SQL Best Practices

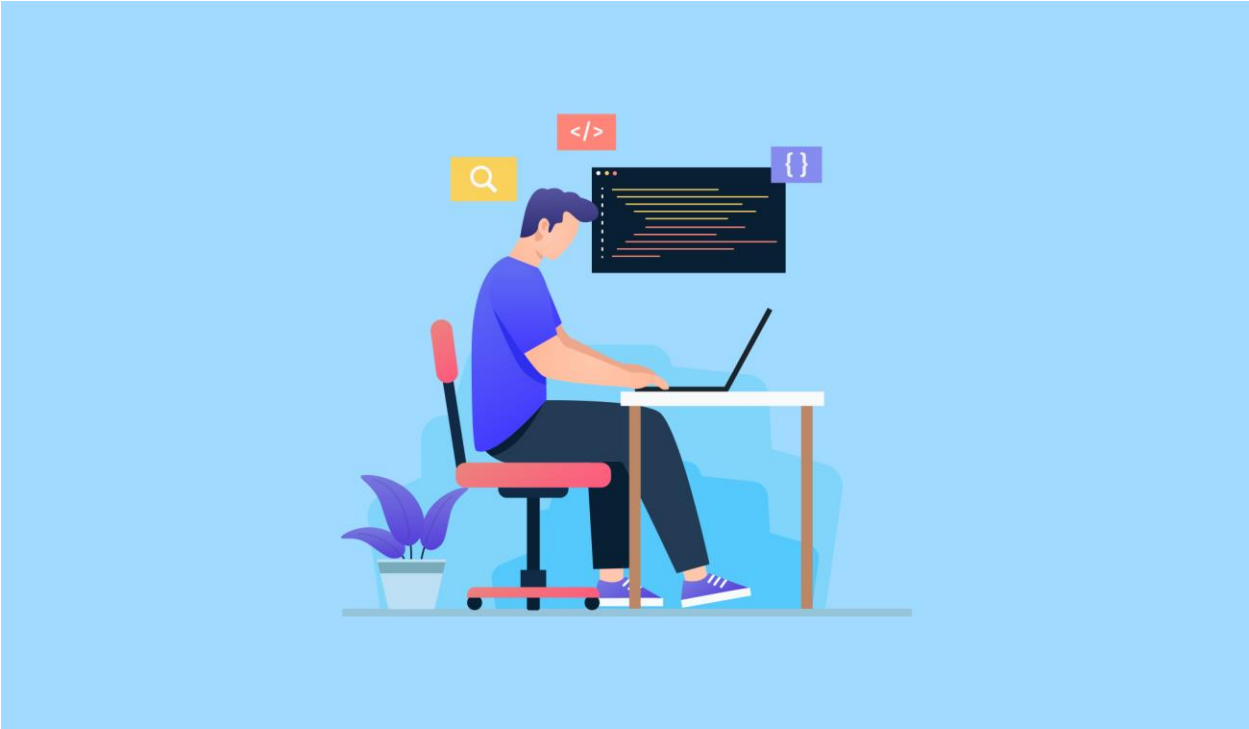
Lesson Agenda

- Coding Standard/Convention
- Write query effectively

Section 1

SQL CODING STANDARD

SQL Coding Standard



- Standardization has a positive impact on any business. It requires consistent efforts and sheers the focus of the software development team to meet quality goals.

SQL Coding Standard

NAMING CONVENTION AND STYLES

1. **Use UPPER CASE for all T-SQL constructs, excepts Types**

Correct:

```
SELECT MAX([Salary]) FROM dbo.[EmployeeSalary]
```

Incorrect:

```
SELECT max([Salary]) from dbo.[EmployeeSalary]
```

2. **Use lower case for all T-SQL Types and Usernames**

Correct:

```
DECLARE @MaxValue int
```

Incorrect:

```
DECLARE @MaxValue INT
```

3. Use Pascal casing for all UDO's

Correct:

```
CREATE TABLE dbo.EmployeeSalary
(
      EmployeeSalaryID    INT
)
```

Incorrect:

```
CREATE TABLE dbo.Employeesalary
(
      EmployeesalaryID    int
)
```

4. **Avoid abbreviations and single character names**

Correct:

```
DECLARE @RecordCount int
```

Incorrect:

```
DECLARE @Rc int
```

5. **UDO naming must confer to the following regular expression ([a-zA-Z][a-zA-Z0-9]).**

Do not use any special or language dependent characters to name objects. Constraints can use the underscore character.

Correct:

```
CREATE TABLE dbo.[EmployeeSalary]
```

Incorrect:

```
CREATE TABLE dbo.[Employee Salary]
```

6. Use the following prefixes when naming objects

usp_: User stored procedures

ufn_: User defined functions

view_: Views

IX_: Indexes

usp_: User stored procedures

DF_: Default constraints

PK_: Primary Key constraints

FK_: Foreign Key constraints

CHK_: Check constraints

UNI_: Unique constraints

Correct:

```
CREATE PROCEDURE dbo.usp_EmployeeSelectAll
```

Incorrect:

```
CREATE PROCEDURE dbo.EmployeeSelectRetired --without preffixed
```

7. **Name tables in the singular form**

Correct:

```
CREATE TABLE dbo.[Employee]
```

Incorrect:

```
CREATE TABLE dbo.[Employees]
```

8. Tables that map one-to many, many-to-many relationships should be named by concatenating the names of the tables in question, starting with the most central table's name.

