**Database Guideline Document**

**for   
KaspickDB**

**Version Release History**

|  |  |
| --- | --- |
| **Document Name:** | Database Guideline Document |
| **Author:** | Niveditha Narasimha |
| **Status:** | Final |
| **Approved by:** |  |

|  |  |  |
| --- | --- | --- |
| **Date** | **Version #** | **Change Description** |
| 01-Apr-2013 | v1 | Draft |
| 11-Apr-2013 | v2 | Includes additional inputs from Kaspick |
| 29-Apr-2013 | v3 | Input from Abhishek , Kasper app prefix added, TOC added |

Template Meta Information

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Date** | | | **Version #** | **Change Description** | | **Author** | |
| 26-Jun-2009 | | | 2.0 | Changed ITEAMIC to CIBER documentation format. | | SEPG | |
| 25-May-2010 | | | 5.0 | Updated the template to meet the CM process requirement as per CMMI 1.3v | | SEPG | |
| 10-Jan-2010 | | | 4.0 | Section 1.1:Added Naming Convention column  Section 1.2 is deleted | | SEPG | |
| 09-Aug-2011 | | | 5.0 | Updated the section for alternate design 1 and 2 in section 3 | | Ashok Singh | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
|  | | |  |  | |  | |
| **#** | **Reviewer List** | | | **Location** | |
| 1 | Suresh Bagalore | | | Kaspick | |
| 2 | Manjiri Purohit | | | Kaspick | |
| 3 | Gary Wells | | | Kaspick | |
| 4 | Abhishek Chadha | | | Kaspick | |
| 5 | Anne Dyer-Bennet | | | Kaspick | |
|  |  | | |  | |

|  |  |  |
| --- | --- | --- |
| **#** | **Distribution List** | **Location** |
| 1 | Suresh Bagalore | Kaspick |
| 2 | Manjiri Purohit | Kaspick |
| 3 | Gary Wells | Kaspick |
| 4 | Abhishek Chadha | Kaspick |
| 5 | Anne Dyer-Bennet | Kaspick |
| 6 | Onshore & Offshore Development team | Ciber |

**Table of Contents:**

[DB Naming Conventions 4](#_Toc355027166)

[Naming Convention Guidelines – 6](#_Toc355027167)

[DB Coding guidelines - 8](#_Toc355027168)

[APPENDIX SECTION 12](#_Toc355027169)

DB Naming Conventions  
Below are the naming conventions for Database Objects -

1. **Table Naming Conventions -**   
   TBL\_<APP>\_<Name>  
   **Example -** **TBL\_PP\_AccountRule**  
   Note: Please refer appendix for <APP> - application names

***Prefix for the table types -***Audit table - <AUDIT>

Staging table - <STG>

Archival table - <ARCH>  
**Example** - **TBL\_PP\_AUDIT\_AccountRule** OR **TBL\_PP\_STG\_AccountRule**

1. **Primary Key column naming convention -**   
   PK\_< TableName >\_<ColName1>\_<ColName2>  
    **Example** – **PK\_TBL\_PP\_ AccountRule\_SLNO**
2. **Foreign Key naming convention -**   
   FK\_<ReferenceTableName>\_<ReferencingTableName>\_< ColName>  
   **Example:**  
   ALTER TABLE TBL\_PP\_ValidationResult   
    ADD CONSTRAINT   
    **FK \_TBL\_PP\_ValidationResult\_TBL\_EIS\_PP\_ValidationRule\_RuleID** FOREIGN KEY ([RuleID])  
    REFERENCES [TBL\_EIS\_PP\_ValidationRule] ([RuleID])
3. **Unique Key naming convention -**   
   UK\_< TableName>\_< ColName1>\_<ColName2>  
   **Example:**  
   ALTER TABLE TBL\_PP\_AccountRule  
    ADD CONSTRAINT   
    **UK\_TBL\_PP\_AccountRule\_CustomerAccountNumber\_RuleID** UNIQUE NONCLUSTERED

([CustomerAccountNumber] ASC

,[RuleID] ASC)

1. **Index naming convention -**IX\_ <TableName>\_< ColName>

**Example** –

*Single Column –*

**IX\_TBL\_PP\_AccountRule\_RuleID**

*In case of multiple columns –*

**IX\_TBL\_PP\_AccountRule\_ClientID\_TemplateID**(Index on TBL\_PP\_TemplateTypeRule for ClientID, TemplateID)

1. **Procedure naming convention -**USP\_<APP>\_<Action><ProcedureName>  
   **Example** - USP\_PP\_GetAccountRule

Actions -

* Get - Fetch record
* Save – Compound Insert/Update/Delete record
* Ins – Insert only
* Upd – Update only
* Del – Delete only

