```
-- a. Simple UPDATE statement
-- b. Declare variables
-- 1. Get info on sps, tables etc
-- 2. Create a list of numbers, padded with zeros.
-- 3. CREATE AN INSERT SCRIPT FROM ROWS IN A TABLE
-- 4. PUT A LIST OF NUMBERS IN ORDER
-- 5. Create a string version of a number, padded on the left with
leading zeros
-- 6. COUNT function
-- 7. Convert from literal string to smalldatetime
-- 8. Using INSERT INTO to copy data from one database to another
-- 9. Get the id of the last inserted record
-- 10. A neat little routine for cycling through a result set (using
WHILE) without using a cursor
-- 11. Good example of replacing a cursor with a set-based op
-- 12. A (redundant) example of a cursor, but it gives you the basic idea
-- 13. ISNULL() function is used to specify how we want to treat NULL
values
-- 14. Use CASE statement to specify conditional details in a select
statement
-- 15. initialise a variable at the point of declaration
-- 16. Find all tables which contain a particular column
-- 17. Find out how many rows affected by previous select, delete, insert
or update statement
-- 18. Convert date field into nicely formatted string ('3 Nov 2011')
-- 19. Get current date
-- 20. Add days to current date
-- 21. Add time to current time
-- 22. Convert int field into string
-- 23. Fetch data into local variables
-- 24. Output an error
-- 25. Add index (but check if it exists first)
-- 26. Add foreign key (but check if it exists first)
-- 27. Show indexes for a particular table
-- 28. Create a nonclustered index
-- 29. Find indexes on a particular column on a particular table
-- 30. Get rid of a foreign key
-- 31. Change the type of the main ID column  (alter a column on a table)
-- 32. Use ALTER TABLE to change col to not allow nulls
-- 33. Find primary AND foreign key constraints on a particular table
(and optionally for a particular column)
-- 34. Find foreign key constraints on a particular table
-- 35. Insert audit info in Traveller
-- 36. Order by Count([columnName])
-- 37. WITH RECOMPILE
-- 38. SELECT TOP 100 (or LIMIT)
-- 39. Get the ID of the last inserted record
-- 40. Another way of inserting multiple rows of data (see also num 44)
-- 41. Example of using FOR XML PATH:
-- 42. Recursive CTEs
-- 43. Adding a new col to a table.
-- 44. Inserting several rows at once (using row constructors). (see also
num 40)
-- 45. Going straight to a stored proc definition.
```

```
-- 46. FOR loops (there is no FOR in SQL, only WHILE) replaced with
set-based op, to numerically label rows for debug purposes
-- 47. Make a table from a comma-separated list, using maketable function
-- 48. Change varchar col to be length 255
-- 49. Drop index
-- 50. getting a list of unique file names
-- 51. Change (shorten) the length of a varchar column
-- 52. ALTER TABLE ADD COLUMN
-- 53. Useful description of the ALTER TABLE statement
-- 54. Turn a comma-separated string of IDs into a temp table of IDs (see
also num 55 below)
-- 55. Converting a comma-separated list into data, part 2: using xml
-- 56. Create a temp table.
-- 57. Creating copies (duplicates) of existing rows in the same table
-- 58. Traveller: Find what screen / tab an ASP file refers to (eg "what
does xFld.asp do?")
-- 59. Use subqueries / derived tables to find gaps in a sequence /
continuous stream of consecutive data.
--          (A similar technique can also be used to find cumulative
aggregates / sums)
-- 60. Search for / find particular word / text in all sprocs / stored
procs
-- 61. Checking that a newly inserted range of numbers doesn't overlap
with an existing range of numbers.
-- 62. A THG example using transactions + rollback, etc
-- 63. Convert a list of results into a comma-separated list
-- 64. COUNT DISTINCT: Count how many unique non-NULL values there are
-- 65. DELETE FROM and DELETE FROM ... FROM
-- 66. copy data from one table to another
-- 67. Using the UNION operator in the SELECT INTO statement.
-- 68. Fetching XML from a file using OPENROWSET
-- 69. GetDate in foreign format (ie using a different language or
locale)
-- 70. Display date in this format: '13-APR-12'
-- 71. Use CASE statement to determine what value to give a variable
-- 72. UPDATE ... FROM
-- 73. Extract Year as an integer from a Date object
-- 74. Select from a range of values (you don't use IN for this, you use
BETWEEN)
-- 75. WITH (NOLOCK)
-- 76. Improving sproc performance
-- 77. Extended Events (see ExtendedEvents.sql)
-- 78. Searching on / entering full range of characters
-- 79. Add a NOT NULL column to a table (this also show how to add / drop
a default on a column)
-- 80. Adding Traveller lookup type + lookup entries using a release
script
-- 81. Auditing updates: An example which simultaneously updates a table
and stores the previous values in local vars, for comparison to see
what's changed
-- 82. Use conditional WHERE clauses, and only search on col if its value
not NULL
-- 83. Using CASE with ORDER BY (ie being very specific about how things
are ordered)
```

```
-- 84. Call a sproc for every row in a table
-- 85. Call one sproc from another
-- 86. Add a computed column
-- 87. Using the FAST query hint to quickly retrieve the first 200 rows
-- 88. To determine whether a function / column is deterministic or
precise...
-- 89. Creating a partitioned view.
-- 90. TRUNCATE TABLE is quicker than DELETE FROM without a WHERE clause.
-- 91. Use APPLY to create a new column on the fly by applying some logic
to existing columns
-- 92. Sharing data between stored procedures (INSERT .. SELECT .. tables
returned by sprocs)
-- 93. SELECT NULL if there is no data there (if not exists), several
times in a row - works with UNION ALL
-- 94. Don't join if the joining column has NULL in it
-- 95. What does COALESCE do?
-- 96. Using COALESCE and LEFT JOIN to return a French-language row if
one exists, English if not.

------------------------------------------------------------------------
----------------------------
-- a. simple UPDATe statement
------------------------------------------------------------------------
----------------------------
--UPDATE table_name
--SET column1=value1,column2=value2,...
--WHERE some_column=some_value;
UPDATE tbl_ClientCommunicationMessages
SET ThirdPartyKey = 'AOGer_OrderConf'
WHERE
      CommunicationMessageId = 2
      AND ClientID = 44

------------------------------------------------------------------------
----------------------------
-- b. Declare variables
------------------------------------------------------------------------
----------------------------
DECLARE @MyCounter int;

------------------------------------------------------------------------
----------------------------
-- 1. Get info on sps, tables etc:
------------------------------------------------------------------------
----------------------------
Highlight the name of the thing you want info on, and click Alt F1 - it
will run sp_help
Also you can double-click on the name of a stored proc and do Alt F1 and
it will give you input and output parameters

------------------------------------------------------------------------
----------------------------
-- 2. Create a list of numbers, padded with zeros.
```

```
-------------------------------------------------------------------------
----------------------------
SELECT 'See NumListCreator.sql'

-------------------------------------------------------------------------
----------------------------
-- 3. CREATE AN INSERT SCRIPT FROM ROWS IN A TABLE
-------------------------------------------------------------------------
----------------------------
SELECT 'See TurningRowsIntoAnInsertScript.sql'

-------------------------------------------------------------------------
----------------------------
-- 4. PUT A LIST OF NUMBERS IN ORDER:
-------------------------------------------------------------------------
----------------------------
IF OBJECT_ID('tempdb..#rawdata') IS NOT NULL DROP TABLE #rawdata;
CREATE TABLE #rawdata (NumToSort int not null);
BULK INSERT #rawdata
FROM 'c:\temp\values_to_sort.txt'       --* SET UP TXT FILE PATH FIRST
WITH (
  ROWTERMINATOR = '\n'
  , TABLOCK
  );
SELECT DISTINCT NumToSort FROM #rawdata order by NumToSort asc;

-------------------------------------------------------------------------
----------------------------
-- 5. Create a string version of a number, padded on the left with
leading zeros:
-------------------------------------------------------------------------
----------------------------
-- (NB: RIGHT returns the specified number of right-most characters)
RIGHT('0000000000' + CONVERT(VARCHAR,ListPrice), 10)

-------------------------------------------------------------------------
----------------------------
-- 6. COUNT function:
-------------------------------------------------------------------------
----------------------------
SELECT COUNT(*) AS NumberOfOrders FROM Orders -- Counts how many rows in
table & calls the result "NumberOfOrders"
SELECT COUNT(DISTINCT CustomerId) AS NumberOfCustomers FROM Orders  --
counts how many distinct values of CustomerId there are.