Correct:

```
CREATE TABLE dbo.[EmployeeSalary]
```

Advantages of Implementing Coding Standards

1. **Enhanced Efficiency**
2. **Risk of project failure is reduced**
3. **Minimal Complexity**
4. **Easy to Maintain**
5. **Bug Rectification**
6. **A Comprehensive Look**
7. **Cost-Efficient**

SQL Comment

- Microsoft SQL Server supports two types of comments:
  - ✓ Line comment:

```
USE AdventureWorks2012
GO
-- First line of a multiple-line comment
-- Second line of a multiple-line comment
SELECT * FROM HumanResources.Employee
SELECT * FROM Person.Address -- Single line comment
GO
```

  - ✓ Block comment:

```
USE AdventureWorks2012
GO
/* First line of a multiple-line comment.
   Second line of a multiple-line comment. */
SELECT * FROM HumanResources.Employee
SELECT * FROM Person.Address /*Single line comment*/
GO
```

SQL Comment

- Always use comment to explain your code.
- Use natural/human language in comment to easy understand.
- All comments should be same format.
- Break comment line to avoid horizontal scroll bar.

Naming conventions

- Ensure the name is unique and does not exist as a reserved keyword.
- Names must begin with a letter and may not end with an underscore.
- Avoid abbreviations and if you have to use them make sure they are commonly understood.

SQL Coding Standard

Naming conventions

- Use a collective name or, a plural form for table names
- Never give a table the same name as one of its columns and vice versa
- Always use the singular name for columns.
- Use table/column aliases for easier reading

Format code

- Always use **UPPERCASE** for the reserved keywords like SELECT and WHERE.
- Break line to avoid horizontal scroll bar. It recommended that start line with **KEYWORD**

```
(SELECT b.species_name,
        AVG(b.height) AS average_height, AVG(b.diameter) AS average_diameter
   FROM botanic_garden_flora AS b
  WHERE b.species_name = 'Banksia'
     OR b.species_name = 'Sheoak'
     OR b.species_name = 'Wattle'
  GROUP BY b.species_name, b.observation_date);
```

Section 2

SQL BEST PRACTICES

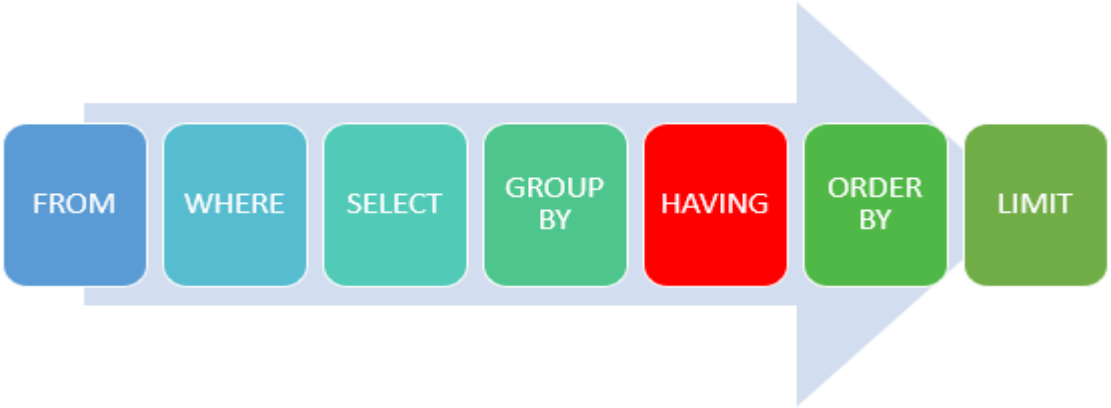
SQL Best Practices

1. Avoid select *, use column names

- Use Column Names Instead of * in a SELECT Statement
- **Original query:**
  - × SELECT * FROM Students;
- **Improved query:**
  - ✓ SELECT Name, ClassName FROM Students;

2. Avoid including a HAVING clause in SELECT statements

- The HAVING clause is used to filter the rows after all the rows are selected and it is used like a filter.
- It works by going through the final result table of the query parsing out the rows that don't meet the HAVING condition.



