

**A
BOOK
ON
T-SQL**

1. WHAT IS DATA?

ANY RAW FACTS / INFORMATION. STORED IN FILES. txt, xls, pdf, doc..

2. WHAT IS DATABASE?

DATABASE MANAGEMENT SYSTEM (DBMS) FROM MICROSOFT. TO HOST & MANAGE DATABASES

3. WHY SQL SERVER?

FREE TECHNICAL SUPPORT

FREE UPDATES

Cheaper 24500 usd

Ease of Use & Integration

Polybase

Highly Scalable, Faster

FREE BI [BUSINESS INTELLIGENCE TOOLS] FOR ANALYSIS, FORECASTS, Rich Support to Web Technologies including Python, Node JS, Ruby, ADO etc

Complete support to Data Science Technologies including Azure, HDFS, etc

Ref: <https://academy.microsoft.com/en-us/professional-program/tracks/data-science/>

4. DEFINITION OF CLOUD:

A PLATFORM TO STORE APPLICATIONS, DATA, PROGRAMS, OS, ANY OTHER SOFTWARE REMOTELY

SUPPORTED PLATFORMS FOR SQL SERVER DATABASES IN CLOUD:

1. MICROSOFT AZURE AZURE.MICROSOFT.COM

2. GOOGLE CLOUD <https://cloud.google.com/sql-server/>

3. AMAZON CLOUD (AWS) <https://aws.amazon.com/rds/sqlserver/>

5. TYPES OF DATABASES WITH SQL SERVER?

1. DATABASES TO STORE REAL-TIME (CURRENT, ONGOING, ACTIVE) DATA

T-SQL = OLTP DATABASE: ONLINE TRANSACTION PROCESSING DATA EX: LIVE CRICKET MATCH

2. DATABASES TO STORE HISTORICAL (OLD, INACTIVE, PAST) DATA

SSIS = DWH : DATAWAREHOUSES EX: STORE PAST MATCH DATA

3. DATABASES TO STORE DATA FOR FASTER ANALYSIS AND ACCURATE FORECASTS

SSAS = OLAP : ONLINE ANALYTICAL PROCESSING DATA EX: REPORTS FOR ANALYSIS OF PAST

EX: REPORTS FOR PREDICTIONS

6: ROLE

TYPE OF DATABASE

SQL DEVELOPERS TO DESIGN OLTP DATABASES USING T-SQL LANGUAGE (T-SQL: TRANSACT SQL)

MSBI DEVELOPERS TO DESIGN OLAP DATABASES USING MDX, XMLA, DMX, DAX LANGUAGES

TO DESIGN DWH DATABASES USING ETL TOOLS AND T-SQL

SQL DBAs TO MANAGE OLTP DATABASES

7. WHAT IS SQL?

STRUCTURED QUERY LANGUAGE ::: COMMON TO ANY DATABASE.

A PLATFORM TO COMMUNICATE WITH ANY DATABASE.

8. WHAT IS T-SQL?

TRANSACT SQL :: SPECIFIC TO SQL SERVER DATABASE OPERATIONS.

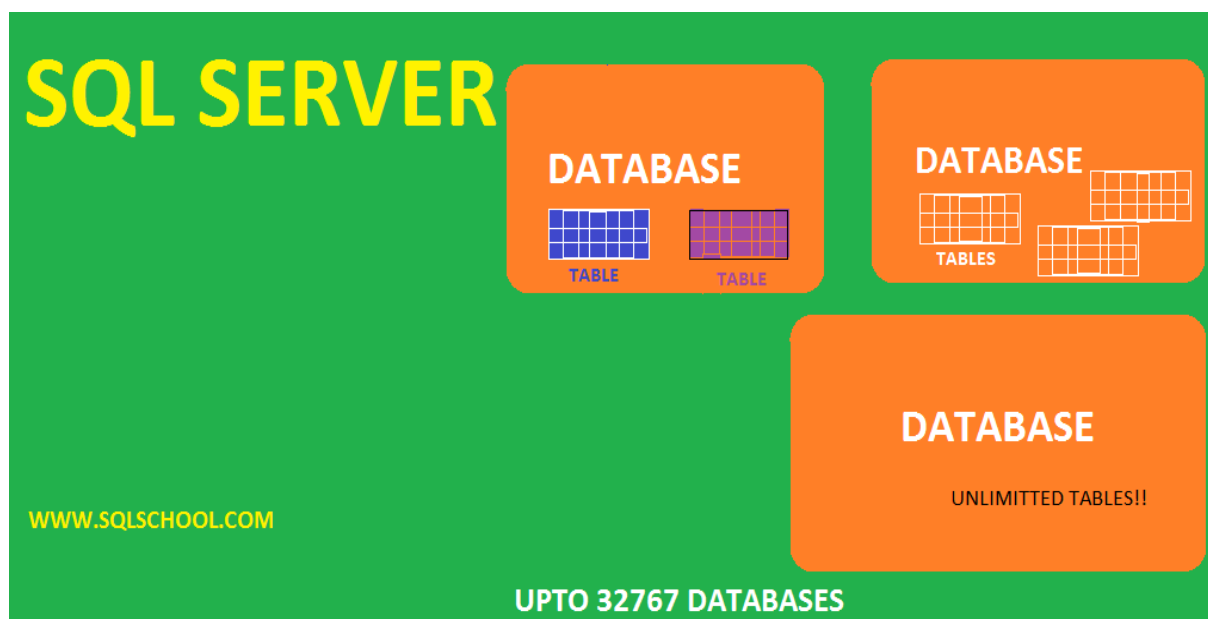
PLATFORM TO COMMUNICATE WITH SQL SERVER DATABASES.

