SQL Text to a file, or get name of existing (duplicate) file

    DUPFILE="";
    print S_TEXT > TMP_SQL;
```

```
    close( TMP_SQL);

# Check if a duplicate SQL file exists
    if ( OS == "AIX" )
    {
# Get filesize of TMP_SQL
    "ls -l " TMP_SQL | getline DIRLINE;
    close( "ls -l " TMP_SQL );
    DFN=split( DIRLINE, DIRARRAY);
    FILESIZE=DIRARRAY[5];

    # Loop thru sql files in reverse creation order
    # Compare only if same filesize
    while ( (DUPFILE == "") &&
    ("ls -lt " QUERIES "/[0-9]*.sql" | getline DIRLINE ) )
    {
    DFN=split( DIRLINE, DIRARRAY);
    if ((DIRARRAY[5] == FILESIZE) &&
        (TMP_SQL != DIRARRAY[DFN])  )
    {
    "diff " TMP_SQL " " DIRARRAY[DFN]   \
    " >/dev/null 2>&1 && "              \
    " basename "  DIRARRAY[DFN] | \
    getline DUPFILE;
    close(  "diff " TMP_SQL " " DIRARRAY[DFN]   \
    " >/dev/null 2>&1 && "              \
    " basename "  DIRARRAY[DFN] );
    }
    }
    close( "ls -lt " QUERIES "/[0-9]*.sql" );

    system( "rm " TMP_SQL);
    }
    else# Untested
    {
    # delete TMP_SQL
    }

    if ( DUPFILE != "" )
    {  # A duplicate SQL file was found
    SQL_OUT = DUPFILE;
    }
    else
    {  # Create SQL file
    LAST_QUERY=LAST_QUERY + 1;
    SQL_OUT=LAST_QUERY ".sql";
    print S_TEXT > QUERIES "/" SQL_OUT;
    }
}
```

Disk I/O Activity

The script `iostat.ksh` traps `iostat` output and displays only part of the information horizontally, making it easier to track changes in activity. The output of the `iostat.ksh` script is saved under the `results` directory.

Here is the syntax:

```
iostat.ksh -c "comment" [-i m] [-p] [-o N] [-s n ] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required).
        -i:   Interval for iostat (default=1).
        -p:   Only display peak values, when they change, for all disks, etc.
              Default is to display all values, as they change.
        -o:   Save output in dir N under $RESULTS (-1=current dir, default).
        -s:   Save iostat output with n in the filename (0=dont save,default).
        -r:   Don't get iostat, reuse iostat output specifies by -s.
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Stop the utility by hitting Ctrl-C.
        Values for -i, -s and -p are part of Summary output filename.
        In most cases, best viewed in window-width which is multiple of 10.
```

The source is the following:

```
#!/bin/ksh
# iostat.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" [-i m] [-p] [-o N] [-s n ] [-r] [-b]
Summerize the output of iostat.
     -c:   Comment placed on first line of Summary output file (required).
     -i:   Interval for iostat (default=1).
     -p:   Only display peak values, when they change, for all disks, etc.
                Default is to display all values, as they change.
     -o:   Save output in dir N under \$RESULTS (-1=current dir, default).
     -s:   Save iostat output with n in the filename (0=dont save,default).
     -r:   Don't get iostat, reuse iostat output specifies by -s.
     -b:   DONT save prior (old) results to bak directory (default=save)
     (Not saved anyway unless -o is greater than 0)
     Notes:
     Stop the utility by hitting Ctrl-C.
     Values for -i, -s and -p are part of Summary output filename.
     In most cases, best viewed in window-width which is multiple of 10.
     "
}

# Constants

RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`"
RES_EXT=".out"
SUM_EXT=".sum"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

RESULTS_DIR=-1# -1 defaults to current dir
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

INTERVAL=1
SAVE_NAME=0
ONLYPEAKS=0

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-i") shift; INTERVAL=$1; shift;;
    "-p") shift; ONLYPEAKS=1;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-s") shift; SAVE_NAME=$1;   shift;;
    "-r") shift; REUSE_OUT=1; ;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ $RESULTS_DIR -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

[ $INTERVAL -ge 1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -i param"

[ $SAVE_NAME -ge 0 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -s param"
```

```
[ $REUSE_OUT -eq 1 -a $SAVE_NAME -eq 0 ] && \
    PARSE_ERROR="${PARSE_ERROR} -Cant use -r with -s 0 or missing -s"

if [ "$PARSE_ERROR" != "" ]
then
    echo ""
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

RES_EXT="_${INTERVAL}${SAVE_NAME}${RES_EXT}"
SUM_EXT="_${INTERVAL}${SAVE_NAME}${ONLYPEAKS}${SUM_EXT}"

if [ $RESULTS_DIR -ge 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$

    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
    mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
    [ $REUSE_OUT -eq 0 ] && \
    cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
    cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

export ONLYPEAKS

# BEGIN .......

[ $SAVE_NAME -eq 0 ] && TEE_OUT="" || TEE_OUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

[ $SAVE_NAME -ne 0 ] && echo "[Creating: $SUM_OUT]"

eval echo "$COMMENT"$TEE_OUT
eval echo "----------------------------------------"$TEE_OUT
eval echo "Invocation: $0 $PARAMS"$TEE_OUT
eval echo "            `date`"$TEE_OUT
eval echo $TEE_OUT

if [ $REUSE_OUT -eq 1 ]
then
    [ ! -f $RES_OUT ] && echo "Can't reuse $RES_OUT - Missing" && exit

    awk -f $AWKSCRIPT $RES_OUT | tee -a $SUM_OUT
else
    echo "Hit Ctrl-C to stop..."
    trap "" 1 2 3
    if [ $SAVE_NAME -eq 0 ]
    then
    echo "iostat output and summary not saved"
    echo "Waiting for disk i/o ..."

    iostat $INTERVAL | awk -f $AWKSCRIPT
    else
    echo "[Creating: $RES_OUT]"
    echo "Waiting for disk i/o ..."

    iostat $INTERVAL | tee $RES_OUT | awk -f $AWKSCRIPT
    awk -f $AWKSCRIPT $RES_OUT >> $SUM_OUT
    fi
fi
```

The `iostat.ksh` script uses the following `iostat.awk` file:

```
# iostat_disk.awk
```

```
# Raanon Reutlinger, IBM Israel, May 2000

BEGIN{
    HEADCOUNTMAX=10;  # Heading every x lines
    ONLYPEAKS=ENVIRON[ "ONLYPEAKS"];  # All changes or all Peaks
#ALLCHANGES=ENVIRON[ "ALLCHANGES"];# All changes or all Highest
    IOWAIT="IOWAIT";  # Heading for IOWAIT
    changedflag=0;
    firstdisk="";
    lastdisk="";
    last_iowait=0;
    cnt=0;
    if (ONLYPEAKS == "1")
    ALLCHANGES=0;
    else
    ALLCHANGES=1;  # Default is ALLCHANGES=YES
}

/^Disks:/ || /^$/{ next }

/^tty:/{

    ++cnt;
    get_iowait=1;
    next;
}

get_iowait{

    get_iowait=0;
    iowait=$NF;
    next;
}

(cnt == 1){ lastdisk=$1 }

# Skip the first two entries of every disk - ramp up of values
(cnt > 2){

    if ( (iowait > 0) && ( ALLCHANGES || (iowait > last_iowait) ) )
    {
    changed[IOWAIT]=cnt;
    reads[IOWAIT]=iowait;
    writes[IOWAIT]=iowait;
    last_iowait=iowait;
    changedflag=1;
    }
    if ( ($5 > 0) && ( ALLCHANGES || ($5 > reads[$1]) ) )
    {
    changed[$1]=cnt;
    reads[$1]=$5;
    changedflag=1;
    }
    if ( ($6 > 0) && ( ALLCHANGES || ($6 > writes[$1]) ) )
    {
    changed[$1]=cnt;
    writes[$1]=$6;
    changedflag=1;
    }
}

($1 == lastdisk) && (changedflag){  display_results( "-" ) }

END{
    if (! ALLCHANGES)
    {
    headcount = HEADCOUNTMAX - 1;
    display_results( "=" );
    }
}

############################
function display_results( underline_char )
{
    # Build heading, then replace spaces with dashes
    headcount+=1;
    headline1="";
    for (disk in changed)
```

```
    if (changed[disk] > 0)
    headline1 = headline1 sprintf( " %-9.9s", disk);
    gsub( / /, underline_char, headline1);
    if ( (headline1 != headline) || (headcount == HEADCOUNTMAX) )
    {
    headline = headline1;
    print headline;
    headcount=0;
    }

    display( "R", reads);
    display( "W", writes);
    changedflag=0;
}

############################
function display( RW, activity )
{
    got_activity=0;
    line="";
    for (disk in changed)
    {
    if (changed[disk] > 0)
    {
        if ( (activity[disk]) && (changed[disk] || ! ALLCHANGES))
        {
    line = line sprintf( "%s%9s", RW, activity[disk]);
    if (disk != IOWAIT)
    got_activity=1;
        }
        else
    line = line sprintf( "%s%9s", RW, "");
        RW="|";
    }
    if (ALLCHANGES)
    activity[disk]="";
    }
    if (got_activity)
    print line;
}
```

Lock Information

The `locks.ksh` script summarizes the current locks information being captured by the locks snapshot monitor. All of the information displayed is collected even without turning the `LOCKS` monitor switch `ON`. The number of locks are displayed at the database, application and table levels.

Here is the syntax:

```
locks.ksh -c "comment" -d dbname [-o N] [-a] [-db] [-q] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -o:   Save output in dir N under $RESULTS
                (0=Current dir; -1=Not Saved,default)
        -a:   DONT display Application level information
        -db:  DONT display Database level information
        -q:   Quiet (default is to display output)
        -r:   Don't get snapshot, reuse existing snapshot output file
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of snapshot output filename.
        Values for -d, -a and -db are part of Summary output filename.
```

The following is the source:

```
#!/bin/ksh
# locks.ksh
# Raanon Reutlinger, IBM Israel, May 2000
```

```
display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname [-o N] [-a] [-db] [-q] [-r] [-b]
Summerize the LOCKS information being captures by a LOCKS snapshot.
    -c:   Comment placed on first line of Summary output file (required)
    -d:   Database name (required)
    -o:   Save output in dir N under \$RESULTS
    (0=Current dir; -1=Not Saved,default)
    -a:   DONT display Application level information
    -db:  DONT display Database level information
    -q:   Quiet (default is to display output)
    -r:   Don't get snapshot, reuse existing snapshot output file
    -b:   DONT save prior (old) results to bak directory (default=save)
    (Not saved anyway unless -o is greater than 0)
    Notes:
    Value for -d is part of snapshot output filename.
    Values for -d, -a and -db are part of Summary output filename.
"
}

# Constants

RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`"
RES_EXT=".out"
SUM_EXT=".sum"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

QUIET=0
RESULTS_DIR=-1# -1 defaults to not saved
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

NO_APPS=0;
NO_DB=0;

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-d") shift; DB_NAME=$1; shift;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-a") shift; NO_APPS=1;;
    "-db") shift; NO_DB=1;;
    "-q") shift; QUIET=1;;
    "-r") shift; REUSE_OUT=1;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ $RESULTS_DIR -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

echo "$PARAMS" | awk '/-o -1/ && /-r/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -r with -o -1"

echo "$PARAMS" | awk '/-o -1/ && /-q/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -q with -o -1"

if [ "$PARSE_ERROR" != "" ]
then
```

```
    echo ""
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`

RES_EXT="_${DB_NAME}${RES_EXT}"
SUM_EXT="_${DB_NAME}_${NO_APPS}${NO_DB}${SUM_EXT}"

if [ $RESULTS_DIR -gt 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $REUSE_OUT -eq 1 -a ! -f $RES_OUT ]
then
    echo "Can't reuse $RES_OUT - Missing"
    exit
fi

if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
    mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
    [ $REUSE_OUT -eq 0 ] && \
    cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
    cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
fi

export NO_APPS NO_DB

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

if [ $RESULTS_DIR -ge 0 ]
then
    echo "[Creating: $SUM_OUT]"
else
    Q_OUTPUT=""
    echo "[No Output Saved]"
fi

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ----------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--             `date`"$Q_OUTPUT
eval echo "-- "$Q_OUTPUT

if [ $RESULTS_DIR -eq -1 ]
then
    echo db2 get snapshot for locks on $DB_NAME
    echo ""
    db2 get snapshot for locks on $DB_NAME | awk -f $AWKSCRIPT
else
    if [ $REUSE_OUT -eq 0 ]
    then
    eval echo db2 get snapshot for locks on $DB_NAME $Q_OUTPUT
    echo ""
    echo "[Creating: $RES_OUT]"
    db2 get snapshot for locks on $DB_NAME > $RES_OUT
    else
    echo ""
    echo "[Reusing: $RES_OUT]"
```