-------------------------------------------------------------------------
----------------------------
-- 7. Convert from literal string to smalldatetime
-------------------------------------------------------------------------
----------------------------
CAST('2011-09-08 08:55:00' AS SMALLDATETIME)

-------------------------------------------------------------------------
----------------------------
```

```
-- 8. Using INSERT INTO to copy data from one database to another
-------------------------------------------------------------------------------
-----------------------------
INSERT INTO new_db.dbo.TableA
SELECT * FROM old_db.dbo.TableB

-------------------------------------------------------------------------------
-----------------------------
-- 9. Get the id of the last inserted record
-------------------------------------------------------------------------------
-----------------------------
-- (Returns the last identity value inserted into an identity column in
the same scope)
SET @l_nEmailBatchID = SCOPE_IDENTITY()

-------------------------------------------------------------------------------
-----------------------------
-- 10. A neat little routine for cycling through a result set without
using a cursor
-------------------------------------------------------------------------------
-----------------------------
-- also increments a numeric field for each record, meaning that when
displaying
-- the records you can easily see whether they're all there or not (for
debug purposes)
-- also shows how to use a WHILE loop, although see num 46 below - loops
are best avoided
--
-- for ref, here is a simple while loop:
DECLARE @nextID AS INTEGER = 1;
WHILE (@nextID <= 22)
BEGIN
      -- do stuff
      SET @nextID = @nextID + 1;
END
-- now here is the thing you came here for:
DECLARE @v_AllCount as INTEGER;
DECLARE @nextID AS INTEGER;
DECLARE @CURRENTID AS INTEGER;
DECLARE @MAXID AS INTEGER;
select @MAXID = MAX (id) from TaxiBooking_CBS where BookingID = 362;
SET @v_AllCount = 1;
SET @CURRENTID = 0;
SET @nextID = 0;
WHILE (@nextID < @MAXID)
BEGIN
      SELECT @nextID = MIN(ID) from TaxiBooking_CBS where BookingID = 362
AND ID > @CURRENTID;
      IF NOT ISNULL(@nextId,0) = 0
      BEGIN
            UPDATE TaxiBooking_CBS set Price = @v_AllCount where
BookingID = 362 and ID = @nextID;
            SET @v_AllCount = @v_AllCount + 1;
            SET @CURRENTID = @nextID;
```

```
      END
END
-- CHECK IT WORKED
SELECT * FROM TaxiBooking_CBS where BookingID = 362;


------------------------------------------------------------------------
----------------------------
-- 11. Good example of replacing a cursor with a set-based op:
------------------------------------------------------------------------
----------------------------
http://www.sql-server-performance.com/2004/operations-no-cursors/

------------------------------------------------------------------------
----------------------------
-- 12. A (redundant) example of a cursor, but it gives you the basic idea
------------------------------------------------------------------------
----------------------------
-- (it's redundant because it does the same as just doing SELECT
ProductId, Name, ListPrice FROM Production.Product)
-- Also it's always better to use set-based processes in preference to
cursors (see pp165-166 in 70-433 book for a comparison)
-- (all that is meant by set-based is that you use normal SQL to carry
out commands on all several rows, eg UPDATE Employee SET HireDate =
GetDate())
-- (see pp159-162, chap5 lesson1 in SQL 70-433 book for more info)
DECLARE
      @ProductID int
      , @ProductName varchar(50)
      , @ListPrice money
DECLARE curproducts CURSOR FOR
      SELECT ProductId, Name, ListPrice FROM Production.Product
FOR READ ONLY
OPEN curproducts
FETCH curproducts INTO @ProductID, @ProductName, @ListPrice
WHILE @@FETCH_STATUS = 0
BEGIN
      SELECT @ProductID, @ProductName, @ListPrice
      FETCH curproducts INTO @ProductID, @ProductName, @ListPrice
END
CLOSE curproducts
DEALLOCATE curproducts

------------------------------------------------------------------------
----------------------------
-- 13. ISNULL() function is used to specify how we want to treat NULL
values.
------------------------------------------------------------------------
----------------------------
-- ISNULL() here returns a zero (1st field) or the value of another col
(2nd field) if the value is NULL:
SELECT     ISNULL(P.UnitsOnOrder,0)
      , ISNULL(P.Headline,P.DefaultHeadline)
FROM Products P
```

```
-------------------------------------------------------------------------------
-----------------------------
-- 14. Use CASE statement to specify conditional details in a select
statement:
-------------------------------------------------------------------------------
-----------------------------
SELECT     P.ID,
      CASE WHEN P.Alt=1 THEN @Head2 ELSE @Head1 END+P.Headline,
      CASE ProductLine
            WHEN 'R' THEN 'Road'
            WHEN 'M' THEN 'Mountain'
            WHEN 'T' THEN 'Touring'
            WHEN 'S' THEN 'Other sale items'
            ELSE 'Not for sale'
      END,
      CASE
               WHEN ListPrice =  0 THEN 'Mfg item - not for resale'
               WHEN ListPrice < 50 THEN 'Under $50'
               WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Under $250'
               WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Under
$1000'
               ELSE 'Over $1000'
      END,
      IsNull(CASE WHEN Age.Percentage = 1 THEN I.S1/100.0*Age.S1 ELSE
Age.S1 END, I.S1)

-------------------------------------------------------------------------------
-----------------------------
-- 15. initialise a variable at the point of declaration
-------------------------------------------------------------------------------
-----------------------------
DECLARE @bTypeExists bit = 1;

-------------------------------------------------------------------------------
-----------------------------
-- 16. Find all tables which contain a particular column
-------------------------------------------------------------------------------
-----------------------------
SELECT name as TableName FROM sysobjects WHERE id IN ( SELECT id FROM
syscolumns WHERE name = 'THE_COLUMN_NAME' )

-------------------------------------------------------------------------------
-----------------------------
-- 17. Find out how many rows affected by previous select, delete, insert
or update statement
-------------------------------------------------------------------------------
-----------------------------
-- careful though, it gets set to 0 after an if statement, so it's always
best to asisgn it to a local variable
-- (More here:
http://beyondrelational.com/blogs/madhivanan/archive/2010/08/09/proper-us
age-of-rowcount.aspx)
DECLARE @MYROWCOUNT INT = 0;
```

```
DELETE FROM TaxiBooking_CBS WHERE ID = @p_nTaxiRecordID
SET @MYROWCOUNT = @@ROWCOUNT;
IF @MYROWCOUNT > 0
BEGIN
      SELECT 'Do audit stuff';
END

------------------------------------------------------------------------
----------------------------
-- 18. Convert date field into nicely formatted string ('3 Nov 2011')
------------------------------------------------------------------------
----------------------------
SELECT
      CONVERT(nvarchar(32), TB.BookingDate, 106) AS 'Booking Date'
FROM
      TaxiBooking_CBS AS TB

------------------------------------------------------------------------
----------------------------
-- 19. Get current date
------------------------------------------------------------------------
----------------------------
getdate()

------------------------------------------------------------------------
----------------------------
-- 20. Add days to current date
------------------------------------------------------------------------
----------------------------
DATEADD(DAY, 1, GETDATE())

------------------------------------------------------------------------
----------------------------
-- 21. Add time to current time
------------------------------------------------------------------------
----------------------------
convert(time, DATEADD(HOUR, 1, GETDATE()))

------------------------------------------------------------------------
----------------------------
-- 22. Convert int field into string
------------------------------------------------------------------------
----------------------------
SET @l_strAudit = CAST(@p_nTaxiRecordID AS nvarchar)

------------------------------------------------------------------------
----------------------------
-- 23. Fetch data into local variables
------------------------------------------------------------------------
----------------------------
DECLARE @l_nBookingID AS INT
DECLARE @l_strCollectionDate nvarchar
DECLARE @l_strBookingRef nvarchar
SELECT
```

```
      @l_nBookingID = TB.BookingID
      ,@l_strCollectionDate = CONVERT(nvarchar, TB.CollectionDate, 106)
      ,@l_strBookingRef = TB.BookingRefNum
FROM TaxiBooking_CBS AS TB
WHERE ID = @p_nTaxiRecordID

-------------------------------------------------------------------------------
-----------------------------
-- 24. Output an error
-------------------------------------------------------------------------------
-----------------------------
DECLARE @l_bThing AS BIT
SET @l_bThing = 1
IF @l_bThing = 1
BEGIN
      SELECT 'Do some stuff';
END
ELSE
BEGIN
      RAISERROR (N'Here is an error.',1,1);
END

-------------------------------------------------------------------------------
-----------------------------
-- 25. Add index (but check if it exists first)
-------------------------------------------------------------------------------
-----------------------------
-- !! NB!! Error!! This only checks whether there is an index that
includes the specified column on the table
-- There may be many indexes on that table that include that column
-- but that doesn't mean they contain the exact combination of columns
you are planning!