9. WHAT ARE RESPONSIBILITIES FOR SQL DEVELOPER ?

To plan, design, code/program databases-TSQL

10. WHAT ARE RESPONSIBILITIES FOR SQL ADMINISTRATOR (SQL DBA)?

To manage, monitor, repair databases-T-SQL + Power Shell



WE CAN CREATE UPTO 32767 DATABASES IN SQL SERVER.

INSTALLATION OF SQL SERVER

SQLSERVER HAS TWO PARTS:

1. SERVER (SQL SERVER OR INSTANCE)
2. CLIENT (SQL SERVER MANAGEMNT STUDIO(SSMS) OR SQL OPERATIONS STUDIO).

WE HAVE TO INSTALL BOTH THE PARTS.

1. HOW TO INSTALL SQL SERVER?

DOWNLOAD THE SETUP >> WE SEE A COLLECTION OF FILES AND DIRECTORIES >> DOUBLE CLICK @ SETUP APPLICATION FILE (EXE) >> GO TO INSTALLATION PAGE >> NEW INSTALLATION LINK (2:1)>> SELECT "EVALUATION" EDITION >ACCEPT LICENSING TERMS >> SELECT ALL FEATURES (DATABASE ENGINE, ANALYSIS, REPORTING, INTEGATION, R) >UNCHECK POLYBASE >> SPECIFY INSTANCE NAME >> SPECIFY "ADD CURRENT USER" AS ADMINISTRATOR >> INSTALL.

2. CLIENT TOOLS INSTALLATION

a. HOW TO INSTALL SSMS (SQL SERVER MANAGEMENT STUDIO) TOOL?

Applicable for Windows OS

Software Download Link: <http://sqlschool.com/downloads>

DOWNLOAD THE SETUP FILE >> WE SEE ONE FILE WITH "ENU" [SSMS-SETUP-ENU] >> DOUBLE CLICK >>INSTALL >> ACCEPT LICENSING TERMS >> INSTALL

CLICK "YES" IF THERE IS ANY PROMPT TO CONTINUE THE INSTALLATION

b. HOW TO INSTALL SOS (SQL Operations Studio) TOOL?

Applicable for LINUX/MAC/UBUNTU OS

<https://docs.microsoft.com/en-us/sql/sql-operations-studio/download>

DOWNLOAD THE SETUP FILE >> double click the file >> start installation.

**** SERVER & TOOLS CAN BE INSTALLED IN PARALLEL. AT SAME TIME. THIS IS CALLED "LIGHT WEIGHT INSTALLATION"**

3. HOW TO CONNECT TO ABOVE SERVER USING THE SSMS TOOL?

WINDOWS >> APPS >> SEARCH >> "SSMS" >> FIND SQL SERVER MANAGEMENT STUDIO >

CLICK THE ENTRY TO LAUNCH THE APPLICATION.

THEN WE SEE A PROMPT FOR PROVIDING THE SQL SERVER NAME.

SERVER NAME > DROP DOWN > BROWSE FOR MORE > EXPAND 1ST ENTRY : DATABASE ENGINE

SELECT 1ST SERVER >> OK >> CONNECT.

