SQL

Table of contents

General

- No more than 200 lines (formatted) allowed in any stored procedure, user-defined function or view. *If more than 200 lines is unavoidable, please discuss this with a Senior Developer or Database Administrator first.*
- Aliases should be used for tables in SELECTs for readability purposes and alias names should make sense. If alias is too long (>10 characters) , use an abbreviation that makes sense.
- **EditedUser** in a SQL batch job is the name of the stored procedure where it is called from. When the edited user is provide to the procedure the SP name must be appended to e.g. {Username}_MyProc. This must be applied to any parent procedure that perform data modification.
- Script name assigned correctly http://shakespeare/MasterBuilder/Tools/GenerateScript when committing your scripts.
- *Stored Procedure, Views and Functions heading convention* followed (see example below)

> 2018/11/30 : dbotha : 56789 : Added check for qualifications offered by my learning provider.
> {Date} : {Author} : {TP#} Short concise description of change

- **UPDATE** statements should always contain a **WHERE** clause.
- **WITH(NOLOCK)** should be used in SELECTs to prevent unnecessary locking.
- **SQL KEYWORDS** should always be in **CAPS.**
- Dynamic SQL should only be used when absolutely necessary.
- The **SchemaId** column in tables must not be nullable.
- The statement **SET NOCOUNT ON** should be at the top of the stored procedure unless in the unlikely case where the counts obtained from the stored procedure is used in code.
- When performing an INSERT, always specify the column list

  > **Correct** : INSERT INTO tmp (Value) SELECT @variable INSERT INTO tmp (Value) VALUES(@variable)

  > **Not correct** : INSERT INTO tmp SELECT @variable INSERT INTO tmp VALUES(@variable)

- When writing an automation stored procedure in SQL and the code becomes too complex and long-winded, break the stored procedure into multiple stored procedures.
- Using in-line user defined functions in SELECT statements should only be used if absolutely necessary. NOTE : Complex user-defined functions used in-line in large SELECT statements returning many rows (1000+) can potentially slow down your code significantly. In-line user defined functions should be tested on large amounts of data.

Naming conventions

General

- Decide per module if abbreviation (e.g. *prs* for Personnel module) or full name (e.g. *leave*) will be used for database Objects.
- Do not use spaces in the names of database objects.
- Avoid using **ntext**, **text**, and **image** data types in new development work. Use nvarchar (max), varchar (max), and varbinary (max) instead.

Note: The parent / grouping determines the module the data is stored on E.g. EmployeePDPs (an employee's PDP's) vs. pdpPeriod (a PDP's periods)

Table 1 : Database Module Abbreviations

Abbreviation	**Module**
cc	Career Conversation
cfg	Configuration
cl	Catalogue
com	Communication (Import / Export)
cpd	Credits
dbd	Dashboards
ab	Assessment Builder
ee	Employee Evaluation
el2	E-Learning v2
el	e-Learning
els	Learning Store

em	Event Management / Training and Scheduling
fais	FAIS
ate	Ask The Expert/ Discussion Forum
icn	Icodeon
ir	Internal Relations / Disciplinary Actions
jl	Job Leveling
jp	Job Profiler
leave	Leave Management - New
lic	Licences
mc	Mentors and Coaches
ntf	Notifications
org	Organisational Structure
pdm	Performance Management
pdp	Personal Development Plan
prc	HR Processes
prs	Personnel
pw	Pathways
rb	Report Builder
rec	Recruitment
rem	Remuneration

rp	Resource Planning
rpt	Reports / Report Management
sms	SMS Notifications
sr	Salary Review
ss	Salary Scenario
sty	System Framework
sys	System Administration
tal	Talent Management - New
sc	Succession and Career Planning
tM	Talent Management - Old
txAudit	Auditing - Old
wf	Work Flows
tr_	Trigger

Tables

- A table name must always be prefixed with the module name abbreviation (see above).
- A database table name must always be plural
  - *prsEmployees* – There will most likely be more than one employee in the system
  - *LeaveGroupTypes* – Each Leave Group can have one or more Leave Type
  - *eeImports* – Only one import can run at a time
  - *pdpStatuses* – There are multiple statuses for the module
- A column name must be the shortest descriptive name possible
  - Do not specify module prefix e.g.