Note: Please refer appendix for <APP> - application names

1. **Trigger naming convention**TRG\_<APP>\_<Action><TriggerName>

**Example** - TRG\_PP\_InsAccountRule

Action can be **Insert/Update/Delete/Audit**

* Ins – Insert only
* Upd – Update only
* Del – Delete only
* Aud - Audit

1. **View naming convention**VW\_<APP>\_<ViewName>  
   **Example** - VW\_PP\_AccountRule
2. **Default Constraint naming convention**DF\_<TableName>\_<ColName> **Example** - ALTER TABLE [dbo].[TBL\_PP\_AccountRule] ADD CONSTRAINT

**[DF\_PP\_AccountRule\_DeleteFlag**] DEFAULT (0) FOR [DeleteFlag]

1. **Check Constraint naming convention**CK\_<TableName>\_<ColName> **Example** *-* ALTER TABLE [dbo].[TBL\_PP\_AccountRule] ADD CONSTRAINT [**CK\_PP\_AccountRule\_DeleteFlag**] CHECK CreditRating>0
2. **Function naming convention**  
   FN\_ <FunctionName>

**Example** – FN\_ ComputeSum

1. **Synonym naming convention**SYN\_<DBNickname>\_<ObjectName>  
   **Example** – SYN\_GW\_TBL\_PP\_AccountRule  
   Note: Limit DBNickName to two characters here
2. **Rule Naming Convention**RL\_<TableName>\_<ColName>

**Example –** RL\_TBL\_PP\_AccountRule\_RuleID

Naming Convention Guidelines –

1. Do not use spaces in the names
2. DB Object name should reflect the object’s content and function
3. Do not use keywords in the name
4. Object Name should be Singular, for example “Customer” instead of “Customers”
5. Follow the grouping specified for applications while creating objects (Refer appendix for Application Group Names)
6. The object name should be **PascalCase** without an underscore. Underscore should be used for standard prefix specified above   
   **Example** – Table names for Paragon should start with TBL\_PP\_<ObjectName>.
7. Column/Variable name should be meaningful
8. Audit columns like CreatedBy, CreatedDate, ModifiedBy, ModifiedDate should be added as standard audit columns in table. Any changes in audit should be for specific requirements only.
9. No Columns or variables should have VARCHAR (MAX) as their datatype.
10. The size of varchar datatype should be consistent numbers. It cannot have odd numbers like 25, 35, 40 etc.
11. Any Key constraints created on a table should always be clustered unless they are performing frequent DML operations. Non key indexes created on a table should be **nonclustered.**
12. Limit the use of abbreviations (can lead to misinterpretation of names). Use “Account” in place of “Acct”
13. Foreign key fields should have the exact same name as they have in the parent table where the field is the primary. For example, in the Customers table the primary key field might be "CustomerId". In an Orders table where the customer id is kept, it would also be "CustomerId". There is one exception to this rule, which is when you have more than one foreign key field per table referencing the same primary key field in another table. In this situation, it might be helpful to add a descriptor before the field name. An example of this is if you had an Address table. You might have another table with foreign key fields like HomeAddressId, WorkAddressId, MailingAddressId, or ShippingAddressId
14. Datatype specific naming - Bit fields should be given affirmative Boolean names like "IsDeleted", "HasPermission", or "IsValid" so that the meaning of the data in the field is not ambiguous; Negative Boolean names are harder to read when checking values in T-SQL because of double-negatives (e.g. “Not IsNotDeleted”). If the field holds date and/or time information, the word "Date" or "Time" should appear somewhere in the field name. It is sometimes appropriate to add the unit of time to the field name also, especially if the field holds data like whole numbers ("3" or "20"). Those fields should be named like "RuntimeHours" or "ScheduledMinutes".
15. Naming your views should be different depending on the type or purpose of the view. For simple views that just join one or more tables with no selection criteria, combine the names of the tables joined. For example, joining the "Customer" and "StateAndProvince" table to create a view of Customers and their respective geographical data should be given a name like "CustomerStateAndProvince"
16. Limit the use of acronyms (some acronyms have more than one meaning e.g. "ASP")
17. An Object name should be readable.
18. Fillfactor – please refer the link - <http://blog.sqlauthority.com/2011/01/31/sql-server-what-is-fill-factor-and-what-is-the-best-value-for-fill-factor/>  
    If anyone has any doubts on this, please contact the DB Designer at onsite/offshore
19. There should not be any variable Names like var1, temp1 etc. The variable should be named meaningfully w.r.t the context used

DB Coding guidelines -

1. Using JOIN is better for performance then using sub queries or nested queries

With ANSI joins, the WHERE clause is used only for filtering data whereas with older style joins, the WHERE clause handles both the join condition and filtering data. The first of the following two queries shows the old style join, while the second one show the new ANSI join syntax:

1. SELECT a.au\_id, t.title FROM titles t, authors a, titleauthor ta WHERE a.au\_id = ta.au\_id AND ta.title\_id = t.title\_id AND t.title LIKE ‘%Computer%’
2. SELECT a.au\_id, t.title

FROM authors a

**INNER JOIN**

titleauthor ta

ON

a.au\_id = ta.au\_id

**INNER JOIN**

titles t

ON

ta.title\_id = t.title\_id

WHERE t.title LIKE ‘%Computer%’

1. Store image paths or URLs in database instead of images. It has less overhead.
2. Use proper database types for the fields. If StartDate is database field use datetime as datatypes instead of VARCHAR (20). The Varchar should be defined with appropriate length it should not be MAX.
3. Specify column names instead of using \* in SELECT statement.
4. Use LIKE clause properly. If you are looking for exact match use “=” instead.
5. Proper indexing will improve the speed of operations in the database. The key columns frequently searched should always be indexed. Make sure, you do not index a column which is updated frequently.
6. Use SET NOCOUNT ON at the beginning of your SQL batches, stored procedures and triggers in production environments, as this suppresses messages like ‘(1 row(s) affected)’ after executing INSERT, UPDATE, DELETE and SELECT statements. This improves the performance of stored procedures by reducing network traffic.
7. Do not hardcode any parameters, instead use parameter passing. This will enable the code reusability (method overload).
8. Use Common Table Expressions (CTE) instead of temporary table as much as possible.
9. Avoid dynamic SQL statements as much as possible. Dynamic SQL tends to be slower than static SQL, as SQL Server must generate an execution plan every time at runtime. IF and CASE statements come in handy to avoid dynamic SQL.
10. Perform all your referential integrity checks and data validations using constraints (foreign key and check constraints) instead of triggers, as they are faster. Limit the use triggers only for auditing, custom tasks and validations that cannot be performed using constraints. Constraints save you time as well, as you don’t have to write code for these validations, allowing the RDBMS to do all the work for you.
11. Always check the global variable @@ERROR immediately after executing a data manipulation statement (like INSERT/UPDATE/DELETE); so that you can rollback the transaction in case of an error (@@ERROR will be greater than 0 in case of an error). This is important, because, by default, SQL Server will not rollback all the previous changes within a transaction if a particular statement fails. This behaviour can be changed by executing SET XACT\_ABORT ON. The @@ROWCOUNT variable also plays an important role in determining how many rows were affected by a previous data manipulation (also, retrieval) statement, and based on that you could choose to commit or rollback a particular
12. Offload tasks, like string manipulations, concatenations, row numbering, case conversions, type conversions etc., to the front-end applications if these operations are going to consume more CPU cycles on the database server. Also try to do basic validations in the front-end itself during data entry. This saves unnecessary network roundtrips
13. Use normalized tables in the database. Small multiple tables are usually better than one large table. Make sure you normalize your data at least to the 3rd normal form. At the same time, do not compromise on query performance. A little bit of denormalization helps queries perform faster
14. Follow standard naming conventions while creating database objects and provide proper aliases during join operations
15. Use comments for readability as well guidelines for next developer who comes to modify the same code. Proper documentation of application will also aid help too
16. For procedures use code headers (templates). Comments should be descriptive in code history section of the template as well as the line of code change. The description should include the DATE, NAME OF THE PERSON WHO CHANGED THE CODE as well as the CLEAR DESCRIPT OF WHAT WAS CHANGED!
17. Formatting and indentation of the code must be followed religiously.

**Example** –

SELECT TempFootNoteTable.Columns.value('@FootnoteID', 'BIGINT'),

TempFootNoteTable.Columns.value('@FootnoteClientID', 'BIGINT'),

TempFootNoteTable.Columns.value('@FootNoteType', 'CHAR(1)'),

TempFootNoteTable.Columns.value('@Footnote', 'VARCHAR(MAX)'),

TempFootNoteTable.Columns.value('@IsGeneric', 'BIT'),

TempFootNoteTable.Columns.value('@ClientID', 'BIGINT'),

(case when TempFootNoteTable.Columns.value('@CategoryID', 'BIGINT')=0 then null

else TempFootNoteTable.Columns.value('@CategoryID', 'BIGINT') end) as CategoryID,

(CASE WHEN DATEDIFF(DAY,TempFootNoteTable.Columns.value('@CurrentUntilDate',

'DATETIME'),'01/01/1900')=0 THEN NULL

ELSE TempFootNoteTable.Columns.value('@CurrentUntilDate', 'DATETIME') END) as

EffectiveThroughDate,

TempFootNoteTable.Columns.value('@Display', 'BIT')