```
    fi

    eval echo ""$Q_OUTPUT
    eval awk -f $AWKSCRIPT $RES_OUT $Q_OUTPUT
fi
```

The `locks.ksh` script uses the following `locks.awk` file:

```
BEGIN{
    NO_APPS  = ENVIRON[ "NO_APPS"   ];
    NO_DB    = ENVIRON[ "NO_DB"     ];
    headline = "";
    dashes   = "------------------------------";
    print "";
}

/SQL1611W/{ print }

# Snapshot info at database level
!NO_DB && /Database name /{ print }
!NO_DB && /Database path /{ print }
!NO_DB && /Input database alias /{ print }
!NO_DB && /Locks held / && ! A_HAND{ print }
!NO_DB && /Applications currently connected /{ print }
!NO_DB && /Agents currently waiting on locks /{ print }
!NO_DB && /Snapshot timestamp /{ print ; print "" }

# Application info
/Application handle /{ A_HAND = ""  ;
      A_NAME= ""  ;
      A_USER= ""  ;
      H_HAND= ""  ;
      A_ID= ""  ;
      A_STATUS= ""  ;
      A_LOCKS= ""  ;
      A_WAIT= ""  ;
}
/Application handle /{ A_HAND = $NF }
/Application ID /{ A_ID = $NF ; sub( /\.[0-9]*$/, "", A_ID) }
/Application name /{ A_NAME = $NF }
/Authorization ID /{ A_USER = $NF }
/Application status /{ A_STATUS = substr( $0, index( $0, "=") + 2 )  }
/Status change time /{ A_TIME = $NF }
/Locks held / && A_HAND{ A_LOCKS = $NF }
/Total wait time \(ms\)/{ A_WAIT = $NF }

/Total wait time \(ms\)/ && (! NO_APPS) {

    if (A_WAIT == "Collected")  A_WAIT = "";

    if (! headline )
    {
    headline = sprintf( \
    "%-10.10s %-8.8s %-6.6s %-20.20s %-15.15s %-5.5s %-7.7s",
    "APP.NAME",
    "APP.USER",
    "HANDLE",
    "APP.ID",
    "APP.STATUS",
    "LOCKS",
    "WAIT.ms");

    underline = sprintf( \
    "%-10.10s %-8.8s %-6.6s %-20.20s %-15.15s %-5.5s %-7.7s",
    dashes,
    dashes,
    dashes,
    dashes,
    dashes,
    dashes,
    dashes);

    print headline;
    print underline;
    }
    dataline = sprintf( \
    "%-10.10s %-8.8s %6.6s %-20.20s %-15.15s %5.5s %s",
```

```
    A_NAME,
    A_USER,
    A_HAND,
    A_ID,
    A_STATUS,
    A_LOCKS,
    A_WAIT);
    print dataline;
}

# Lock info
/ Object Type /{ L_TYPE = ""  ;
      L_SCHEMA = ""  ;
      L_TABLE = ""  ;
      L_MODE = ""  ;
      L_STATUS = ""  ;
      L_ESC = ""  ;
}
/ Object Type /{ L_TYPE = $NF }
/ Table Schema /{ L_SCHEMA = $NF }
/ Table Name /{ L_TABLE = $NF }
/ Mode /{ L_MODE = $NF }
/ Status /{ L_STATUS = $NF }
/ Lock Escalation /{ L_ESC = $NF ;
      LOCKS [ L_SCHEMA ,
      L_TABLE  ,
      L_TYPE   ,
      L_ESC    ,
      L_MODE   ,
      L_STATUS ] ++;
}

END{
    headline = "";
    for (ind in LOCKS)
    {
    if (! headline)
    {
    headline = sprintf( \
    "%-30.30s | %-10.10s | %-3.3s | %-4.4s | %-10.10s | %5.5s",
    "TABLE NAME",
    "TYPE",
    "ESCALATED",
    "MODE",
    "STATUS",
    "COUNT");

    underline = sprintf( \
    "%-30.30s | %-10.10s | %-3.3s | %-4.4s | %-10.10s | %5.5s",
    dashes,
    dashes,
    dashes,
    dashes,
    dashes,
    dashes);

# underline = headline;
# gsub( /./, "-", underline);

    print "" ;
    print headline;
    print underline;
    }

    split( ind, LOCK_INFO, SUBSEP);
    SCHEMA_TABLE = LOCK_INFO[1] "." LOCK_INFO[2];
    dataline = sprintf( \
    "%-30.30s | %-10.10s | %-3.3s | %-4.4s | %-10.10s | %5.5s",
    SCHEMA_TABLE,
    LOCK_INFO[3],
    LOCK_INFO[4],
    LOCK_INFO[5],
    LOCK_INFO[6],
    LOCKS[ind]);
    print dataline;
    }

    if (! headline)
```

```
    {
    print "";
    print "*** NO LOCKS ***";
    }
}
```

Statement Event Monitor

The `mon_stmt.ksh` script returns information about the SQL statements (For example, `OPEN CURSOR`, `FETCH`) reaching the DB2 Engine as collected by the statements Event Monitor. This sample script issues the `db2evmon` command which reads the event records and generates a report. Before using the script, the Event Monitor has to be created and activated.

Here is the syntax:

```
mon_stmt.ksh -c "comment" -d dbname -m mon_name [-o N] [-f]
             [-s N] [-sql] [-i Q] [-p] [-ps] [-q] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -m:   Monitor name (required)
        -o:   Save output in dir N under $RESULTS
                (0=Current dir; -1=Not Saved,default)
        -f:   Save each SQL stmt to a different file in the $QUERIES dir
        -s:   Display N characters of SQL stmt (-1=all, default)
        -sql: Only display SQL statements, without statistics
        -i:   Include info in output, where Q could contain any of the following
               S=Start-Time ; O=Operation ; T=Timing ; R=Row-Counts; X=Sort-Info
               N=None_of_the_above (default=SORTX)
        -p:   mon_name is a PIPE; Don't save the summary to an output file.
        -ps:  mon_name is a PIPE; Save the summary to an output file.
               Summary output is only visible when the event monitor closes.
        -q:   Quiet (default is to display output)
        -r:   Don't read the monitor files, reuse existing output (extraction)
        -b:   DONT save prior (old) results to bak directory (default=save)
                   (0=Current dir; -1=Not Saved,default)
        -f:   Save each SQL stmt to a different file in the $QUERIES dir
        -s:   Display N characters of SQL stmt (-1=all, default)
        -sql: Only display SQL statements, without statistics
        -i:   Include info in output, where Q could contain any of the following
               S=Start-Time ; O=Operation ; T=Timing ; R=Row-Counts; X=Sort-Info
               N=None_of_the_above (default=SORTX)
        -p:   mon_name is a PIPE; Don't save the summary to an output file.
        -ps:  mon_name is a PIPE; Save the summary to an output file.
               Summary output is only visible when the event monitor closes.
        -q:   Quiet (default is to display output)
        -r:   Don't read the monitor files, reuse existing output (extraction)
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of monitor output filename.
        Values for -s, -f, -sql and -i are part of Summary output filename.
        Pipes: Make sure to run this utility BEFORE starting the event monitor
               (prompts given).  For -ps only flushed summary output is
               visible in real-time (instructions given). So for real-time
               viewing, use -p, which doesn't save summary to a file.
        In most cases, best viewed in 132 column window.
```

The following is the source:

```
#!/bin/ksh
# mon_stmt.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname -m mon_name [-o N] [-f]
          [-s N] [-sql] [-i Q] [-p] [-ps] [-q] [-r] [-b]
Summerize STATEMENT statistics captured from an Event Monitor.
```

```
    -c:   Comment placed on first line of Summary output file (required)
    -d:   Database name (required)
    -m:   Monitor name (required)
    -o:   Save output in dir N under \$RESULTS
    (0=Current dir; -1=Not Saved,default)
    -f:   Save each SQL stmt to a different file in the \$QUERIES dir
    -s:   Display N characters of SQL stmt (-1=all, default)
    -sql: Only display SQL statements, without statistics
    -i:   Include info in output, where Q could contain any of the following
           S=Start-Time ; O=Operation ; T=Timing ; R=Row-Counts; X=Sort-Info
           N=None_of_the_above (default=SORTX)
    -p:   mon_name is a PIPE; Don't save the summary to an output file.
    -ps:  mon_name is a PIPE; Save the summary to an output file.
           Summary output is only visible when the event monitor closes.
    -q:   Quiet (default is to display output)
    -r:   Don't read the monitor files, reuse existing output (extraction)
    -b:   DONT save prior (old) results to bak directory (default=save)
    (Not saved anyway unless -o is greater than 0)
    Notes:
    Value for -d is part of monitor output filename.
    Values for -s, -f, -sql and -i are part of Summary output filename.
    Pipes: Make sure to run this utility BEFORE starting the event monitor
           (prompts given).  For -ps only flushed summary output is
                   visible in real-time (instructions given). So for real-time
           viewing, use -p, which doesn't save summary to a file.
    In most cases, best viewed in 132 column window.
"
}

# Constants

QUERIES=~/queries
RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`"
RES_EXT=".out"
SUM_EXT=".sum"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

QUIET=0
DISPLAY_SQL=-1
SAVE_SQL=0
SQL_ONLY=0
RESULTS_DIR=-1# -1 defaults to not saved
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

INCLUDE_OPT="SORTX"
PIPE=0
PIPE_SAVE=0

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-d") shift; DB_NAME=$1; shift;;
    "-m") shift; MON_NAME=$1; shift;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-f") shift; SAVE_SQL=1;;
    "-s") shift; DISPLAY_SQL=$1; shift;;
    "-sql") shift; SQL_ONLY=1;;
    "-i") shift; INCLUDE_OPT=$1; shift;;
    "-p") shift; PIPE=1;;
    "-ps") shift; PIPE=1; PIPE_SAVE=1;;
    "-q") shift; QUIET=1;;
    "-r") shift; REUSE_OUT=1;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters
```

```
[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ "$MON_NAME" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Monitor Name is required"

[ $DISPLAY_SQL -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -s param"

[ $RESULTS_DIR -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

echo "$PARAMS" | awk '/-sql/ && /-i/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -sql with -i"

echo "$PARAMS" | awk '/-p[ $]/ && /-ps/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -ps with -p"

echo "$PARAMS" | awk '/-p[ $]/ && /-q/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -q with -p (only with -ps)"

echo "$PARAMS" | awk '/-o -1/ && /-r/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -r with -o -1"

echo "$PARAMS" | awk '/-o -1/ && /-q/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -q with -o -1"

[ $SQL_ONLY -eq 1 ] && INCLUDE_OPT="N"

echo "$INCLUDE_OPT" | awk '/[^SORTX]/ && ($0 != "N"){exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid value for -i param"

if [ "$PARSE_ERROR" != "" ]
then
    echo ""
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

if [ $SAVE_SQL -eq 1 ]
then
    # Get last query file number
    mkdir $QUERIES 2>/dev/null
    LAST_QUERY=`ls $QUERIES | sort -n | tail -1`
    LAST_QUERY=`basename $LAST_QUERY .sql`
fi

RESULTS_FILE=`echo $MON_NAME | tr [a-z] [A-Z]`
DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`

RES_EXT="_${DB_NAME}${RES_EXT}"
SUM_EXT="_${DB_NAME}_${DISPLAY_SQL}${SAVE_SQL}${SQL_ONLY}${INCLUDE_OPT}${SUM_EXT}"

if [ $RESULTS_DIR -gt 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $REUSE_OUT -eq 1 -a ! -f $RES_OUT ]
then
    echo "Can't reuse $RES_OUT - Missing"
    exit
fi

if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
```

```
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
    mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
    [ $REUSE_OUT -eq 0 ] && \
    cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
    cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
fi

TMP_SQL="${QUERIES}/sql.$$.tmp"

# Clean up previous aborts (trap didn't work :(
rm ${QUERIES}/sql.[0-9]*.tmp 2>/dev/null

export QUERIES DISPLAY_SQL SAVE_SQL SQL_ONLY LAST_QUERY TMP_SQL
export INCLUDE_OPT

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

if [ $RESULTS_DIR -ge 0 ]
then
    R_OUTPUT="| tee $RES_OUT"

    # with -p, Summary output not saved
    if [ $PIPE -eq 1 -a $PIPE_SAVE -eq 0 ]
    then
    Q_OUTPUT=""
    echo "[Summary Output Not Saved]"
    else
    echo "[Creating: $SUM_OUT]"
    fi
else
    R_OUTPUT=""
    Q_OUTPUT=""
    echo "[No Output Saved]"
fi