IF NOT EXISTS (SELECT DISTINCT OBJECT_NAME(si.OBJECT_ID) TableName
            FROM sys.indexes si
            INNER JOIN sys.columns sc ON si.OBJECT_ID = sc.OBJECT_ID
            WHERE si.OBJECT_ID = OBJECT_ID('TaxiBooking_CBS')
            and sc.Name = 'BookingID')
BEGIN
      CREATE NONCLUSTERED INDEX ix_TaxiBookingCBS_BookingID
            ON TaxiBooking_CBS (BookingID);
END

-------------------------------------------------------------------------------
-----------------------------
-- 26. Add foreign key (but check if it exists first)
-------------------------------------------------------------------------------
-----------------------------
IF NOT EXISTS (SELECT f.object_id
            FROM sys.foreign_keys AS f
            where OBJECT_NAME(f.parent_object_id) = 'TaxiBooking_CBS'
            AND OBJECT_NAME (f.referenced_object_id) = 'Booking')
BEGIN
      ALTER TABLE TaxiBooking_CBS
            ADD FOREIGN KEY (BookingID)
```

```
            REFERENCES Booking(ID);
END

------------------------------------------------------------------------------
-----------------------------
-- 27. Show indexes for a particular table
------------------------------------------------------------------------------
-----------------------------
EXEC sp_helpindex 'TaxiBooking_CBS'

------------------------------------------------------------------------------
-----------------------------
-- 28. Create a nonclustered index
------------------------------------------------------------------------------
-----------------------------
CREATE NONCLUSTERED INDEX ix_TaxiBookingCBS_BookingID
 ON TaxiBooking_CBS (BookingID)

------------------------------------------------------------------------------
-----------------------------
 -- 29. Find indexes on a particular column on a particular table
------------------------------------------------------------------------------
-----------------------------
 SELECT
      si.name,
      si.type_desc
      --DISTINCT OBJECT_NAME(si.OBJECT_ID) TableName
 FROM sys.indexes si
 INNER JOIN sys.columns sc ON si.OBJECT_ID = sc.OBJECT_ID
 WHERE si.OBJECT_ID = OBJECT_ID('TaxiBooking_CBS')
 and sc.Name = 'BookingID'

------------------------------------------------------------------------------
-----------------------------
 -- 30. Get rid of a foreign key
------------------------------------------------------------------------------
-----------------------------
 ALTER TABLE TaxiBooking_CBS
 DROP FOREIGN KEY fk_PerOrders

------------------------------------------------------------------------------
-----------------------------
 -- 31. Change the type of the main ID column (alter a column on a table)
------------------------------------------------------------------------------
-----------------------------
 -- ***NB you may need to use the second part of 33 below to find what
the definition of the default constraint is, before you drop it
 -- a. get rid of the primary key
 ALTER TABLE TaxiBooking_CBS
 DROP CONSTRAINT PK__TaxiBook__3214EC27497EA1FE
-- b. change the type of the column
ALTER TABLE TaxiBooking_CBS
      ALTER COLUMN ID int
 -- ***EITHER
```

```
 -- c. add the primary key back in again
 ALTER TABLE TaxiBooking_CBS
 ADD PRIMARY KEY (ID)
 -- ***OR
 -- d. add the default constraint back in again
 ALTER TABLE Supplier
            ADD CONSTRAINT DF_Supplier_Address
            DEFAULT '' FOR Address

-------------------------------------------------------------------------------
-----------------------------
 -- 32. Use ALTER TABLE to change col to not allow nulls
-------------------------------------------------------------------------------
-----------------------------
 -- First get rid of all null values, if there are any
 UPDATE TaxiBooking_CBS SET BookingID = 0 WHERE BookingID is NULL
 -- Now alter the column.
 ALTER TABLE TaxiBooking_CBS
      ALTER COLUMN BookingID int NOT NULL

-------------------------------------------------------------------------------
-----------------------------
-- 33. Find primary AND foreign key constraints on a particular table
(and optionally for a particular column)
-------------------------------------------------------------------------------
-----------------------------
-- (more on primary keys here:
http://www.w3schools.com/sql/sql_primarykey.asp)
Use THG_Test
SELECT
      OBJECT_NAME(so.OBJECT_ID) AS NameofConstraint,
      SCHEMA_NAME(so.schema_id) AS SchemaName,
      OBJECT_NAME(so.parent_object_id) AS TableName,
      so.type_desc AS ConstraintType
      ,sc.name as ColumnName -- optional
 FROM
      sys.objects so
      INNER JOIN sys.columns sc ON so.object_id = sc.default_object_id --
optional
 WHERE
      so.type_desc LIKE '%CONSTRAINT'
      and OBJECT_NAME(so.parent_object_id) = 'Supplier'   -- table name
      and sc.Name = 'Address' -- column name (optional)
 GO
 -- ... and here's a way of finding more info on (for instance) default
constraints:
 select
      c.name as ConstraintName,
      col.name as ColumnName,
      o.name as TableName
      ,*
      --,s.name as SchemaName
from
      sys.default_constraints c
```

```
      inner join sys.columns col on col.default_object_id = c.object_id
      inner join sys.objects o on o.object_id = c.parent_object_id
      --inner join sys.schemas s on s.schema_id = o.schema_id
where
      o.name = @TableName
      and col.name = @ColumnName
      --and s.name = @SchemaName

-------------------------------------------------------------------------------
-----------------------------
-- 34. Find foreign key constraints on a particular table
-------------------------------------------------------------------------------
-----------------------------
-- (more on foreign keys here:
http://www.w3schools.com/sql/sql_foreignkey.asp)
SELECT f.name AS ForeignKey,
 OBJECT_NAME(f.parent_object_id) AS TableName,
 COL_NAME(fc.parent_object_id,
 fc.parent_column_id) AS ColumnName,
 OBJECT_NAME (f.referenced_object_id) AS ReferenceTableName,
 COL_NAME(fc.referenced_object_id,
 fc.referenced_column_id) AS ReferenceColumnName
 FROM sys.foreign_keys AS f
 INNER JOIN sys.foreign_key_columns AS fc
 ON f.OBJECT_ID = fc.constraint_object_id
 where OBJECT_NAME(f.parent_object_id) = 'TaxiBooking_CBS'

-------------------------------------------------------------------------------
-----------------------------
-- 35. Insert audit info
-------------------------------------------------------------------------------
-----------------------------
IF @@ROWCOUNT > 0
BEGIN
      DECLARE @l_strAudit AS nvarchar(200)
      DECLARE @l_strTaxiOperator AS nvarchar(100)
      -- NB If this was a delete, any select statements might need to be
moved to before the delete happens
      SELECT @l_strTaxiOperator = Name
            FROM Supplier
            WHERE ID = @p_nOperatorID;
      SET @l_strAudit =N'<b>Taxi Booking added: Ref = '
                  + @p_sBookingRefNum
                  + N', Collection date = '
                  + CONVERT(nvarchar, @p_dtdCollectionDate, 106)
                  + N', Operator = '
                  + @l_strTaxiOperator
                  + N', ID = '
                  + CAST(@p_nTaxiRecordID AS nvarchar(20))
                  + N'<\b>';

      EXEC db_Audit    @UID = @p_nLoginID,
                  @Cat = @p_nCategory,
                  @ID = @p_nParentBookingID,
```

```
                  @Notes = @l_strAudit,
                  @Pri = DEFAULT
END

-------------------------------------------------------------------------------
-----------------------------
-- 36. Order by Count([columnName])
-------------------------------------------------------------------------------
-----------------------------
SELECT      ID,
      TMCount
FROM  (SELECT     Q.ID,
            COUNT(BC.ID) AS TMCount
      FROM Quote Q
            INNER JOIN Booking B ON B.QuoteID = Q.ID
            INNER JOIN Client BC ON B.ClientID = BC.ID
                        AND BC.TID = 209 -- Tour Manager
      GROUP BY   Q.ID) AS A
ORDER BY    TMCount DESC
--
-- or do it like this instead?