* INSTANCE : A SUCCESSFUL INSTALLATION OF SQL SERVER

TYPES OF INSTANCES:

DEFAULT INSTANCE:

INSTANCE NAME = MSSQLSERVER

USED IN REAL-TIME

DEFAULT PORT: 1433

SERVER NAME = COMPUTERNAME

NAMED INSTANCE(S):

INSTANCE NAME IS USER DEFINED DURING INSTALLATION

USED IN DEV, TEST ENVIRONMENTS

PORT IS CONFIGURED AUTOMATICALLY

SERVER NAME = COMPUTERNAME\INSTANCENAME

NOTE: WE CAN INSTALL UPTO 50 INSTANCES PER OPERATING SYSTEM

** IF YOU DO NOT SEE ANY SERVER NAME IN "BROWSE" SECTION IT MEANS YOU DID NOT INSTALL THE SERVER. YOU INSTALLED ONLY TOOLS.

===TO TEST: USE "SQL SERVER CONFIGURATION MANAGER"

4.WHAT IS SERVICE?

A COMMUNICATION PATH BETWEEN WINDOWS AND SQL SERVER

EXAMPLE:MAIN GATE

5. WHAT IS SERVICE ACCOUNT ?

WINDOWS USERS TO CONTROL THE SERVICE

EXAMPLE:WATCHMEN

6. WHAT IS AUTHENTICATION?

A COMUNICATION PATH BETWEEN SQL SERVER AND END USERS

EXAMPLE: MAIN DOOR

7. WHAT ARE LOGINS?

WINDOWS / NON WINDOWS USERS TO CONNECT AND OPERATE ON THE SERVER EXAMPLE: SECURITY.

DURING INSTALLATION, ONE LOGIN ="sa" IS AUTO CREATED.

sa MEANS SYSTEM ADMINISTRATOR.

8. WHAT IS COLLATION?

A SERVER LEVEL PROPERTY USED TO CONTROL LANGUAGE SETTINGS.

EX: CI CASE INSENSITIVE - TO STORE USER NAMES

EX: CS CASE SENSITIVE - TO STORE PASSWORDS

9. WHAT IS A FILESTREAM?

AN SQL SERVER FEATURE TO STORE BLOB (BINARY LARGE OBJECTS) DATA.

10. WHAT ARE SYSTEM DATABASES?

DURING SQL SERVER INSTALLATION A SET OF FIVE DATABASES ARE AUTO CREATED.

THESE DEFAULT, AUTOCREATED DATABASES ARE CALLED SYSTEM DATABASES.

1. MASTER :USED TO CONTROL USER CONNECTIONS (LOGINS) & QUERIES IN USER DATABASES.
2. MODEL :USED AS A TEMPLATE FOR NEW DATABASES WE CREATE MANUALLY IN THE SERVER
3. MSBD :USED BY SQL DB ADMINISTRATORS FOR REPAIRS, JOBS, MONITORING..
4. TEMPDB :USED FOR QUERY TUNING (MAKING QUERIES RUN FASTER) AND DATA REPORTING.
5. RESOURCE :USED TO CONTROL SQL SERVER INSTANCE AND INTERNAL CONFIGURATIONS.

IN ADDITION TO ABOVE AUTO CREATED DATABASES IN SQL SERVER INSTANCE WE CAN CREATE UPTO 32,767 DATABASES. PER EACH INSTANCE (SERVER).

CLIENT -SERVER ARCHITECTURE

HOW SERVER & CLIENT COMMUNICATES EACH OTHER?

HERE: SERVER MEANS INSTANCE (WE INSTALLED THE INSTANCE)

CLIENT MEANS SSMS TOOL (EITHER AUTO INSTALLED OR MANUAL INSTALL).

CLIENT (SSMS) RECEIVES "SQL QUERIES" & "USER INTERFACE ACTIONS".

CLIENT UNDERSTANDS THEM AND DEFINE "TDS PACKETS"-TABULAR DATA STREAM PACKETS

THESE PACKETS ARE SENT OVER THE NETWORK TO THE SERVER.

SERVER UNDERSTAND THE TDS PACKETS CONTENT. "COMPILE" THE QUERIES AND "EXECUTE" THE QUERIES.

RESULT IS GIVEN BACK TO SSMS CLIENT IN THE FORM OF TDS PACKETS.

**** COMPILATION:** PROCESS OF CONVERTING QUERIES / USER ACTIONS TO MACHINE LEVEL CODE.

SQL TO BINARY CODE.