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ----------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--              `date`"$Q_OUTPUT
eval echo "-- "$Q_OUTPUT

if [ $PIPE -eq 0 -o $REUSE_OUT -eq 1 ]
then
    if [ $RESULTS_DIR -eq -1 ]
    then
    echo db2evmon $DB_NAME $MON_NAME
    echo ""
    db2evmon $DB_NAME $MON_NAME | awk -f $AWKSCRIPT
    else
    if [ $REUSE_OUT -eq 0 ]
    then
    eval echo db2evmon $DB_NAME $MON_NAME$Q_OUTPUT
    echo ""
    echo "[Creating: $RES_OUT]"
    db2evmon $DB_NAME $MON_NAME > $RES_OUT
    else
    echo ""
    echo "[Reusing: $RES_OUT]"
    fi

    echo ""
    eval awk -f $AWKSCRIPT $RES_OUT $Q_OUTPUT
    fi
else
    echo "\
------------------------------------------------------------------
Start the event monitor to begin summarizing (filtering) as follows:
  Use: db2 SET EVENT MONITOR $MON_NAME STATE 1
------------------------------------------------------------------"
#  or: . `dirname $0`/mon_state.ksh $MON_NAME 1
    if [ $PIPE_SAVE -eq 0 ]
    then
```

```
    # Gives better real-time display without pipe after awk
    echo ""
    echo db2evmon $DB_NAME $MON_NAME
    echo ""
    eval db2evmon $DB_NAME $MON_NAME $R_OUTPUT | awk -f $AWKSCRIPT
    else
    echo "\
Only when all pipe buffers are flushed will be output be visible.
To flush event monitor records manually
  Use: db2 FLUSH EVENT MONITOR $MON_NAME BUFFER
Closing the Event Monitor also flushes all records.
  Use: db2 SET EVENT MONITOR $MON_NAME STATE 0
------------------------------------------------------------------------"
#  or: . `dirname $0`/mon_state.ksh $MON_NAME 0
    eval echo ""$Q_OUTPUT
    eval echo db2evmon $DB_NAME $MON_NAME $Q_OUTPUT
    eval echo ""$Q_OUTPUT
    [ "$R_OUTPUT" != "" ] && echo "[Creating: $RES_OUT]"
    eval db2evmon $DB_NAME $MON_NAME $R_OUTPUT | \
    eval awk -f $AWKSCRIPT$Q_OUTPUT
    fi
fi
```

The mon_stmt.ksh script uses the mon_stmt.awk file as the following source:

```
BEGIN{
    OS          = "AIX";
    QUERIES    = ENVIRON[ "QUERIES"     ];
    DISPLAY_SQL = ENVIRON[ "DISPLAY_SQL" ];
    SAVE_SQL    = ENVIRON[ "SAVE_SQL"    ];
    SQL_ONLY    = ENVIRON[ "SQL_ONLY"    ];
    LAST_QUERY  = ENVIRON[ "LAST_QUERY"  ];
    TMP_SQL     = ENVIRON[ "TMP_SQL"     ];
    INCLUDE_OPT = ENVIRON[ "INCLUDE_OPT" ];

#       S=Start-Time ; O=Operation ; T=Timing ; R=Row-Counts; X=Sort-Info;N=None
    if (INCLUDE_OPT != "N")
    {
    if (INCLUDE_OPT ~ /S/) OPT_ST_TIME = 1;
    if (INCLUDE_OPT ~ /O/) OPT_OPER    = 1;
    if (INCLUDE_OPT ~ /T/) OPT_TIMES   = 1;
    if (INCLUDE_OPT ~ /R/) OPT_ROWS    = 1;
    if (INCLUDE_OPT ~ /X/) OPT_SORTS   = 1;
    }

    STMT_NUM    = 0;
    MIL         = 1000000;
    CON_REC     = 0;
    header      = 0;
    h=0;  # Counter of Handles
}

/) Connection Header Event .../{ CON_REC=1;           }
/  Appl Handle: / && CON_REC{ HANDL      ++h] = $NF ; H=$NF ;
          CA_HANDL    H ] = $NF}
/  Appl Id: /&& CON_REC{ CA_IDL      H ] = $NF}
/  Appl Seq number: /&& CON_REC{ CA_SEQL     H ] = $NF}
/  Program Name    : /{ C_PROGL     H ] = $NF}
/  Authorization Id: /{ C_DBUSERL   H ] = $NF}
/  Execution Id    : /{ C_CLUSERL   H ] = $NF}
/  Client Process Id: /{ C_CLPROCL   H ] = $NF}
/  Client Database Alias: /{ C_DBNAMEL   H ] = $NF}
/  Client Communication Protocol: /{ C_PROTOCOLL H ] = $NF}
/  Client Network Name: /{ C_NETL      H ] = $NF}
/  Connect timestamp: /{ C_TIMEL     H ] = $(NF-1) $NF }
###########################################################################
/) Statement Event .../{ CON_REC       = 0       ;
          A_HANDLE= ""  ;
                                        A_ID= ""  ;
                                        A_SEQ= ""  ;
                                        S_TYPE= ""  ;
                                        S_OPER= ""  ;
                                        P_SECTION= ""  ;
                                        P_CREATOR= ""  ;
                                        P_PACKAGE= ""  ;
                                        S_CURSOR= ""  ;
                                        S_BLOCKED= ""  ;
                                        S_TEXT = ""  ;
```

```
                                        S_TYPE = ""  ;
                                        S_START_T= ""  ;
                                        S_EXEC_T= ""  ;
                                        S_AGENTS= ""      ;
                                        S_U_CPU_T= ""      ;
                                        S_S_CPU_T= ""      ;
                                        S_FETCHES= ""      ;
                                        S_SORTS= ""      ;
                                        S_SORT_T= ""      ;
                                        S_SORT_OFLOWS = ""      ;
                                        S_ROWS_READ= ""      ;
                                        S_ROWS_WRITTEN= ""      ;
                                        S_SQLCODE= ""      ;
}
#) Statement Event ...
/  Appl Handle: /&& ! CON_REC{ A_HANDLE= $NF  }
/  Appl Id: /&& ! CON_REC{ A_ID = substr( $NF, length($NF)-4) }
/  Appl Seq number: /&& ! CON_REC{ A_SEQ= $NF  }
#  ----------------------------------------
/  Type     : /{ S_TYPE= $NF  }
/  Operation: /{ S_OPER = substr( $0, 14)}
/  Section  : /{ P_SECTION= $NF  }
/  Creator  : /{ P_CREATOR= $NF  }
/  Package  : /{ P_PACKAGE= $NF  }
/  Cursor   : /{ S_CURSOR= $NF  }
/  Cursor was blocking: /{ S_BLOCKED= $NF  }
/  Text     : /{ S_TEXT = substr( $0, 14)}
#  ----------------------------------------
/  Operation: Static Commit/{ S_TYPE = "Static";
       S_OPER= "Commit"}
/  Operation: Execute Immediate/{ S_OPER= "ExecImmed"}
#  ----------------------------------------
/  Start Time: /{ S_START_T= $NF  }
#  Stop Time:  05-16-2000 18:25:00.650866 - for datetime: $(NF-1) $NF }
/  Exec Time: /{ S_EXEC_T= $(NF-1) } #Sec
/  Number of Agents created: /{ S_AGENTS= $NF     }
/  User CPU: /{ S_U_CPU_T= $(NF-1) } #Sec
/  System CPU: /{ S_S_CPU_T= $(NF-1) } #Sec
/  Fetch Count: /{ S_FETCHES= $NF  }
/  Sorts: /{ S_SORTS= $NF  }
/  Total sort time: /{ S_SORT_T= $NF  } #Msec
/  Sort overflows: /{ S_SORT_OFLOWS = $NF  }
/  Rows read: /{ S_ROWS_READ= $NF  }
/  Rows written: /{ S_ROWS_WRITTEN= $NF  }
#  Internal rows deleted: 0
#  Internal rows updated: 0
#  Internal rows inserted: 0
#  SQLCA:
/   sqlcode: /{ S_SQLCODE= $NF  }
#   sqlstate: 00000