--
SELECT L.Name, TID, COUNT (TID)
FROM Supplier
      INNER JOIN Lookup L
      ON L.ID = TID
GROUP BY TID, L.Name
ORDER BY COUNT (TID) DESC

-------------------------------------------------------------------------------
-----------------------------
-- 37. WITH RECOMPILE
-------------------------------------------------------------------------------
-----------------------------
In SQL, WITH RECOMPILE means that the query plan is thrown away, which
means that when testing you will get genuine speed results each time
(rather than the query appearing to speed up cos it''s cached)
MORE HERE:
http://www.techrepublic.com/article/understanding-sql-servers-with-recomp
ile-option/5662581

-------------------------------------------------------------------------------
-----------------------------
-- 38. SELECT TOP 100 (or LIMIT)
-------------------------------------------------------------------------------
-----------------------------
-- The top part just goes after the select and before everything else:
SELECT
      TOP 10000 -- this avoids 'slow-running script' error when too much
data is displayed
      Min(I.ID),
      Convert(char(11),
```

```
!! In other versions of SQL - eg Postgres - you use LIMIT instead of TOP.
Like this:
SELECT
      *
FROM TABLE
LIMIT 1;

------------------------------------------------------------------------
----------------------------
-- 39. Get the ID of the last inserted record
------------------------------------------------------------------------
----------------------------
select @@IDENTITY

------------------------------------------------------------------------
----------------------------
-- 40. Another way of inserting multiple rows of data
------------------------------------------------------------------------
----------------------------
DECLARE @t TABLE (
RootNode VARCHAR(15),
ParentNode VARCHAR(10),
Node VARCHAR(5),
Name VARCHAR(5),
Number VARCHAR(1),
Valid VARCHAR(5),
Value VARCHAR(15))
INSERT INTO @t
SELECT 'Reference','Basic','Book','Book1','1','true','AH.KL.LO'
UNION ALL
SELECT 'Reference','App','A','App1','1','true','AIK.LPO'
UNION ALL
SELECT 'Reference','App','A','App2','2','true','JUI.MKJ'
UNION ALL
SELECT 'Reference','Sub','B','SubA','1','false','LOP.MJH'
UNION ALL
SELECT 'Reference','Sub','B','SubB','2','false','GTY.JUI'
UNION ALL
SELECT 'Reference','DI','C',NULL,'1',NULL,'PLW.KJU'

------------------------------------------------------------------------
----------------------------
-- 41. Example of using FOR XML PATH:
------------------------------------------------------------------------
----------------------------
http://beyondrelational.com/blogs/jacob/archive/2009/01/09/for-xml-path-y
et-another-shaping-example-using-for-xml-path.aspx
SELECT
      name AS '@Name',
      number AS '@number',
      valid AS '@valid'
FROM @t
WHERE node = 'Book'
FOR XML PATH('Book')
```

```
-- The above code generates the following XML output:
<Book Name="Book1" number="1" valid="true" />
---
---
SELECT
      (
            SELECT
                  name AS '@Name',
                  number AS '@number',
                  valid AS '@valid',
                  value AS 'data()'
            FROM @t
            WHERE node = 'Book'
            FOR XML PATH('Book'), TYPE
      ) AS Basic,
      (
            SELECT
                  name AS '@Name',
                  number AS '@number',
                  valid AS '@valid',
                  value AS 'data()'
            FROM @t
            WHERE node = 'A'
            FOR XML PATH('A'), TYPE
      ) AS App
FOR XML PATH(''), ROOT('Reference')
-- The above code generates the following XML output:
<Reference>
      <Basic>
            <Book Name="Book1" number="1" valid="true">AH.KL.LO</Book>
      </Basic>
      <App>
            <A Name="App1" number="1" valid="true">AIK.LPO</A>
            <A Name="App2" number="2" valid="true">JUI.MKJ</A>
      </App>
</Reference>

-------------------------------------------------------------------------
----------------------------
-- 42. Recursive CTEs
-------------------------------------------------------------------------
----------------------------
-- See SQL 70-433, Chapter04Lesson01.sql

-------------------------------------------------------------------------
----------------------------
-- 43. Adding a new col to a table.
-------------------------------------------------------------------------
----------------------------
alter table [Comments] add TimeSpent time null -- column is called
TimeSpent, is of type time, and is nullable

-------------------------------------------------------------------------
----------------------------
```

```
-- 44. Inserting several rows at once (using row constructors).
-------------------------------------------------------------------------------
-----------------------------
-- When you use the VALUES clause to create fresh data, this is called a
row constructor.
-- This example contains three row constructors:
-- NB Note that you don't have to repeat the VALUES keyword for each new
row.
INSERT INTO @emp (FirstName, LastName)
VALUES
      ('Jacob', 'Sebastian'),
      ('Bob', 'Jones'),
      ('Doe','Jane');

-------------------------------------------------------------------------------
-----------------------------
-- 45. Going straight to a stored proc definition.
-------------------------------------------------------------------------------
-----------------------------
-- There are three ways of doing this.
-- 1. USE THIS ONE.
-- This is the simplest to type, and effectively formats it for you by
having a separate row for each line of text
-- just select the whole column, right-click in the cell and choose Copy
EXEC sp_helptext N'THG_Test.dbo.spThgOwnerBookingCreateMultiple';
-- 2. Not much use really because the whole sp is contained in one cell
and can't be easily readably formatted
SELECT OBJECT_DEFINITION
(OBJECT_ID(N'THG_Test.dbo.spThgOwnerBookingCreateMultiple'));
-- 3. Not much use for the same reasons as number 2.
SELECT definition
FROM sys.sql_modules
WHERE object_id =
(OBJECT_ID(N'THG_Test.dbo.spThgOwnerBookingCreateMultiple'));

-------------------------------------------------------------------------------
-----------------------------
-- 46. FOR loops (there is no FOR in SQL, only WHILE) replaced with
set-based op, to numerically label rows for debug purposes
-------------------------------------------------------------------------------
-----------------------------
-- SQL doesn't have FOR loops, only WHILE
-- But mostly it pays to find a set based solution rather than write
loops. (they are quicker and more efficient)
-- A common solution is to keep a table of numbers in your database
(single column of numbers from 0 to some very large number). This can
-- help avoid loops in many places:
create table MyNumbers (MyNum int not null); -- NB I have already done
this, in THG_Test
DECLARE @count INTEGER = 1;
WHILE @count < 1000001 -- one million and one. Watch out though, takes a
few minutes.
BEGIN
      insert into MyNumbers select @count;
```

```
      set @count = @count + 1;
END
select MAX(MyNum) from MyNumbers
-- simple example:
INSERT INTO table_name (...)
 SELECT MyNum, ...
 FROM Numbers
 WHERE MyNum BETWEEN 1 AND 10


-------------------------------------------------------------------------------
-----------------------------
 -- 47. Make a table from a comma-separated list, using maketable
function
-------------------------------------------------------------------------------
-----------------------------
 -- The w_CotTop stored proc in THG is called like this:
 exec w_CotTop
NULL,NULL,3143535,NULL,NULL,NULL,N'500,564,540,574,550,590,560,570,580,52
5,530,581,582,576'
 -- ... and contains this:
 CREATE PROCEDURE [dbo].[w_CotTop];1 @ID int, @AID [Key], @ClientID int,
@Brand_Id int=NULL
 , @RID [key]=NULL, @AgentID int=NULL, @QTAccess [Notes]=NULL
 -- ...
      INNER JOIN Lookup AS L ON SHR.SupplierTypeID = L.ID
      INNER JOIN dbo.maketable(@QTAccess) AS Q ON L.Val5 = CAST(Q.Str AS
int)

-------------------------------------------------------------------------------
-----------------------------
 -- 48. Change varchar col to be length 255
-------------------------------------------------------------------------------
-----------------------------
ALTER TABLE thg_DaybooksHeaderDetail
      ALTER COLUMN Comments varchar(255) NULL

-------------------------------------------------------------------------------
-----------------------------
-- 49. Drop index
-------------------------------------------------------------------------------
-----------------------------
drop index PaymentStreamBandID
-- or you can do it via SSMS, by right-clicking on the table and
selecting Design
-- then highlighting all cols, right-clicking and selecting Indexes /
Keys
-- and deleting the index in there.

-------------------------------------------------------------------------------
-----------------------------
-- 50. getting a list of unique file names
-------------------------------------------------------------------------------
-----------------------------
```

```
-- See U:\My_Documents\_Really Mine\Issues\Scripts\fileNames.xlsx
-- copy your file names into here, extend the enclosing columns if
necessary
-- (NB you can use the little cross in the bottom right of the cell and
then just drag it
-- downwards - when you reach the bottom of the screen it will
automatically scroll) and
-- then copy the cells into notepad and then into sql to make the insert
statements
-- and run the following:
use testDB;
drop table myFileNames;
create table myFileNames (myFileName nvarchar(max) not null)
-- insert statements go here
select distinct myFileName from myFileNames order by myFileName

------------------------------------------------------------------------
----------------------------
-- 51. Change (shorten) the length of a varchar column
------------------------------------------------------------------------
----------------------------
-- Can't be done! You have to drop the table and recreate. If necessary
copy data into a temp table and truncate the varchar fields that are too
long.