**** SQL : STRUCTURED QUERY LANGUAGE**

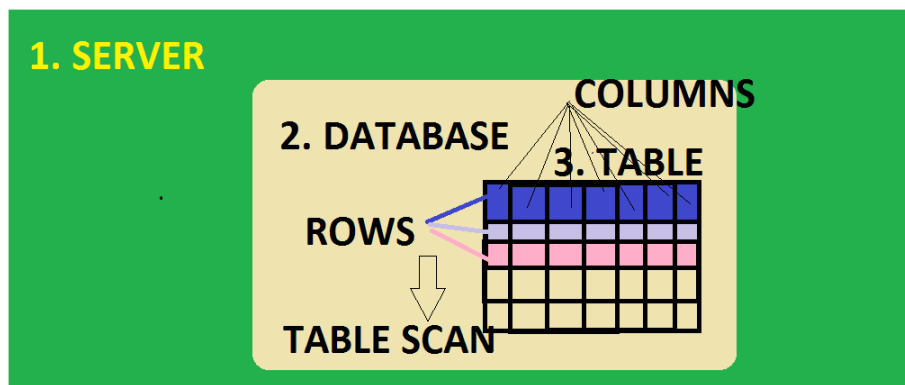
USED TO SPECIFY OPERATIONS TO DATABASES HOSTED INSIDE SQL SERVER.

**** PROVIDERS (DRIVERS) ARE PROGRAMS RESPONSIBLE TO DRIVE THE TDS PACKETS BETWEEN THE CLIENT AND THE SERVER.**

SQL NATIVE CLIENT IS THE DEFAULT PROVIDER/DRIVER

HOW TO VERIFY THE TDS PACKETS?

RIGHTCLICK ON QUERY >> INCLUDE CLIENT STATISTICS



-- QUERY 1: HOW TO CREATE NEW DATABASE?

```
CREATE DATABASE RETAIL_BIZ_DATABASE
```

-- QUERY 2: HOW TO CONNECT / USE THE DATABASE?

```
USE RETAIL_BIZ_DATABASE
```

-- QUERY 3: HOW TO CREATE TABLE FOR DATA STORAGE?

```
CREATE TABLE TBLPRODUCTS
```

```
(
```

```
ProductID    INTEGER, -- THIS COLUMN ACCEPT DIGITS
```

```
Name         CHARACTER(8000), -- THIS COLUMN ACCEPT UPTO 80 DIGITS/SYMBOLS/ALPHABETS
```

```
OrderYear    INT,  -- THIS COLUMN ACCEPT DIGITS
```

Size CHAR -- THIS COLUMN ACCEPT UPTO 1 DIGIT/SYMBOL/ALPHABET

)

-- QUERY 4: HOW TO INSERT/STORE DATA INTO ABOVE TABLE [AUTO SAVE]

INSERT INTO TBLPRODUCTS VALUES (1001, 'CAPS', 2001, 'S')
INSERT TBLPRODUCTS VALUES ('1002', 'CAPS', 2001, 'M')

INSERT TBLPRODUCTS VALUES ('1003', 'BALLS', 2001, 'S'),

(1004, 'BALLS', 2002, 'M'),

('1005', 'BALLS', 2003, 'L')

-- SINGLE QUOTE MANDATORY FOR ALPHABETS & SYMBOLS

-- SINGLE QUOTE OPTIONAL FOR INTEGERS/DIGITS

-- SINGLE QUOTE IS USED PROTECT THE FORMAT OF DATA. EX: -VE & POSITIVE VALUES

-- QUERY 5: HOW TO VERIFY THE ABOVE INSERTED DATA ?

SELECT * FROM TBLPRODUCTS

-- ORDER OF INSERTION = ORDER OF STORAGE = ORDER OF RETREIVAL

-- "TABLE SCAN"

-- * MEANS TO REPORT all COLUMNS

-- QUERY 6: HOW TO REPORT ALL PRODUCTS WITH SIZE = S?

SELECT * FROM TBLPRODUCTS

WHERE

SIZE = 'S'

-- QUERY 7: HOW TO REPORT ALL PRODUCTS BETWEEN 1002, 1004?

SELECT * FROM TBLPRODUCTS

WHERE

ProductID BETWEEN 1002 AND 1004

-- QUERY 8: HOW TO REPORT ALL PRODUCTS WITH IDs 1002 OR 1004?

-- optional conditions

SELECT * FROM TBLPRODUCTS

WHERE

ProductID = 1002 OR ProductID = 1004

-- QUERY 9: HOW TO REPORT ALL SMALL PRODUCTS WITH IDs ABOVE 1002?

-- mandatory conditions

SELECT * FROM TBLPRODUCTS

WHERE ProductID > 1002 AND SIZE = 'S'

-- QUERY 10: HOW TO REPORT ALL PRODUCTS EXCEPT SMALL SIZE?