/   sqlstate: /{  # Begin Display

    STMT_NUM=STMT_NUM + 1;
    SQL_OUT="";

    if (DISPLAY_SQL == -1)
    SQL_LEN = length( S_TEXT);
    else
    SQL_LEN = DISPLAY_SQL;

    # Save SQL Text to a file, or get name of existing (duplicate) file
    if ( S_TEXT && SAVE_SQL )  Save_Sql();

    headline = "";
    dataline = "";

    if ( SQL_ONLY )
    {
    if (SAVE_SQL)S_Save_Info();
    }
    else
    {
    S_App_Info();
    if (SAVE_SQL)S_Save_Info();
    if (OPT_OPER) S_Oper_Info();
    if (OPT_TIMES) S_Timing_Info();
    if (OPT_ROWS)  S_Rows_Info();
```

```
    if (OPT_SORTS)
    {
    # When all options displayed, remove '|' to fit in 132
    if (INCLUDE_OPT ~ /[SORTX]{5}/)
    {
    rm_Trailing();
    headline = headline " ";
    dataline = dataline " ";
    }
    S_Sort_Info();
    }
    }

    S_SQL_Text();

    rm_Trailing();

    if ( headline && ! header)
    {
    for (i=1; i<=length( headline); i++)
    underline = underline "-";

    print headline;
    print underline;
    header = 1;
    }

    if (dataline) print dataline;
}

################################################################################
function S_App_Info() {

    headline = headline sprintf( \
    "%4.4s %5.5s %-4.4s",
    "Hand",
    "AppID",
    "Seq");
    dataline = dataline sprintf( \
    "%4s %5s %4s",
    A_HANDLE,
    A_ID,
    A_SEQ);

    if (OPT_ST_TIME)
    {
    headline = headline sprintf( \
    " %-15.15s",
    "Start-Time");
    dataline = dataline sprintf( \
    " %15s",
    S_START_T);
    }

    headline = headline " | ";
    dataline = dataline " | ";
}

################################################################################
function S_Save_Info() {

    headline = headline sprintf( \
    "%8.8s | ",
    "SQL-File");
    if ( ( ! SQL_ONLY ) || ( SQL_ONLY && S_TEXT ) )
    dataline = dataline sprintf( \
    "%8s | ",
    SQL_OUT);
}

################################################################################
function  S_Oper_Info() {

    headline = headline sprintf( \
    "%-2.2s %-9.9s %5.5s | ",
    "Type",
    "Oper",
```

```
    "Code");
    dataline = dataline sprintf( \
    "%-2.2s %-9s %5s | ",
    S_TYPE,
    S_OPER,
    S_SQLCODE );
}

################################################################################
function S_Timing_Info() {

    headline = headline sprintf( \
    "%8.8s %8.8s %8.8s | ",
    "Exec(s)",
    "UCPU(s)",
    "SCPU(s)");
    dataline = dataline sprintf( \
    "%8s %8s %8s | ",
    substr( S_EXEC_T, 1, 8),
    substr( S_U_CPU_T, 1, 8),
    substr( S_S_CPU_T, 1, 8));
}

################################################################################
function S_Rows_Info() {

    headline = headline sprintf( \
    "%9.9s %9.9s %5.5s | ",
    "Read",
    "Written",
    "Fetch");
    dataline = dataline sprintf( \
    "%9s %9s %5s | ",
    S_ROWS_READ,
    S_ROWS_WRITTEN,
    S_FETCHES);
}

################################################################################
function S_Sort_Info() {

    headline = headline sprintf( \
    "%5.5s %5.5s %9.9s | ",
    "Sorts",
    "SOVFL",
    "Sort(ms)");
    dataline = dataline sprintf( \
    "%5s %5s %9s | ",
    S_SORTS,
    S_SORT_OFLOWS,
 S_SORT_T);
}

################################################################################
function S_SQL_Text() {

    if ( DISPLAY_SQL )
    headline = headline sprintf( \
    "%-*s | ",
    SQL_LEN, "SQL-Text");
    if (S_TEXT && SQL_LEN)
    dataline = dataline sprintf( \
    "%-*.*s | ",
    SQL_LEN, SQL_LEN, S_TEXT);

    # exception to rm_Trailing, force removal of seperator when no SQL text
    # since this is always the last entry on the line
    if (S_TEXT == "")
    dataline = substr( dataline, 1, (len - 3) );
}

################################################################################
function rm_Trailing() {
# Get rid of trailing separator

    if ( substr( headline, (len=length( headline))-2) == " | " )
    {
    headline = substr( headline, 1, (len - 3) );
```

```
    dataline = substr( dataline, 1, (len - 3) );
    }
}

################################################################################
function Save_Sql() {
# Save SQL Text to a file, or get name of existing (duplicate) file

    DUPFILE="";
    print S_TEXT > TMP_SQL;
    close( TMP_SQL);

    # Check if a duplicate SQL file exists
    if ( OS == "AIX" )
    {
    # Get filesize of TMP_SQL
    "ls -l " TMP_SQL | getline DIRLINE;
    close( "ls -l " TMP_SQL );
    DFN=split( DIRLINE, DIRARRAY);
    FILESIZE=DIRARRAY[5];