------------------------------------------------------------------------
----------------------------
-- 52. ALTER TABLE ADD COLUMN -- you don't need the word COLUMN
------------------------------------------------------------------------
----------------------------
ALTER TABLE thg_DaybooksHeaderDetail
      ADD NewComments varchar(255) NULL

------------------------------------------------------------------------
----------------------------
-- 53. Useful description of the ALTER TABLE statement
------------------------------------------------------------------------
----------------------------
http://db.apache.org/derby/docs/10.2/ref/rrefsqlj81859.html

------------------------------------------------------------------------
----------------------------
-- 54. Turn a comma-separated string of IDs into a temp table of IDs (see
also num 55 below)
------------------------------------------------------------------------
----------------------------
-- useful so you can say WHERE ID IN ([list of IDs])
-- THG has its own split function: tfnThgSplitStr
Declare @IDsToDelete table (DeleteID int)
INSERT INTO @IDsToDelete
SELECT str FROM dbo.tfnThgSplitStr('72558, 72584', ',')
-- Now use it
DELETE Commission
WHERE ID IN (SELECT DeleteID FROM @IDsToDelete)
```

```
-------------------------------------------------------------------------------
------------------------------
-- 55. Converting a comma-separated list into data, part 2: using xml
-------------------------------------------------------------------------------
------------------------------
CREATE FUNCTION dbo.Split3 ( @strString varchar(4000))
RETURNS  @Result TABLE(Value BIGINT)
AS
BEGIN
DECLARE @x XML
SELECT @x = CAST('<A>'+ REPLACE(@strString,',','</A><A>')+ '</A>' AS XML)
INSERT INTO @Result
SELECT t.value('.', 'int') AS inVal
FROM @x.nodes('/A') AS x(t)
RETURN
END
GO

-------------------------------------------------------------------------------
------------------------------
-- 56. Create a temp table.
-------------------------------------------------------------------------------
------------------------------
Declare @IDsToDelete table (DeleteID int)

-------------------------------------------------------------------------------
------------------------------
-- 57. Creating copies (duplicates) of existing rows in the same table
-------------------------------------------------------------------------------
------------------------------
-- NB you have to execute all statements in one go, because you are using
a table variable.
--
-- First, create a temp table to copy the originals over (NB: use
right-click - "script table as" to get the table definition)
--    NB don't include the ID col in this definition
Declare @SourcesToCopy table (
--    [ID] [int] NOT NULL,
      [Name] [dbo].[Name] NULL,
      [Insertion] [smalldatetime] NULL)
-- now populate the temp table - again, don't include the ID col
INSERT INTO @SourcesToCopy (
      [Name],
      [Insertion])
SELECT
      [Name],
      [Insertion]
 FROM Source
-- now update the copies to change whatever you want to change
update @SourcesToCopy set IsBookingSource = 0
-- now copy them back to the original.
--    Again, don't include the ID col.
INSERT INTO  Source (
```

```
      [Name],
      [Insertion])
SELECT * FROM @SourcesToCopy

-------------------------------------------------------------------------------
-----------------------------
-- 58. Traveller: Find what screen / tab an ASP file refers to (eg "what
does xFld.asp do?")
-------------------------------------------------------------------------------
-----------------------------
SELECT *
FROM Menu
WHERE Script = 'Diary'

-------------------------------------------------------------------------------
-----------------------------
-- 59. Use subqueries / derived tables to find gaps in a sequence /
continuous stream of consecutive data.
-------------------------------------------------------------------------------
-----------------------------
-- NB A similar technique can also be used to find cumulative aggregates
/ sums. See Chapter04Lesson02.sql.
-- This version actually find the ranges of unbroken nummbers around the
gaps, but see Chapter04Lesson02.sql
-- for 2 versions that explicitly find the gaps.
SELECT
      -- The call to MAX means that we only get the one nearest to
EndRange
      MAX(StartNumbers.ChequeNum) AS StartRange
      , EndNumbers.ChequeNum AS EndRange
FROM
      -- StartNumbers is a list of all the cheque nums that don't have
another consecutive one immediately before them.
      (SELECT    Nums1.UsedChequeNo AS ChequeNum
            FROM @UsedChequeNumbersToKeep Nums1
            WHERE NOT EXISTS (SELECT * FROM @UsedChequeNumbersToKeep
Nums2
                        WHERE (Nums1.UsedChequeNo - Nums2.UsedChequeNo) =
1)) StartNumbers
      INNER JOIN
      -- EndNumbers is a list of all the cheque nums that don't have
another consecutive one immediately AFTER them.
      (SELECT    Nums1.UsedChequeNo AS ChequeNum
            FROM @UsedChequeNumbersToKeep Nums1
            WHERE NOT EXISTS (SELECT * FROM @UsedChequeNumbersToKeep
Nums2
                        WHERE (Nums2.UsedChequeNo - Nums1.UsedChequeNo) =
1)) EndNumbers
      -- This means we get a list of ALL the StartNumbers entries whose
cheque numbers are less than EndNumbers ones
      -- (but that's OK because of the MAX call above)
      ON StartNumbers.ChequeNum <= EndNumbers.ChequeNum
GROUP BY    -- group by EndNumbers.ChequeNum to make the MAX call above
meaningful
```

```
      EndNumbers.ChequeNum;


-------------------------------------------------------------------------------
-----------------------------
-- 60. Search for / find particular word / text in all sprocs / stored
procs
-------------------------------------------------------------------------------
-----------------------------
-- If you are searching for all stored procs that contain the text
"foobar":
SELECT ROUTINE_NAME, ROUTINE_DEFINITION
    FROM INFORMATION_SCHEMA.ROUTINES
    WHERE ROUTINE_DEFINITION LIKE '%foobar%'
    AND ROUTINE_TYPE='PROCEDURE'


-------------------------------------------------------------------------------
-----------------------------
-- 61. Checking that a newly inserted range of numbers doesn't overlap
with an existing range of numbers.
-------------------------------------------------------------------------------
-----------------------------
DECLARE @NewLastNum INT
DECLARE @NewFirstNum INT
DECLARE @NewRowId INT
      SELECT ID
      FROM TableContainingRanges TCR
      -- If the existing First is less than the new Last, then the
existing Last should also be less than the new First.
      -- If not, we have an overlap (and the = takes care of when any of
the values are equal).
      WHERE (TCR.FirstNum <= @NewLastNum AND TCR.LastNum >= @NewFirstNum)
            AND TCR.Id != @NewRowId


-------------------------------------------------------------------------------
-----------------------------
-- 62. A THG example using transactions + rollback, etc
-------------------------------------------------------------------------------
-----------------------------
spThg_OwnerPaymentRunProcessUpdateOwnerAccounts contains useful
transaction stuff.


-------------------------------------------------------------------------------
-----------------------------
-- 63. Convert a list of results into a comma-separated list
-------------------------------------------------------------------------------
-----------------------------
-- This is actually very clever but really quite hard to see what it's
doing:
-- The initial SELECT statement is re-executed for every result, hence it
gradually builds up into a full list.
```

```
-- The second example doesn't work, because there is no reassignment for
each row returned.
-- COALESCE just takes a series of arguments, and returns the first
non-null argument
DECLARE @listStr VARCHAR(MAX)
SELECT @listStr = COALESCE(@listStr+',' ,'') + Name
FROM Money
SELECT @listStr
-- this doesn't work:
DECLARE @listStr2 VARCHAR(MAX)
SELECT COALESCE(@listStr2+',' ,'') + Name
FROM Money
WHERE ID <= 3
```

```
-------------------------------------------------------------------------
-----------------------------
-- 64. COUNT DISTINCT: Count how many unique non-NULL values there are
-------------------------------------------------------------------------
-----------------------------
--Count how many unique non-NULL values of CustomerId there are:
SELECT COUNT(DISTINCT CustomerId) AS NumberOfCustomers FROM Orders
```

```
-------------------------------------------------------------------------
-----------------------------
-- 65. DELETE FROM and DELETE FROM ... FROM
-------------------------------------------------------------------------
-----------------------------
DELETE FROM <tablename>
--Without a WHERE clause, all rows are deleted.