SELECT * FROM TBLPRODUCTS

WHERE SIZE != 'S'

-- QUERY 11: HOW TO MODIFY ALREADY INSERTED TABLE DATA?

-- CASE 1: BELOW UPDATE APPLICABLE FOR SPECIFIC PRODUCT

UPDATE TBLPRODUCTS

SET OrderYear = 2010

WHERE

ProductID = 1001

--CASE 2: BELOW UPDATE APPLICABLE FOR ALL PRODUCTS

UPDATE TBLPRODUCTS

SET OrderYear = 2010

-- QUERY 12: HOW TO REMOVE DATA FROM A TABLE?

DELETE FROM TBLPRODUCTS

WHERE ProductID = 1001

-- QUERY 13: MODIFY THE STRUCTURE / DEFINITION OF TABLE?

ALTER TABLE TBLPRODUCTS ADD PROD_PRICE INT

-- QUERY 14: HOW TO DROP COLUMN IN A TABLE?

ALTER TABLE TBLPRODUCTS DROP COLUMN PROD_PRICE

-- QUERY 15: HOW TO REMOVE TABLE FROM DATABASE?

DROP TABLE TBLPRODUCTS

TASKS FOR ADDITIONAL PRACTICE?

1. HOW TO REPORT ALL PRODUCTS WITH SIZE = S OR SIZE = M

SELECT * FROM TBLPRODUCTS WHERE SIZE='S'OR SIZE='M'

2. HOW TO REPORT ALL PRODUCTS WITH SIZES EXCEPT S OR M

SELECT * FROM TBLPRODUCTS WHERE SIZE!='S'OR SIZE!='M'

3. HOW TO REPORT ALL PRODUCTS WITH ORDER YEAR EITHER 2001 OR 2002 ?

SELECT * FROM TBLPRODUCTS WHERE OrderYear=2001 OR OrderYear=2002

4. HOW TO REPORT ALL PRODUCTS WITH ORDER YEAR NEITHER 2001 NOR 2002 ?

SELECT * FROM TBLPRODUCTS WHERE OrderYear!=2001 AND OrderYear!=2002

5. HOW TO REPORT ALL MEDIUM SIZE PRODUCTS WITH SALE YEAR (ORDER YEAR) ABOVE 2002?

SELECT * FROM TBLPRODUCTS WHERE SIZE='M'AND OrderYear>2002

6. HOW TO REPORT PRODUCT IDs AND PROUDCT NAMEs WITH SALE YEAR (ORDER YEAR) ABOVE 2002?

SELECT ProductID,NAME,OrderYear FROM TBLPRODUCTS where OrderYear>2002

7. HOW TO REPORT PRODUCT IDs AND PROUDCT NAMEs WITH SALE YEAR (ORDER YEAR) ABOVE 2002 OR 2003

SELECT ProductID,NAME,OrderYear FROM TBLPRODUCTS where OrderYear>2002 or Orderyear >2003

8. HOW TO REPORT PRODUCT IDs AND PROUDCT NAMEs WITH SALE YEAR (ORDER YEAR) ABOVE 2003 OR BELOW 2002?

SELECT ProductID,NAME,OrderYear FROM TBLPRODUCTS where OrderYear>2003 or Orderyear <2002

CREATE:THIS STATEMENT IS USED TO DEFINE / CREATE NEW STRUCTURES IN DATABASE

**INSERT:THIS STATEMENT IS USED TO STORE DATA INTO TABLE
WE CAN STORE ONE ROW AT A TIME OR MULTIPLE ROWS AT A TIME.**

ROW MEANS A COLLECTION OF COLUMN VALUES.

EX: (1001, 'CAPS', 2001, 'S')

SELECT :THIS STATEMENT IS USED TO REPORT DATA

"*" IS USED TO REPORT ALL COLUMNS IN THE TABLE

"WHERE" IS USED TO SPECIFY CONDITIONS TO FILTER THE ROWS

EX: SELECT * FROM TABLE WHERE COL1 > 10

SQL SERVER DATA TYPES

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,770,000	9,223,372,036,854,770,000
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	1E+38	10 ³⁸ -1
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38
datetime	Jan 1, 1753	31-Dec-99
smalldatetime	1-Jan-00	6-Jun-79
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	
char	Maximum length of 8,000 characters.	
varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).	
varchar(max)	Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).	
text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.	
sysname	The sysname data type is used for table columns, variables, and stored procedure parameters that store object names.	
xml	Stores xml formatted data.maximum2 GB	

cursor	Stores a reference to a cursor used for database operations	
uniqueidentifier	Stores a globally unique identifier(guid)	
image	To store images. binary format.	
binary(n)	To store media information, binary data	
table	To store table data (array of values)	
varbinary(max)	variable width binary string,maxi 2gb	
nvarchar	variable width binary string,max 4000char	
money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807'	
sql_variant	stores upto 8,000 characters	

-- QUERY 1: HOW TO CREATE DATABASE?