    > ***Correct column name*** *EmployeeId*

    > ***Incorrect column name*** *prsEmployeeId*

  - **Exceptional case:** If more than one column in the same table are the “same” e.g. *CategoryId*, specify module prefix e.g. *cpdCategoryId, pdpCategoryId*
- A column name must refer to a single and not multiple instances

    - Use *UnitId* instead of *UnitsId*
- Rather use **varchar(max)** instead of **text** or **varchar(8000)** types for string columns where applicable
- Each table that has a single identity column must also have a clustered primary key with the following naming convention:
    - PK_{*TableName*}_{*IdentityColumn*}
- A foreign key constraint name must be in the following format:
    - FK_{*TableName*}_{*Column1*}

Views

- A View's name must follow the same convention as table names (add *View* at end of name)
    - Use *prsTerminatedEmployeesView* instead of *viewTerminatedEmployees*
    - *naming: {prefix}{Description of the data resturned}View*

User-Defined Functions

- A User-Defined function's name must follow the same convention as table names.
- A user defined function must be prefixed with the module (do not add *fnc* prefix)
    - Use *el2SelectScholarshipManagerNotificationDays* instead of *fncSelectScholarshipManagerNotificationDays*

- General user defined functions (module-unspecific) can be the description of the output
    - Use *Split* instead of *fncSplit*
    - Use *CleanHtmlTags* instead of *fncCleanHTMLTags*

Stored Procedures

- A stored procedure name must always be prefixed with the module.
- A stored procedure must indicate it's intention by using a keyword on what action will be performed
    - Select
    - InsertUpdate
    - Insert
    - Update
    - Delete
    - Check/ Verify
    - Copy
    - Archive
    - Reset
    - Apply
- Examples
    - prsSelectEmployeesAll
    - prsSelectEmployeesList - paging
    - prsSelectEmployee - single

- prsInsertUpdateEmployee
- prsDeleteEmployee
- prsCheckEmployeeIDNumber
- styResetUserPassword
- pdmCopyContract

Temporary Tables

- Single use temp table: #{Descriptive table name}
- Global use temp table: ##{Descriptive table name}
- Variable temp table @{Descriptive table name}

Common Table Expressions (CTE's)

- CTE table names are declared with the prefix **cte.**

Indexes

- A non-clustered index name must be in the following format:
  - IX_{*TableName*}_{*Column1*}_{*Column2*}
  - Indexes have a maximum size of 900 or 1700 depending on the index type and SQL version. Do not create a non-clustered index on a column with a max length of more than 500.

- Always check with the Database Administrator whether indexes should be created during development. Assume that indexes will always be created.

Constraints

- A default constraint name must be in the following format:
  - DF_{*TableName*}_{*Column1*}

- A unique constraint name must be in the following format:
  - UQ_{*TableName*}_{*Column1*}_{*Column2*}

- A check constraint name must be in the following format:
  - CK_{*TableName*}_{*Column1*}_{*Column2*}

- Columns with Default value constraint should not allow NULLs.

TODO: Add a link to main page for each section

Cursors

- Use cursors only when absolutely necessary.
- If the function performed by the cursor could have been achieved by another SQL function e.g. PIVOT or Common Table Expression then rather do that as CURSORS are expensive.
- When using a cursor to only cycle once through records without updating them, use the following syntax to make the cursor as light as possible:

```
DECLARE @SchemaID INT

DECLARE curs CURSOR LOCAL FORWARD_ONLY STATIC READ_ONLY FOR
  SELECT
    SchemaID
  FROM cfgSchemaID WITH(NOLOCK)
  WHERE
    GETDATE() BETWEEN ValidFrom AND ValidTo
    AND SysID = 101

OPEN curs

FETCH NEXT FROM curs
INTO @SchemaID

WHILE @@FETCH_STATUS = 0
BEGIN

     /*Do your commands for @SchemaID here*/

/*Get the next author.*/
FETCH NEXT FROM curs
INTO @SchemaID
END

CLOSE curs
DEALLOCATE curs
```

- When evaluating the use of cursors first consider the use of for XML path to loop through each item in a table

Common table expressions (CTE's)

- CTE table names are declared with the prefix **cte**
- Used to simplify complex joins and subqueries.
- Use a Common Table Expression for paging instead of Dynamic SQL.
- Always start with a semi-colon before the WITH.
- Chaining CTE's must be limited to 3 instances.
- CTE's must be filtered as soon as possible to limit the number of records stored in memory.
- CTEs can only be used when data is only required for a single use in the procedure.
- CTEs must always be provided named column and not use the * selector.

```
;WITH cteEmployees
    AS (SELECT
              Name
            , Surname
            , EmployeeNumber
        FROM
            prsEmployees WITH(NOLOCK))
    SELECT
          *
    FROM
        cteEmployees
```

The use of recursive CTEs

- Always ensure a termination condition is defined.
- For an example view this site
- e.g.

```
WITH Managers AS
(
--initialization
SELECT EmployeeID, LastName, ReportsTo
FROM Employees
```

```
WHERE ReportsTo IS NULL
UNION ALL
--recursive execution
SELECT e.employeeID,e.LastName, e.ReportsTo
FROM Employees e INNER JOIN Managers m
ON e.ReportsTo = m.employeeID
)
SELECT * FROM Managers
```

Temporary tables

- Temp tables are used for the large temporary storage of data.
- Only use local temp tables.
- Use temporary tables cautiously / only when necessary e.g. early filtering in reports / complex queries.
- When a temporary table is used in a stored procedure, evaluate if it is absolutely necessary.
- Ensure that temporary table are always explicitly dropped at the end of the stored procedure.
- When the possibility exist that the temp table does not exist test its existence in the temp..DB before dropping e.g. conditionally created temp table

```
IF OBJECT_ID('tempdb..#TheTable') IS NOT NULL
BEGIN
/*Do Stuff*/
END
```