    # Loop thru sql files in reverse creation order
    # Compare only if same filesize
    while ( (DUPFILE == "") &&
    ("ls -lt " QUERIES "/[0-9]*.sql" | getline DIRLINE ) )
    {
    DFN=split( DIRLINE, DIRARRAY);
    if ((DIRARRAY[5] == FILESIZE) &&
        (TMP_SQL != DIRARRAY[DFN])  )
    {
        "diff " TMP_SQL " " DIRARRAY[DFN]   \
        " >/dev/null 2>&1 && "              \
        "basename "  DIRARRAY[DFN] | \
        getline DUPFILE;
        close(  "diff " TMP_SQL " " DIRARRAY[DFN]   \
       " >/dev/null 2>&1 && "               \
       "basename "  DIRARRAY[DFN] );
    }
    }
    close( "ls -lt " QUERIES "/[0-9]*.sql" );

    system( "rm " TMP_SQL);
    }
    else# Untested
    {
    # delete TMP_SQL
    }

    if ( DUPFILE != "" )
    {  # A duplicate SQL file was found
    SQL_OUT = DUPFILE;
    }
    else
    {  # Create SQL file
    LAST_QUERY=LAST_QUERY + 1;
    SQL_OUT=LAST_QUERY ".sql";
    print S_TEXT > QUERIES "/" SQL_OUT;
    }
}
```

Benchmark Tool

The `db2batch.ksh` shell script executes `db2batch` tool and store the output to the directory structure. The syntax of this script is the following:

```
db2bench.ksh -c "comment" -d dbname [-u user/pwd] -q queryNum -o N
             [-ir rows] [-if rows] [-e m] [-v] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -u:   Userid and password seperated by slash (default=db2inst1/db2inst1)
        -q:   Query number to execute. File with .sql extension must exist in
                $QUERIES directory (required)
        -o:   Save output in dir N under $RESULTS (required)
                (0=Current dir; -1=Not Valid; no default)
        -ir:  Rows to return to output file (-1=all,default=10)
```

```
        -if:  Rows to fetch even if not returned (-1=all,default)
        -e:   Explain level: 0=ExecuteOnly; 1=ExplainOnly; 2=Explain&Execute
                (default=2)
        -v:   Verbose (default is NOT to display output)
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of output and summary filename.
```

The source is the following:

```
#!/bin/ksh
# db2bench.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
        echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname [-u user/pwd] -q queryNum -o N
              [-ir rows] [-if rows] [-e m] [-v] [-b]
Execute db2batch tool which executes the SQL found in a file and reports on
execution times and snapshot information.
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -u:   Userid and password seperated by slash (default=db2inst1/db2inst1)
        -q:   Query number to execute. File with .sql extension must exist in
                \$QUERIES directory (required)
        -o:   Save output in dir N under \$RESULTS (required)
                (0=Current dir; -1=Not Valid; no default)
        -ir:  Rows to return to output file (-1=all,default=10)
        -if:  Rows to fetch even if not returned (-1=all,default)
        -e:   Explain level: 0=ExecuteOnly; 1=ExplainOnly; 2=Explain&Execute
                (default=2)
        -v:   Verbose (default is NOT to display output)
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of output and summary filename.
"
}

# Constants

QUERIES=~/queries
RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`.awk"
RES_EXT=".out"
SUM_EXT=".sum"

# Defaults

QUIET=1
RESULTS_DIR=""          # no default
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

USERPASS="db2inst1/db2inst1"
QRYNUM=""
IROWS=10
IFETCH=-1
EXPLAIN=2

# Parse parameters

while [ "$1" != "" ]
do
        case "$1" in
        "-c"    ) shift; COMMENT=$1;     shift  ;;
        "-d"    ) shift; DB_NAME=$1;     shift  ;;
        "-u"    ) shift; USERPASS=$1;    shift  ;;
        "-q"    ) shift; QRYNUM=$1;      shift  ;;
        "-ir"   ) shift; IROWS=$1;       shift  ;;
        "-if"   ) shift; IFETCH=$1;      shift  ;;
        "-e"    ) shift; EXPLAIN=$1;     shift  ;;
        "-o"    ) shift; RESULTS_DIR=$1; shift  ;;
        "-v"    ) shift; QUIET=0                ;;
        "-b"    ) shift; SAVE_OLD_RESULTS=0     ;;
```

```
        *       ) shift; PARSE_ERROR="Invalid Param"    ;;
        esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
        PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
        PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ "$QRYNUM" = "" ] && \
        PARSE_ERROR="${PARSE_ERROR} -Query number is required"  || \
[ $QRYNUM -ge 0 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -q param"

[ "$RESULTS_DIR" = "" ] && \
        PARSE_ERROR="${PARSE_ERROR} -Output directory number is required" || \
[ $RESULTS_DIR -ge 0 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

echo "$USERPASS" | awk -F/ '($1 == "") || ($2 == ""){exit -1}' || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid user/pass following -u"
[ $IROWS -ge -1 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -ir param"

[ $IFETCH -ge -1 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -if param"

[ $EXPLAIN -ge 0 -a $EXPLAIN -le 2 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -e param"

if [ "$PARSE_ERROR" != "" ]
then
        echo ""
        echo $PARSE_ERROR
        echo ""
        display_syntax
        exit
fi

DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`
RESULTS_FILE=$QRYNUM

RES_EXT="_${DB_NAME}${RES_EXT}"
SUM_EXT="_${DB_NAME}${SUM_EXT}"

if [ $RESULTS_DIR -gt 0 ]
then
        RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
        SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
        RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
        SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
        RES_OUT=${RESULTS_FILE}${RES_EXT}
        SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $RESULTS_DIR -gt 0 ]
then
        mkdir $RESULTS                  2>/dev/null
        mkdir $RESULTS/$RESULTS_DIR     2>/dev/null
        if [ $SAVE_OLD_RESULTS -eq 1 ]
        then
                mkdir $RESULTS/$RESULTS_DIR/bak 2>/dev/null
                cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
                cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
                [ -f $RES_BAK ] && \
                        echo "[Modifying: $RES_BAK]" && \
                        `dirname $0`/db2bench_prefix.ksh $RES_BAK

        fi
fi

[ $QUIET -eq 1 ] && VERBOSE="off" || VERBOSE="on"