CREATE DATABASE EMPLOYEE_DB

-- QUERY 2: HOW TO CONNECT TO ABOVE DATABASE?

USE EMPLOYEE_DB

-- QUERY 3: HOW TO CREATE NEW TABLE?

CREATE TABLE EMPLOYEE_INFO

(

EMP_ID INT,

EMP_FNAME VARCHAR(15), -- STORES UPTO 15 CHARACTERS
 -- REQUIRES 1 BYTE OF DISK SPACE FOR EACH CHARACTER
 -- reserves upto 15 Bytes of space to store data

EMP_LNAME CHAR(15), -- STORES UPTO 15 CHARACTERS
 -- REQUIRES 1 BYTE OF DISK SPACE FOR EACH CHARACTER
 -- reserves EXACTLY 15 Bytes of space to store data

EMP_CNTRY VARCHAR(30),

EMP_SAL BIGINT -- TO STORE BIGGER INTEGER VALUES

)

-- QUERY 4: HOW TO INSERT DATA INTO ABOVE TABLE?

-- GO STATEMENT IS USED TO DIVIDE SQL SCRIPTS INTO "BATCHES"

-- MEANS, A SET OF SQL STATEMENTS BEFORE EACH go IS A BATCH

-- EACH BATCH IS SUBMITTED TO THE SERVER IN A TDS PACKET

-- TABULAR DATA STREAM PACKET. DEFAULT TDS PACKET SIZE = 4096 BYTES (4 KB)

-- go + 1 = batch = tds packets

INSERT INTO EMPLOYEE_INFO VALUES(1001,'SHEKAR','G','USA',72000)

INSERT INTO EMPLOYEE_INFO VALUES(1002,'MUNI','S','INDIA',32000)

GO

```
INSERT INTO EMPLOYEE_INFO VALUES(1003,'MUNI SHEKAR','S','CANADA',92000)
```

```
INSERT INTO EMPLOYEE_INFO VALUES(1004,'MUNISH','N','CANADA',152000)
```

```
INSERT INTO EMPLOYEE_INFO VALUES(1005,'JOHN','H','INDIA',125000)
```

```
GO
```

```
INSERT INTO EMPLOYEE_INFO VALUES(1006,'JOHNY','J','INDIA',152000)
```

```
INSERT INTO EMPLOYEE_INFO VALUES(1007,'JEFF','K','USA',12500)
```

```
-- QUERY 5: HOW TO REPORT / TEST ABOVE ABOVE TABLE DATA?
```

```
SELECT * FROM Employee_INFO
```

```
-- QUERY 6: HOW TO REPORT THE TABLE DATA FOR FEW COLUMNS?
```

```
SELECT EMP_ID, EMP_FNAME, EMP_LNAME FROM Employee_INFO
```

```
-- QUERY 7: HOW TO REPORT THE TABLE DATA WITH MODIFIED COLUMN NAMES?
```

```
-- "AS" MEANS ALIAS. MEANS, A TEMPORARY NAME GIVEN TO A COLUMN/TABLE/QUERY
```

```
-- ALIAS IS APPLICABLE WITHIN THAT STATEMENT ONLY.
```

```
SELECT EMP_ID AS ID, EMP_SAL AS SALARY, EMP_FNAME AS NAME FROM  
Employee_INFO
```

```
-- QUERY 8: HOW TO APPLY CONDITIONS ON ABOVE TABLE DATA?
```

```
SELECT * FROM Employee_INFO
```

```
WHERE EMP_ID > 1001 AND Emp_CNTRY = 'CANADA'
```

```
-- QUERY 9:
```

```
SELECT * FROM Employee_INFO
```

```
WHERE EMP_ID = 1005 OR EMP_ID = 1006 -- COMPARISION IN SEQUENCE
```

```
-- QUERY 10:
```

```
SELECT * FROM Employee_INFO
```

```
WHERE EMP_ID IN (1005, 1006) -- COMPARISION IN PARALLEL
```

```
-- QUERY 11:
```

```
SELECT * FROM Employee_INFO WHERE EMP_ID NOT IN (1005, 1006)
```

```
-- QUERY 12: HOW TO REPORT LIST OF EMPLOYEES WITH MAXIMUM SALARY?
```

```
SELECT MAX(EMP_SAL) FROM Employee_INFO
```

```
SELECT MAX(EMP_SAL) AS EMP_MAX_SAL FROM Employee_INFO
```