--You can have a second FROM clause in the same way that you can in an
UPDATE statement:
DELETE FROM Products AS P1
FROM NewProducts AS P2
WHERE P1.id = P2.id.
-- The FROM  statement is used in a DELETE  statement to indicate a table
join.
```

```
-------------------------------------------------------------------------
-----------------------------
-- 66. copy data from one table to another
-------------------------------------------------------------------------
-----------------------------
INSERT INTO ... SELECT
--means you can
--copy data from one table to another
```

```
-------------------------------------------------------------------------------
-----------------------------
-- 67. Using the UNION operator in the SELECT INTO statement.
-------------------------------------------------------------------------------
-----------------------------
-- NB SELECT INTO creates a new table (Not to be confused with INSERT
INTO)
---------
CREATE TABLE T1 (THING INT NULL);
INSERT INTO T1 VALUES (1), (2), (3);
SELECT * FROM T1;
---------
CREATE TABLE T2 (THING INT NULL);
INSERT INTO T2 VALUES (4), (5), (6);
SELECT * FROM T2;
---------
CREATE TABLE T3 (THING INT NULL);
---------
-- Use the UNION operator in the SELECT INTO statement.
-- Data has to be selected into T4, not T3, because SELECT INTO creates a
new table, so otherwise you get an error
-- that the table already exists.
SELECT THING INTO T4 FROM T1
UNION
SELECT THING FROM T2; -- This line replaces the one below.
--SELECT THING INTO T3 FROM T2; The INTO clause must be included only
once, in the first SELECT statement.
--------
SELECT * FROM T4;


-------------------------------------------------------------------------------
-----------------------------
-- 68. Fetching XML from a file using OPENROWSET
-------------------------------------------------------------------------------
-----------------------------
-- You can fetch XML from a file using OPENROWSET:
SELECT @x = xCol.BulkColumn
FROM OPENROWSET (BULK '<File Directory>\ShippedOrders.xml', SINGLE_BLOB)
AS xCol;


-------------------------------------------------------------------------------
-----------------------------
-- 69. GetDate in foreign format (ie using a different language or
locale)
-------------------------------------------------------------------------------
-----------------------------
-- to find the various language names (although they're mostly obvious):
SELECT * FROM SYS.SYSLANGUAGES;
-- then just set your language and call GETDATE().
-- Don't forget to set language back again:
```

```
-- start in british, for comparison
SET LANGUAGE  British;
DECLARE @UK NVARCHAR(MAX);
SET @UK = GETDATE();
SELECT @UK;
-- now do french
SET LANGUAGE  French;      -- déc  1 2015 12:00AM
DECLARE @France NVARCHAR(MAX);
SET @France = GETDATE();
SELECT @France;
-- this time let's do it in the following format: '13-AVR-12'
SET @France = UPPER(REPLACE(CONVERT(VARCHAR, GETDATE(), 6), ' ', '-'));
SELECT @France;
-- set back to british
SET LANGUAGE  British;

------------------------------------------------------------------------
----------------------------
-- 70. Display date in this format: '13-APR-12'
------------------------------------------------------------------------
----------------------------
DECLARE @UK NVARCHAR(MAX);
SET @UK = UPPER(REPLACE(CONVERT(VARCHAR, GETDATE(), 6), ' ', '-'));
SELECT @UK;


------------------------------------------------------------------------
----------------------------
-- 71. Use CASE statement to determine what value to give a variable
------------------------------------------------------------------------
----------------------------
DECLARE    @Language NVARCHAR(MAX) = 'EN';
DECLARE @SystemLanguage nvarchar(max);
SET @SystemLanguage = CASE @Language
            WHEN 'FR' THEN N'French'
            WHEN 'IT' THEN N'Italian'
            ELSE N'British'
      END;

------------------------------------------------------------------------
----------------------------
-- 72. UPDATE ... FROM
------------------------------------------------------------------------
----------------------------
-- Use a FROM to specify which rows you are updating
UPDATE dbo.thg_OwnerContractSeason SET Year = 2010
FROM  dbo.thg_OwnerContractSeason I
      INNER JOIN dbo.thg_OwnerContract OC ON OC.Id = I.OwnerContract_Id
WHERE OC.Name = 'CTC-Name'
-- USe an alias for the table you are updating
UPDATE     OPRR
      SET   OPRR.ReportStorageGUID = @ReportStorageID,
            OPRR.ReportFilePath = NULL
      FROM  thg_OwnerPaymentRunReport OPRR
```

```
            INNER JOIN thg_OwnerPaymentRunReportLookup OPRRL ON
OPRR.ReportID = OPRRL.ID
      WHERE OPRRL.BreaseReportID = @BreaseReportID
      AND   OPRR.PaymentStreamBandID = @PaymentRunID
-- but I still haven't foudn a way of referring to the updated table in a
subquery. This won't work:
-- experimenting with update .. from
UPDATE      OPRR
      SET   OPRR.ReportStorageGUID = (SELECT OPRRL.ID
      FROM  thg_OwnerPaymentRunReport OPRR
            INNER JOIN thg_OwnerPaymentRunReportLookup OPRRL ON
OPRR.ReportID = OPRRL.ID
      WHERE OPRRL.BreaseReportID = @BreaseReportID
      AND   OPRR.PaymentStreamBandID = @PaymentRunID)

------------------------------------------------------------------------
------------------------------
-- 73. Extract Year as an integer from a Date object
------------------------------------------------------------------------
------------------------------
SELECT YEAR(GETDATE())

------------------------------------------------------------------------
------------------------------
-- 74. Select from a range of values (you don't use IN for this, you use
BETWEEN)
------------------------------------------------------------------------
------------------------------
-- don't do this:
UPDATE thg_OwnerPaymentRunStatement SET CurrencyID = 1 WHERE ID IN (43546
TO 43556)
-- do this:
UPDATE thg_OwnerPaymentRunStatement SET CurrencyID = 1 WHERE ID BETWEEN
43546 AND 43556


------------------------------------------------------------------------
------------------------------
-- 75. WITH (NOLOCK)
------------------------------------------------------------------------
------------------------------
-- SQL automatically locks data for reading / updating, but this can have
a performance hit.
-- Turn it off like this:
SELECT * FROM Service
                  WITH (NOLOCK)
                  where Name like '%'+@Query+'%' OR Code like
'%'+@Query+'%'
-- ... or like this:
SELECT
      b.id,
FROM
      blog_entry b
      INNER JOIN
```

```
      blog_entry_tag_jn btjn (NOLOCK)
-- More here:
http://www.bennadel.com/blog/477-SQL-Server-NOLOCK-ROWLOCK-Directives-To-
Improve-Performance.htm


-------------------------------------------------------------------------
----------------------------
-- 76. Improving sproc performance
-------------------------------------------------------------------------
----------------------------
-- See case 24108 and THG: spThgServiceListGetMinimal
-- Try these things:
-- SET NOCOUNT ON
-- WITH (NOLOCK)
-- TOP 100
-- Reduce size of input parameters (avoid varchar(MAX))
-- Don't return whole records unless necessary (avoid SELECT *)
-- Add covering indexes
-- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
-- Avoid leading wildcards (WHERE Name like '%Thing' - the leading %
prevents the use of indexes)


-------------------------------------------------------------------------
-------------------------------------------------
-- 78. Searching on / entering full range of characters
-------------------------------------------------------------------------
-------------------------------------------------
declare @AllChars varchar(50 )=N'¬`!"£$%^&*()_-+={[}]:;@''~#?/>.<,';
-- ... and here are the subsets, for use when homing in on any potential
problem characters:
-- 1¬`!"£$%^&*()_-+={[}]:;@''~#?/>.<,
--    1¬`!"£$%^&*()_-+=
--          1¬`!"£$%^
--                1¬`!"
--                      1¬`
--                      !"
--                £$%^
--                      £$
--                      %^
--          &*()_-+=
--                &*()
--                      &*
--                      ()
--                _-+=
--                      _-
--                      +=
--    {[}]:;@''~#?/>.<,
--          {[}]:;@''
--                {[}]:
--                      {[}
--                      ]:
--                ;@''
```

```
--                      ;@
--                      ''
--          ~#?/>.<,
--                ~#?/
--                      ~#
--                      ?/
--                >.<,
--                      >.