FROM

@FootNoteXML.nodes('//FootnoteCollection/InsertList/Footnote')AS TempFootNoteTable(Columns)

1. Temp Storage -

Table variables use internal metadata in a way that prevents the engine from using a table variable within a parallel query. SQL Server maintains statistics for queries that use temporary tables but not for queries that use table variables. Without statistics, SQL Server might choose a poor processing plan for a query that contains a table variable. So, limit your use of SQL Server 2000 table variables to reasonably small queries and data sets and use temporary tables for larger data sets.  
Use temporary tables (e.g. #NslClaims) only when absolutely necessary. When temporary storage is needed within a T-SQL statement or procedure, it’s recommended that you use local table variables (e.g. @NslClaims) instead if the amount of data stored is relatively small. This eliminates unnecessary locks on system tables and reduces the number of recompiles on stored procedures. This also increases performance as table variables are created in RAM, which is significantly faster than physical disk.

For transactions, the temp table and table variables operate differently. Temp tables can be rolled back, whereas table variables cannot be rolled back. Please refer the link here for more details - <http://blog.sqlauthority.com/2009/12/28/sql-server-difference-temp-table-and-table-variable-effect-of-transaction/>

1. Do not use cursors, as it impacts performance
2. Deletion of obsolete objects should be taken care as part of any database activity.
3. Use Unicode data types, like NCHAR, NVARCHAR, or NTEXT, ONLY if your database is going to store not just plain English characters, but a variety of characters used all over the world. Use these data types only when they are absolutely needed as they use twice as much space as non-Unicode data types.
4. Always use a column list in your INSERT statements. This helps in avoiding problems when the table structure changes (like adding or dropping a column). Here's an example that shows the problem.

Here's an INSERT statement without a column list , which works perfectly:

*INSERT INTO EuropeanCountries   
SELECT 1, 'Ireland'*

Now, let's add a new column to this table:

*ALTER TABLE EuropeanCountries ADD EuroSupport bit*

Now if you run the above INSERT statement. You get the following error from SQL Server:

*Server: Msg 213, Level 16, State 4, Line 1 Insert Error: Column name or number of supplied values does not match table definition.*

1. Use the CHAR data type for a column only when the column is non-nullable. If a CHAR column is nullable, it is treated as a fixed length column in SQL Server 7.0+. So, a CHAR (100), when NULL, will eat up 100 bytes, resulting in space wastage. So, use VARCHAR (100) in this situation.
2. Avoid using search arguments in the WHERE clause, such as “IS NULL”, “<>”, “!=”, “!>”, “!<”, “NOT”, “NOT EXISTS”, “NOT IN”, “NOT LIKE”, and “LIKE ‘%500'”, charindex , patindex etc. It generally prevents (but not always) the query optimizer from using a useful index to perform a search.
3. Joins in sql query should be made on integer/indexed columns
4. If an input to procedure is huge set of rows OR comma separated values, data must be passed as XML.
5. BIGINT should not be used for numeric data type
6. NULL vs. BLANK - Any input to procedure should beNULL or ANY VALUE, but NOT blank, space or empty string. Character/String should be trimmed before passing the input to procedure. Possible String handling and manipulation should be done in the .net/other source and not in the database code.
7. SQL Server Profiler must be executed after executing a complex query/logic and regression testing of the application to check the performance of the code.
8. Error handling mechanism using Try/Catch block in transactions -

**Example -**

CREATE PROCEDURE dbo.spDoStuff @value INT

AS

SET NOCOUNT ON

BEGIN TRY

BEGIN TRAN

DELETE

FROM MyTable

WHERE Col1 = @value

INSERT INTO MyOtherTable (Col1)

SELECT @value

COMMIT TRANSACTION

END TRY

GO

BEGIN CATCH

ROLLBACK TRAN

DECLARE @ErrorMessage NVARCHAR(4000);

DECLARE @ErrorSeverity INT;

DECLARE @ErrorState INT;

SELECT @ErrorMessage = ERROR\_MESSAGE()

,@ErrorSeverity = ERROR\_SEVERITY()

,@ErrorState = ERROR\_STATE();

RAISERROR (

@ErrorMessage

,-- Message text.

@ErrorSeverity

,-- Severity.

@ErrorState -- State.

);

PRINT N'The transaction is in an uncommittable state. ' + 'Rolling back transaction.'

END

1. All the stored procedure that needs to be modified should be generated using SQL Server management studio and **NOT USING sp\_helptext** as using sp\_helptext does not include the stored procedure specific includes/sets like SET ANSI NULL off/on, SET QUOTED\_IDENTIFIER on/off

APPENDIX SECTION

**<APP> - Application Prefix**

* PP – Paragon
* TR – TRex
* IE – Income Estimation
* IR – IRIS
* PV – PEVA
* KS - Kasper