-- QUERY 13:

```
SELECT * FROM Employee_INFO
```

```
WHERE Emp_Sal = (SELECT MAX(EMP_SAL) FROM Employee_INFO )
```

-- QUERY 14: HOW TO REPORT LIST OF EMPLOYEES WITH MINIMUM SALARY?

```
SELECT * FROM Employee_INFO
```

```
WHERE Emp_Sal = (SELECT MIN(EMP_SAL) FROM Employee_INFO )
```

-- QUERY 15: HOW TO REPORT LIST OF ALL EMPLOYEES WITH BOTH MINIMUM & MAXIMUM SALARIES?

```
SELECT *FROM Employee_INFO
```

```
WHERE Emp_Sal = (SELECT MIN(EMP_SAL) FROM Employee_INFO )
```

```
UNION ALL
```

```
SELECT * FROM Employee_Info
```

```
WHERE Emp_Sal = (SELECT MAX(EMP_SAL) FROM Employee_INFO )
```

DIFFERENCE BETWEEN UNION AND UNION ALL?

*UNION FOLLOWS TABLESCAN ORDER

*UNION ALL FOLLOWS EXECUTION ORDER.

UNION

The UNION command is used to select related information from two tables, much like the JOIN command. However, when using the UNION command all selected columns need to be of the same data type. With UNION, only distinct values are selected.

UNION ALL

The UNION ALL command is equal to the UNION command, except that UNION ALL selects all values.

The difference between UNION and UNION ALL is that UNIONALL will not eliminate duplicate rows, instead it just pulls all rows from all tables fitting your query specifics and combines them into a table.

A UNION statement effectively does a SELECT DISTINCT on the results set. If you know that all the records returned are unique from your union, use UNION ALL instead, it gives faster results.

TO VERIFY WHICH ORDER IT EXECUTES?

INSERT A NEW ROW IN THE TABLE.

EXECUTE THE QUERY WITH UNION AND UNION ALL

-- **QUERY 16: EXAMPLE FOR LOCAL TEMPORARY TABLES?**

CREATE TABLE #TEMP_TABLE

(COL1 INT,COL2 INT,COL3 INT)

INSERT INTO #TEMP_TABLE VALUES (1001, 1,1), (1002,2,2), (1003, 3,3)

-- **QUERY 17: EXAMPLE FOR GLOBAL TEMPORARY TABLES ?**

CREATE TABLE ##TEMP_TABLE

(COL1 INT,COL2 INT,COL3 INT)

INSERT INTO ##TEMP_TABLE VALUES (1001, 1,1), (1002,2,2), (1003, 3,3)

SINGLE '#' REPRESENTS LOCAL

DOUBLE '##' REPRESENTS GLOBAL

DATABASE DESIGN

FILES & FILEGROUPS

FOR END USERS: DATABASE IS LOGICALLY A COLLECTION OF TABLES.

FOR BACKEND (DISK): DATABASE IS PHYSICALLY A COLLECTION OF FILES.

DATA FILES : TO STORE DATABASE PROPERTIES (METADATA) + ACTUAL DATA.
metadata = name, size, location, users...

LOG FILES : TO STORE AUDIT OPERATIONS. WHO PERFORM WHAT OPERATION, WHEN?

TYPES OF DATA FILES:

MDF 1. PRIMARY DATA FILE : THIS IS THE VERY FIRST DATA FILE OF THE DATABASE CONTAINS DATABASE LEVEL PROPERTIES. IT CAN ALSO CONTAIN ACTUAL TABLE DATA.

NDF 2. SECONDARY DATA FILE(S): THESE ARE THE ADDITIONAL FILES ADDED DURING/AFTER DATABASE CREATION.

THESE FILES CONTAIN ACTUAL TABLE DATA only

EACH DATA FILE IS SUB DIVIDED INTO "PAGES". SIZE OF EACH PAGE = 8 KB.

A COLLECTION OF 8 PAGES IS CALLED AN "EXTENT".

ASSUME DATA FILE IS YOUR CLASSROOM. BOYS & GIRLS.