--                      <,


-------------------------------------------------------------------------------
-----------------------------
-- 77. Extended Events (see ExtendedEvents.sql)
-------------------------------------------------------------------------------
-----------------------------
-- See U:\My_Documents\_ReallyMine\Training\SQL\70-451\ExtendedEvents.sql

-------------------------------------------------------------------------------
-----------------------------
-- 79. Add a NOT NULL column to a table (this also show how to add / drop
a default on a column)
-------------------------------------------------------------------------------
-----------------------------
-- You need to make sure your new column has a default value
ALTER TABLE Service
ADD ValidateBedConfigs [Bool] NOT NULL DEFAULT 0
-- To test this works with SVNDB, drop the column and then see if SVNDB
can successfully addit.
-- To drop the column, you first need to drop the default, which means
you need to know the internal name of the default.
-- So start by running this, and you should get an error complaining
about the default, and telling you what its name is:
ALTER TABLE Service
      DROP COLUMN ValidateBedConfigs
-- ... and now you can drop the default:
ALTER TABLE Service
      DROP CONSTRAINT DF__Service__Validat__4A722CAA
-- (incidentally this is supposed to drop a default on a col, but doesn't
seem to work):
ALTER TABLE Service
      ALTER COLUMN ValidateBedConfigs
      DROP DEFAULT
-- and now you can safely drop the column itself:
ALTER TABLE Service
      DROP COLUMN ValidateBedConfigs


-------------------------------------------------------------------------------
-----------------------------
-- 80. Adding Traveller lookup type + lookup entries using a release
script
-------------------------------------------------------------------------------
-----------------------------
```

```
-- Check whether the PID is already in use
IF NOT EXISTS (SELECT Name FROM [LookupName] WHERE LookupPID = 317)
BEGIN
      -- Insert the parent record in LookupName, which gives the lookup
type a name and a PID
      INSERT INTO LookupName (LookupPID, Name)
      VALUES (317, 'Document Msg Type');

      -- Sometimes child lookup records have prefixes that aren't
actually taken by a PID, so need to check for this.
      -- Can't be bothered checking every one, just check the first.
      IF NOT EXISTS (SELECT * FROM [Lookup] WHERE ID = 31701)
      BEGIN
            INSERT INTO Lookup (ID, PID, Name, Notes, Act)
            VALUES (31701, 317, 'Acknowledge', 'Message must be
acknowledged', 1);
      END
      ELSE
      BEGIN
            -- If the lookup row is there and in use for something else,
alert the user (this will have to be fixed manually)
            -- Otherwise it's fine - just means this script has already
been run.
            IF EXISTS (SELECT * FROM [Lookup] WHERE ID = 31701 AND Name
!= 'Acknowledge')
            BEGIN
                  PRINT 'Lookup conflict! You already have a lookup with
ID 31701. See WebApps release team.'
            END
      END

      -- Now insert the rest of the lookup entries, but don't bother
checking for duplication this time.
      IF NOT EXISTS (SELECT * FROM [Lookup] WHERE ID = 31702)
      BEGIN
            INSERT INTO Lookup (ID, PID, Name, Notes, Act)
            VALUES (31702, 317, 'Information', 'Information only', 1);
      END
END
ELSE
BEGIN
      -- If the PID is there and in use for something else, alert the
user (this will have to be fixed manually).
      -- Otherwise it's fine - just means this script has already been
run.
      IF EXISTS (SELECT * FROM [LookupName] WHERE LookupPID = 317 AND
Name != 'Document Message Type')
      BEGIN
            PRINT 'Lookup conflict! You already have lookups with PID
317. See WebApps release team.'
      END
END
```

```
------------------------------------------------------------------------
----------------------------
-- 81. Auditing updates: An example which simultaneously updates a table
and stores the previous values in local vars, for comparison to see
what's changed
------------------------------------------------------------------------
----------------------------
DECLARE    @Audit nvarchar(max)
      , @oDocTypeId INT
      , @oBookingDate BIT
      , @oFromDate DATETIME2
      , @oPostingDate BIT
      , @oRate NUMERIC
UPDATE thg_DocumentMaintenanceRate
SET   @oDocTypeId = DocTypeId
      , DocTypeId = @DocTypeId
      , @oBookingDate = BookingDate
      , BookingDate = @BookingDate
      , @oFromDate = FromDate
      , FromDate = @FromDate
      , @oPostingDate = PostingDate
      , PostingDate = @PostingDate
      , @oRate = Rate
      , Rate = @Rate
      , DocumentMaintenanceId = @DocumentMaintenanceId
WHERE Id = @Id


------------------------------------------------------------------------
----------------------------
-- 82. Use conditional WHERE clauses, and only search on col if its value
not NULL
------------------------------------------------------------------------
----------------------------
-- So, only search for IDs equal to the parameter @DocTypeLookupId if
@DocTypeLookupId has been populated
-- If @DocTypeLookupId is null, the condition is ignored:
AND (@DocTypeLookupId IS NULL OR D.DocTypeLookupId = @DocTypeLookupId)
-- This one works the same way, but also allows for the col AreaLookupId
to be null, in which case the filter is again ignored.
AND (@AreaLookupId IS NULL OR IsNull(D.AreaLookupId, @AreaLookupId) =
@AreaLookupId)


------------------------------------------------------------------------
----------------------------
-- 83. Using CASE with ORDER BY
------------------------------------------------------------------------
----------------------------
-- It helps to understand what this code is doing:
-- In the first two exampoles, you are effectively defining a brand new
column AND putting values into that column AND saying please order by
this column
```

```
-- So, in the first case below, you are saying that every row will have a
new column which contains either 1, 2 or 3.
-- Whether it contains 1, 2 or 3 depends on what it has in its Name
column.
-- Finally that column is the first one used in the ordering, and values
of 1 will appear at the top of the list.
--
-- 1. Place specific values at the top of an ORDER BY list
ORDER BY    CASE Name
            WHEN '<Any>' THEN 1 -- Make sure <Any> is always at the top
of the list
            WHEN 'Other' THEN 2 -- Make sure Other is always next
            ELSE 3
            END ASC
      , Name -- apart from <Any>, just order by Name
--
-- 2. List items depending on some complex criteria to be met
-- In this example I want to order by currency first of all, then within
that cheques that are greater than 5000 to come first,
-- followed by cheques between 0 and 5000, followed by the other payment
types and negative cheques (ie everything that's left),
-- and within that I want everything ordered by owner code:
ORDER BY    S.CurrencyID,
      CASE WHEN ((S.PaymentMethodID = 5810) AND (S.Amount > 5000)) THEN 0
ELSE 1 END,
      CASE WHEN ((S.PaymentMethodID = 5810) AND (S.Amount > 0)) THEN 0
ELSE 1 END,
      S.OwnerCode
-- 3. Use a different order depending on some external factor
-- This one is (I think) slightly different because we are not defining a
new column as above, we are simply specifying a condition
-- to decide whether a column is included in the order-by or not
-- The most common use for this is to respond to the user's attempt to
order a grid by a specific column,
-- in either ascending or descending order.
DECLARE @OrderBy = 'ProductAscending'
ORDER BY
      CASE
            WHEN @OrderBy = 'ProductAscending' THEN ProductName
      END ASC,
      CASE
            WHEN @OrderBy = 'ProductDescending' THEN ProductName
      END DESC,
      CASE
            WHEN @OrderBy = 'AmountAscending' THEN Amount
      END ASC,
      CASE
            WHEN @OrderBy = 'AmountDescending' THEN Amount
      END DESC
-- For numbers, the ascending / descending distinction can be made using
positive and negative:,
DECLARE @OrderBy = 'AmountAscending'
ORDER BY
      CASE
```

```
            WHEN @OrderBy = 'AmountAscending' THEN Amount
            WHEN @OrderBy = 'AmountDescending' THEN -Amount
      END ASC


------------------------------------------------------------------------
----------------------------
-- 84. Call a sproc for every row in a table
------------------------------------------------------------------------
----------------------------
-- use a while loop to loop through all auto discounts and add each one
to the booking
DECLARE @LastCustomerID INT
SET @LastCustomerID = 0
-- define the customer ID to be handled now
DECLARE @CustomerIDToHandle INT
-- select the next customer to handle
SELECT TOP 1 @CustomerIDToHandle = CustomerID
FROM Sales.Customer
WHERE CustomerID > @LastCustomerID
ORDER BY CustomerID
-- as long as we have customers......