- Create the table before addition when a fixed definition is required or multiple data sources are used to populate it

```
    CREATE TABLE #LeaveCycles(
        StartDate   DATETIME,
        EndDate     DATETIME,
        LeaveTypeId INT,
        Name        VARCHAR(200) COLLATE DATABASE_DEFAULT,
        CycleId     INT,
        CreatedDate DATETIME,
        EmployeeId  INT
                    )

    INSERT INTO #LeaveCycles(
          StartDate
        , EndDate
        , LeaveTypeId
        , Name
        , CycleId
        , CreatedDate
        , EmployeeId
```

```
        )
    EXEC LeaveCalculateActiveCyclesByLeaveType
        @EmployeeId = @EmployeeId,
        @SchemaId = @SchemaId,
        @LeaveTypeId = @LeaveTypeId,
        @IsHistoric = @IsHistoric
```

- When a single table is used a select into can be done to create the temp table

Table variables

- Use table variables over temp tables for a small quantity of data (thousands of bytes)

https://stackoverflow.com/questions/11857789/when-should-i-use-a-table-variable-vs-temporary-table-in-sql-server

- Example:

```
DECLARE @product_table TABLE (
    product_name VARCHAR(MAX) NOT NULL,
    brand_id INT NOT NULL,
    list_price DEC(11,2) NOT NULL
);

INSERT INTO @product_table
(
    product_name,
    brand_id,
    list_price
  )
SELECT
    product_name,
    brand_id,
    list_price
FROM
    production.products
WHERE
    category_id = 1;
```

Tables and indexes

- Always use a column list in INSERT statements of SQL queries. This will avoid a problem when table structure changes.

  > **Correct** : INSERT INTO tmp (Value) SELECT @variable
  >
  > INSERT INTO tmp (Value) VALUES(@variable)

  > **Not correct** : INSERT INTO tmp SELECT @variable
  >
  > INSERT INTO tmp VALUES(@variable)

- Perform all referential integrity checks and data validations using constraints instead of triggers, as they are faster.
- Remember to add foreign-key constraints where a table references another.
- Always check with the Database Administrator to confirm what indexes should be added when a new table is added to the database.

User defined functions

- Do not call functions repeatedly in stored procedures, triggers, functions and batches, instead call the function once and store the result in a variable, for later use.
- Unless absolutely necessary, DO NOT USE in-line user-defined functions in SELECTs. *If unavoidable, discuss with a Senior Developer first before implementing it.*
- *When used it must be if possible only use data provided and not do extra selects from other tables.*

Stored procedures

- **EXCEPT** or **NOT EXIST** clause can be used in place of LEFT JOIN or NOT IN for better performance (see example for **EXCEPT** below)

```
SELECT EmpNo, EmpName
FROM EmployeeRecord
WHERE Salery > 1000
EXCEPT
SELECT EmpNo, EmpName
FROM EmployeeRecord
WHERE Salery > 2000
ORDER BY EmpName;
```

- If stored procedure always returns single row resultset, then consider returning the resultset using OUTPUT parameters instead of SELECT statement
- Use query hints to prevent locking if possible (NoLock)
- Avoid using dynamic SQL statements if you can write T-SQL code without using them.
- The number of nested procedures must be limited to no more than 32

Triggers

- Limit the use of triggers only for auditing, custom tasks, and validations that cannot be performed using constraints.
- If possible only exec trigger conditionally e.g. modifying data

Existence checks

- Make use of the existence checking defined here
- Always check for existence when adding new objects to the database (see example below)

```
IF NOT EXISTS
        (
        SELECT TOP 1
            1
        FROM
            sys.all_columns c
            JOIN sys.tables t
              ON t.object_id = c.object_id
        WHERE t.name = 'EmployeeLeave'
             AND c.name = 'ActionStatus')
  BEGIN
    ALTER TABLE EmployeeLeave
    ADD
          ActionStatus INT
END
```

- Also check for existence when editing a database object (see example below)

```
IF EXISTS
        (
        SELECT TOP 1
            1
        FROM
            sys.all_columns c
            JOIN sys.tables t
              ON t.object_id = c.object_id
        WHERE t.name = 'EmployeeLeave'
             AND c.name = 'ActionStatus')
  BEGIN
    ALTER TABLE EmployeeLeave
    ALTER COLUMN ActionStatus NVARCHAR(2) NOT NULL
```

```
END
```

Views

- Incorporate your frequently required, complicated joins and calculations into a view so that you don’t have to repeat those joins/calculations in all your queries. Instead, just select from the view.
- In views always define selects with named columns.
- Avoid the use of views within views.
- If possible rather implement procedures to get filtered datasets.