2. Avoid including a HAVING clause in SELECT statements

- Original query

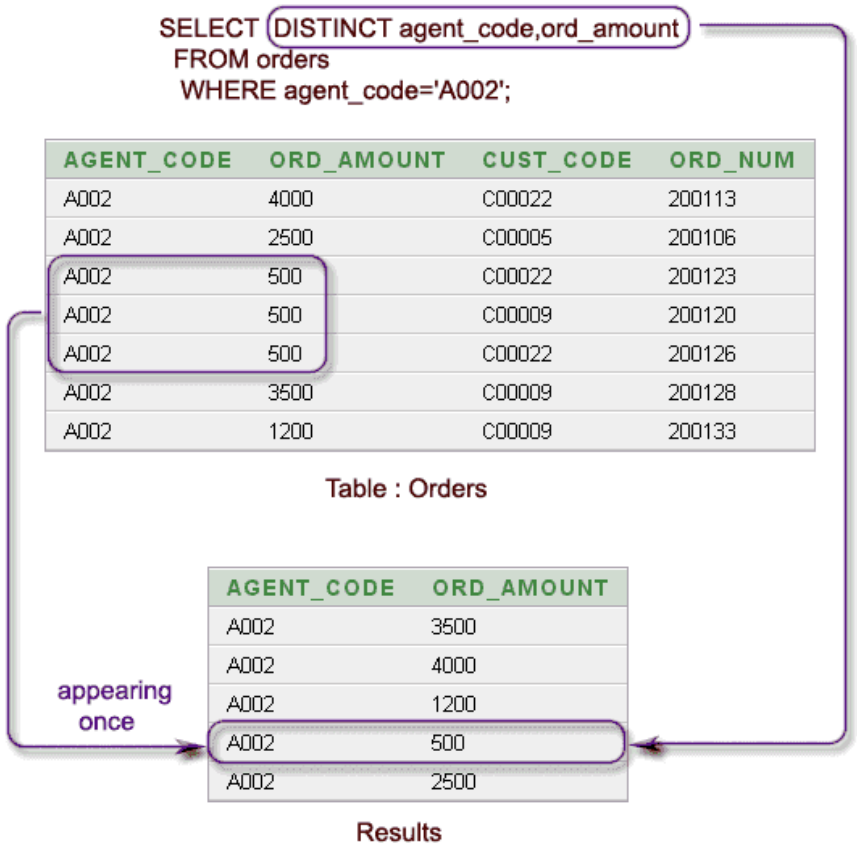
```
1  SELECT s.cust_id,count(s.cust_id)
2  FROM SH.sales s
3  GROUP BY s.cust_id
4  HAVING s.cust_id != '1660' AND s.cust_id != '2';
```

- Improved query

```
1  SELECT s.cust_id,count(cust_id)
2  FROM SH.sales s
3  WHERE s.cust_id != '1660'
4  AND s.cust_id !='2'
5  GROUP BY s.cust_id;
```

3. Eliminate Unnecessary DISTINCT Conditions

- The DISTINCT keyword in the original query is unnecessary because the table_name contains the primary key p.ID, which is part of the result set.



3. Eliminate Unnecessary DISTINCT Conditions

- Original query

```
1  SELECT DISTINCT * FROM SH.sales s
2  JOIN SH.customers c
3  ON s.cust_id= c.cust_id
4  WHERE c.cust_marital_status = 'single';
```

- Improved query

```
1  SELECT * FROM SH.sales s JOIN
2  SH.customers c
3  ON s.cust_id = c.cust_id
4  WHERE c.cust_marital_status='single';
```

4. Use UNION ALL instead of UNION

- The UNION ALL statement is faster than UNION
  - ✓ UNION ALL statement does not consider duplicate s, and
  - ✓ UNION statement does look for duplicates in a table while selection of rows, whether or not they exist.

5. Avoid using OR in join conditions

- Original query

```
1  SELECT *
2  FROM SH.costs c
3  INNER JOIN SH.products p
4  ON c.unit_price =  p.prod_min_price OR c.unit_price = p.prod_list_price;
```

- Improved query

```
1  SELECT *
2  FROM SH.costs c
3  INNER JOIN SH.products p ON c.unit_price = p.prod_min_price
4  UNION ALL
5  SELECT *
6  FROM SH.costs c
7  INNER JOIN SH.products p ON c.unit_price = p.prod_list_price;
```

6. Avoid any redundant mathematics

- Original query

```
1  SELECT *
2  FROM SH.sales s
3  WHERE s.cust_id + 10000 < 35000;
```

- Improved query

```
1  SELECT *
2  FROM SH.sales s
3  WHERE s.cust_id < 25000;
```

7. Consider using COUNT

- COUNT(1) gets converted into COUNT(*) by SQL Server, so there is no difference between these. The 1 is a literal, so a COUNT('whatever') is treated as equivalent.
- COUNT(column_name)
  - ✓ If the column_name definition is NOT NULL, this gets converted to COUNT(*).
  - ✓ If the column_name definition allows NULLs, then SQL Server needs to access the specific column to count the non-null values on the column.

7. Consider using COUNT

- Never use COUNT(*)
  - ✓ tt must read all columns and cause unnecessary reads.
- Always use COUNT(1)
  - ✓ The primary key is the first column in the table and you want it to read the clustered index.
- Always use COUNT(column_name)
  - ✓ You can select which index it will scan.

7. Consider using COUNT

- Original query

```
1  SELECT *
2  FROM SH.sales s
3  WHERE s.cust_id + 10000 < 35000;
```

- Improved query

```
1  SELECT *
2  FROM SH.sales s
3  WHERE s.cust_id < 25000;
```

Summary

Thank you