WHILE @CustomerIDToHandle IS NOT NULL
BEGIN
    -- call your sproc

    -- set the last customer handled to the one we just handled
    SET @LastCustomerID = @CustomerIDToHandle
    SET @CustomerIDToHandle = NULL

    -- select the next customer to handle
    SELECT TOP 1 @CustomerIDToHandle = CustomerID
    FROM Sales.Customer
    WHERE CustomerID > @LastCustomerID
    ORDER BY CustomerID
END

------------------------------------------------------------------------
----------------------------
-- 85. Call one sproc from another
------------------------------------------------------------------------
----------------------------
DECLARE @tmpDiscounts thg_discounts -- thg_discounts is a user-defined
table type
INSERT INTO @tmpDiscounts EXEC spThgDiscountsGet @BookingID

------------------------------------------------------------------------
----------------------------
-- 86. Add a computed column
------------------------------------------------------------------------
----------------------------
ALTER TABLE dbo.AssemblyTaskADD Duration AS EndTime - StartTime PERSISTED
```

```
------------------------------------------------------------------------
----------------------------
-- 87. Using the FAST query hint to quickly retrieve the first 200 rows
------------------------------------------------------------------------
----------------------------
For example, you might use the following query to quickly retrieve the
first 200 rows from the Products table:
SELECT * FROM ProductHistoryORDER BY ProdStyle DESCOPTION (FAST 200);

------------------------------------------------------------------------
----------------------------
-- 88. To determine whether a function / column is deterministic or
precise...
------------------------------------------------------------------------
----------------------------
-- You could use the following query to determine whether the
dbo.udf_get_days_empfunction is deterministic or nondeterministic:
SELECT OBJECTPROPERTY(OBJECT_ID('dbo.udf_get_days_emp'),
'IsDeterministic');
-- Similarly, To determine if expression in computed column is
deterministic and precise, you can use
-- the IsDeterministic and IsPrecise properties with the COLUMNPROPERTY
function


------------------------------------------------------------------------
----------------------------
-- 89. Creating a partitioned view.
------------------------------------------------------------------------
----------------------------
--Partitioned views are used when you have similar data stored in
multiple tables and want to create a view to
--allow access to all of the data as if it were stored in a single table.
--Partitioned views are implemented using the UNION ALL operator.
--For example, if you had three separate tables with identical
structures, you might use the following statement
--to create a partitioned view that allows users to query data from all
three tables:
CREATE VIEW PartView AS SELECT * FROM Table1 UNION ALL SELECT * FROM
Table2 UNION ALL SELECT * FROM Table3;

------------------------------------------------------------------------
----------------------------
-- 90. TRUNCATE TABLE is quicker than DELETE FROM without a WHERE clause.
------------------------------------------------------------------------
----------------------------
-- Syntax:
TRUNCATE TABLE @TempInvoiceItems;
-- (NB: Can't be used on table variables)

------------------------------------------------------------------------
----------------------------
-- 91. Use APPLY to create a new column on the fly by applying some logic
to existing columns
```

```
-------------------------------------------------------------------------
----------------------------
-- Note that the OUTER APPLY is effectively creating a table alias called
Exg, which has a column called CalculatedExchangeRate.
UPDATE     @BookingLines
SET   GrossMarginGBP =  SellPrice * @ExchangeRate - BuyPriceLocal *
Exg.CalculatedExchangeRate
FROM  @Detail
OUTER APPLY (
      SELECT     CASE  WHEN BuyPriceLocal = 0 OR BuyCurrency_Id =
@Created THEN
                        1
                  ELSE
                        ISNULL(dbo.fnThgExchangeRate(BuyCurrency_Id, 1,
@Created, @HolidayStartDate, @PackageType_Id), 0)
            END
) Exg(CalculatedExchangeRate)
WHERE [LineTypeId] NOT IN (13004, 13008) --not payments or refunds

-------------------------------------------------------------------------
----------------------------
-- 92. Sharing data between stored procedures (INSERT .. SELECT .. tables
returned by sprocs)
-------------------------------------------------------------------------
----------------------------
http://www.sommarskog.se/share_data.html#usingtable

-------------------------------------------------------------------------
----------------------------
-- 93. SELECT NULL if there is no data there (if not exists), several
times in a row - works with UNION ALL
-------------------------------------------------------------------------
----------------------------
-- If the row is simply not in the database, then you still want a result
to be output, so you can UNION ALL with the next SELECT statement
-- This is what I started with, but this was simply missing out the lines
where no results existed:
SELECT TOP 1 BO.Description
      FROM tbl_bulletOption BO
      INNER JOIN tbl_bulletTypeOption BTO ON BTO.BulletOptionId = BO.Id
      INNER JOIN tbl_bullet_types BT ON BT.BulletTypeId =
BTO.BulletTypeId
      WHERE BT.Name = 'Normal Dry'
UNION ALL
SELECT TOP 1 BO.Description
      FROM tbl_bulletOption BO
      INNER JOIN tbl_bulletTypeOption BTO ON BTO.BulletOptionId = BO.Id
      INNER JOIN tbl_bullet_types BT ON BT.BulletTypeId =
BTO.BulletTypeId
      WHERE BT.Name = 'Manufacturer Warranty'
UNION ALL
SELECT TOP 1 BO.Description FROM tbl_bulletOption BO
      INNER JOIN tbl_bulletTypeOption BTO ON BTO.BulletOptionId = BO.Id
```

```
      INNER JOIN tbl_bullet_types BT ON BT.BulletTypeId =
BTO.BulletTypeId
      WHERE BT.Name = 'Requires Own Furniture Door'

-- You can't do IF EXISTS, because that will not work with UNION ALL.
-- The answer's surprisingly simple. You basically turn each select into
a subquery and preface it with another select
-- - this forces a NULL instead of just no result at all:
SELECT 'Normal Dry', (
      SELECT TOP 1 BO.Description
            FROM tbl_bulletOption BO
            INNER JOIN tbl_bulletTypeOption BTO ON BTO.BulletOptionId =
BO.Id
            INNER JOIN tbl_bullet_types BT ON BT.BulletTypeId =
BTO.BulletTypeId
            WHERE BT.Name = 'Normal Dry')
UNION ALL
SELECT 'Manufacturer Warranty', (
      SELECT TOP 1 BO.Description
            FROM tbl_bulletOption BO
            INNER JOIN tbl_bulletTypeOption BTO ON BTO.BulletOptionId =
BO.Id
            INNER JOIN tbl_bullet_types BT ON BT.BulletTypeId =
BTO.BulletTypeId
            WHERE BT.Name = 'Manufacturer Warranty')
UNION ALL
SELECT 'Requires Own Furniture Door', (
      SELECT TOP 1 BO.Description FROM tbl_bulletOption BO
            INNER JOIN tbl_bulletTypeOption BTO ON BTO.BulletOptionId =
BO.Id
            INNER JOIN tbl_bullet_types BT ON BT.BulletTypeId =
BTO.BulletTypeId
            WHERE BT.Name = 'Requires Own Furniture Door')

------------------------------------------------------------------------
----------------------------
-- 94. Don't join if the joining column has NULL in it
------------------------------------------------------------------------
----------------------------
-- This is simpler than you think... just use a LEFT JOIN


------------------------------------------------------------------------
----------------------------
-- 95. What does COALESCE do?
------------------------------------------------------------------------
----------------------------
-- COALESCE just takes a series of arguments, and returns the first
non-null argument
-- This will return 'no text' if Text is NULL:
SELECT
      COALESCE(Text , 'no text')
FROM
      TextTable
```

```
-------------------------------------------------------------------------------
-----------------------------
-- 96. Using COALESCE and LEFT JOIN to return a French-language row if
one exists, English if not.
-------------------------------------------------------------------------------
-----------------------------
-- By using LEFT JOIN, we ensure we get a NULL row if no relevant
(French-language) row exists
SELECT
      COALESCE (FrenchInfo.Text, EnglishInfo.Text)
FROM
      laterooms.HotelImportantInformation as EnglishInfo
      LEFT JOIN laterooms.HotelImportantInformation as FrenchInfo
            ON FrenchInfo.HotelId = EnglishInfo.HotelId
            AND FrenchInfo.LanguageId = 2
WHERE
      EnglishInfo.HotelId = 87034
      And EnglishInfo.LanguageId = 1

-------------------------------------------------------------------------------
-----------------------------
-- nn Description
-------------------------------------------------------------------------------
-----------------------------

-------------------------------------------------------------------------------
-----------------------------
-- nn Description
-------------------------------------------------------------------------------
-----------------------------

-------------------------------------------------------------------------------
-----------------------------
-- nn Description
-------------------------------------------------------------------------------
-----------------------------

-------------------------------------------------------------------------------
-----------------------------
-- nn Description
-------------------------------------------------------------------------------
-----------------------------
```