# BEGIN .......
```

```
[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

echo "[Creating: $SUM_OUT]"
echo "[Creating: $RES_OUT]"
echo ""

eval echo "-- $COMMENT"                                    $Q_OUTPUT
eval echo "-- ----------------------------------------" $Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"                     $Q_OUTPUT
eval echo "--             `date`"                         $Q_OUTPUT
eval echo "-- "                                           $Q_OUTPUT

echo "-- $COMMENT"                                    >  $RES_OUT
echo "-- ----------------------------------------"      >> $RES_OUT
echo "-- Invocation: $0 $PARAMS"                      >> $RES_OUT
echo "--             `date`"                          >> $RES_OUT
echo "-- "                                            >> $RES_OUT

eval echo db2batch \
        -d $DB_NAME \
        -f $QUERIES/${QRYNUM}.sql \
        -r ${RES_OUT},${SUM_OUT} \
        -a db2inst1/db2inst1 \
        -c off \
        -i complete \
        -o r $IROWS  f $IFETCH  p 5  e $EXPLAIN \
        -v $VERBOSE                                     $Q_OUTPUT
eval echo ""                                            $Q_OUTPUT

echo db2batch \
        -d $DB_NAME \
        -f $QUERIES/${QRYNUM}.sql \
        -r ${RES_OUT},${SUM_OUT} \
        -a db2inst1/db2inst1 \
        -c off \
        -i complete \
        -o r $IROWS  f $IFETCH  p 5  e $EXPLAIN \
        -v $VERBOSE                                     >> $RES_OUT
echo ""                                                 >> $RES_OUT

db2batch \
        -d $DB_NAME \
        -f $QUERIES/${QRYNUM}.sql \
        -r ${RES_OUT}.tmp,${SUM_OUT}.tmp \
        -a db2inst1/db2inst1 \
        -c off \
        -i complete \
        -o r $IROWS  f $IFETCH  p 5  e $EXPLAIN \
        -v $VERBOSE

[ ! -f ${RES_OUT}.tmp ] && echo "Error! File not created!
        RES_OUT=${RES_OUT}.tmp" &&
        exit

cat ${RES_OUT}.tmp                                      >> $RES_OUT && \
rm  ${RES_OUT}.tmp

cat ${SUM_OUT}.tmp                                      >> $SUM_OUT && \
rm  ${SUM_OUT}.tmp

# Prefix selected lines with filename
echo ""
echo "[Modifying: $RES_OUT]"
`dirname $0`/db2bench_prefix.ksh $RES_OUT
#db2batch -d tpc -f queries/2.sql -a db2inst1/db2inst1 -c off -i complete -o r 2
 f -1 p 5 e 0 -v off | awk '/Summary of Results/{doprint=1}doprint||/seconds *$/
{print}'
```

The db2bench.ksh script calls the following `db2bench_prefix.ksh` file:

```
#!/bin/ksh
# db2bench_prefix.ksh
# Raanon Reutlinger, IBM Israel

[ "$1" = "" ] && echo "\
```

```
SYNTAX: `basename $0` {db2batch-Output-Filename} [Override-Prefix]
Prefixes the filename on selected lines from the db2batch-Output file.

" && exit

SOURCE1=$1
SOURCE=`basename $0`.$$.source
TARGET=`basename $0`.$$.tmp

PREFIX=${2:-`basename $SOURCE1`}
export PREFIX

[ ! -f $SOURCE1 ] && echo "$SOURCE1 doesn't exist" && exit

# To fix problem of missing new-line at end of $SOURCE1 files
awk '{print}' $SOURCE1 > $SOURCE

awk -f `dirname $0`/db2bench_prefix.awk $SOURCE > $TARGET

[ ! -f $SOURCE -o ! -f $TARGET ] && echo "Error! File not created!
        SOURCE=$SOURCE
        TARGET=$TARGET" &&
        exit

LINES_ORIG=`wc -l $SOURCE | awk '{print $1}'`
LINES_TMP=` wc -l $TARGET | awk '{print $1}'`

# Only copy over SOURCE1 if number of lines match exactly
[ $LINES_ORIG -eq $LINES_TMP ]  && mv $TARGET $SOURCE1 \
                                || echo "Error: file not modified"

# As long as SOURCE1 is still around delete TARGET and SOURCE(tmp) if mv failed
[ -f $SOURCE1 ] && rm $TARGET $SOURCE 2>/dev/null
```

The `db2bench.ksh` script calls the following `db2bench_prefix.awk` file. You need to have it in the same directory.

```
BEGIN{
        TOP        = 1;
        PREFIX     = ENVIRON[ "PREFIX" ];
}
{
        PUT_PREFIX = 0 ;
}
#/Statement number: /                                      { TOP=0 }
#TOP                                                       { PUT_PREFIX=1 }
(NR <= TOP)                                                { PUT_PREFIX=1 }
/            \*\*\* Database Snapshot \*\*\*/              { PUT_PREFIX=1 }
/            \*\*\* Bufferpool Snapshot \*\*\*/            { PUT_PREFIX=1 }
/Bufferpool Name  .* = /                                   { PUT_PREFIX=1 }
/            \*\*\* Bufferpool Info Snapshot \*\*\*/       { PUT_PREFIX=1 }
/            \*\*\* Application Info Snapshot \*\*\*/      { PUT_PREFIX=1 }
/            \*\*\* Application Snapshot \*\*\*/           { PUT_PREFIX=1 }
/            \*\*\* Statement Details  \*\*\*/             { PUT_PREFIX=1 }
/Statement operation  .* = /                               { PUT_PREFIX=1 }
/        \*\*\* List of Associated Agents \*\*\*/          { PUT_PREFIX=1 }
/            \*\*\* Tablespace Header Snapshot \*\*\*/     { PUT_PREFIX=1 }
/            \*\*\* Tablespace Snapshot \*\*\*/            { PUT_PREFIX=1 }
/Tablespace Name .* = /                                    { PUT_PREFIX=1 }
/            \*\*\* Table Header Snapshot \*\*\*/          { PUT_PREFIX=1 }
/            \*\*\* Table Snapshot \*\*\*/                 { PUT_PREFIX=1 }
/Table Name .* = /                                         { PUT_PREFIX=1;TAB=$NF}
/Table File ID .* = / && (TAB == "=")                      { PUT_PREFIX=1 }
/            \*\*\* Database manager Snapshot \*\*\*/      { PUT_PREFIX=1 }
/            \*\*\* FCM Snapshot \*\*\*/                   { PUT_PREFIX=1 }

{               sub( /^\[.*\]: /, "")   }       # Strip existing prefix
PUT_PREFIX{     printf "[%-15.15s]: ", PREFIX } # Prefix
{               print                   }
```