TYPES OF EXTENTS:

1. UNIFORM EXTENTS : ALL PAGES OF EXTENT CONTAIN SINGLE TABLE DATA = DEDICATED EXTENTS

2. MIXED EXTENTS : EXTENT CONTAINS DATA FOR DIFFERENT TABLES.

LOG FILES / TRANSACTION LOG FILES / TLOG FILES:

ldf USED TO STORE AUDIT INFORMATION LIKE:WHAT OPERATION? WHEN? STATUS?

THESE LOG FILES ARE USED FOR DATA RECOVERY AND FOR DATABASE MONITORING.

EACH LOG FILE IS SUB DIVIDED INTO "PAGES". SIZE OF EACH PAGE = 8 KB.

A COLLECTION OF FEW LOG PAGES IS CALLED "VIRTUAL LOG FILE" (VLF)

START LOCATION OF EVERY VLF IS CALLED "MINI LSN". LSN = LOG SEQUENCE NUMBER.

FILEGROUPS

FOR EASY DATA ALLOCATION OF TABLES, THE DATA FILES ARE GROUPED INTO "FILEGROUPS"

A FILEGROUP CONTAINS ONE OR MORE DATA FILES.

LOG FILES DO NOT BELONG TO ANY FILEGROUP.

FILEGROUPS ARE A BRIDGE BETWEEN PHYSICAL FILES AND LOGICAL TABLES.

THIS MEANS, FILEGROUPS ARE LINKED TO TABLES.

HOW TO MAKE A TABLE READ ONLY?

NOW, TO MAKE A TABLE READONLY WE NEED TO MAKE THE RELEVANT FILEGROUP READONLY.

WHAT ARE SCHEMAS?

IF A DATABASE CONTAINS MORE NUMBER OF TABLES WE CAN GROUP THEM INTO "SCHEMAS".

EACH SCHEMA CAN CONTAIN ANY NUMBER OF TABLES. FOR EASY USER ACCESS.

PURPOSE OF FILEGROUPS :

FOR FASTER TABLE ACCESS. EASY DATA RECOVERY= BACKEND

PURPOSE OF SCHEMAS :

FOR EASY TABLE ACCESS. EASY SECURITY MANAGMENT = END USER

SYNOPSIS:

DATABASE IS PHYSICALLY A COLLECTION OF FILES. THESE FILES ARE DATA FILES AND LOG FILES

DATA FILES ARE GROUPED INTO FILEGROUPS.

FILEGROUPS ARE USED TO CONTROL TABLE STORAGE

A BIG DATA FILE CAN CONTAIN DATA FOR MULTIPLE TABLES

A BIG TABLE CAN SPAN ACROSS MULTIPLE FILES.

PURPOSE OF MULTIPLE DATA FILES:

EASY DATA RECOVERY. EASY DATA ALLOCATION. BETTER PERFORMANCE

PURPOSE OF FILEGROUPS:

TO BRIDGE THE TABLES AND FILES. EASY TABLE MANAGEMENT (READ ONLY OPTION)

PURPOSE OF LOG FILES:

TO AUDIT / MONITOR THE OPERATIONS ON THE DATABASE

PURPOSE OF MULTIPLE LOG FILES:

EASY AUDIT AND MONITORING. IF ONE LOG FILE IS TOO BUSY TO AUDIT NEW TRANSACTIONS THEN OTHER LOG FILE CAN PERFORM THE AUDITS.

SIMILARITY FOR EASY UNDERSTANDING:

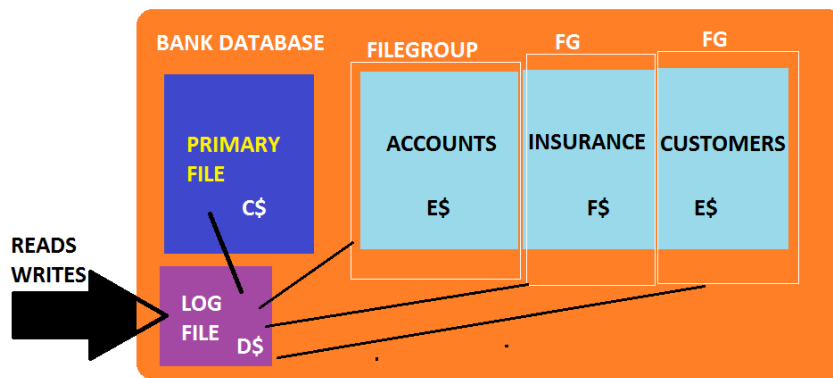
DATABASE--APARTMENT

PRIMARY FILE GROUP--GROUND FLOOR

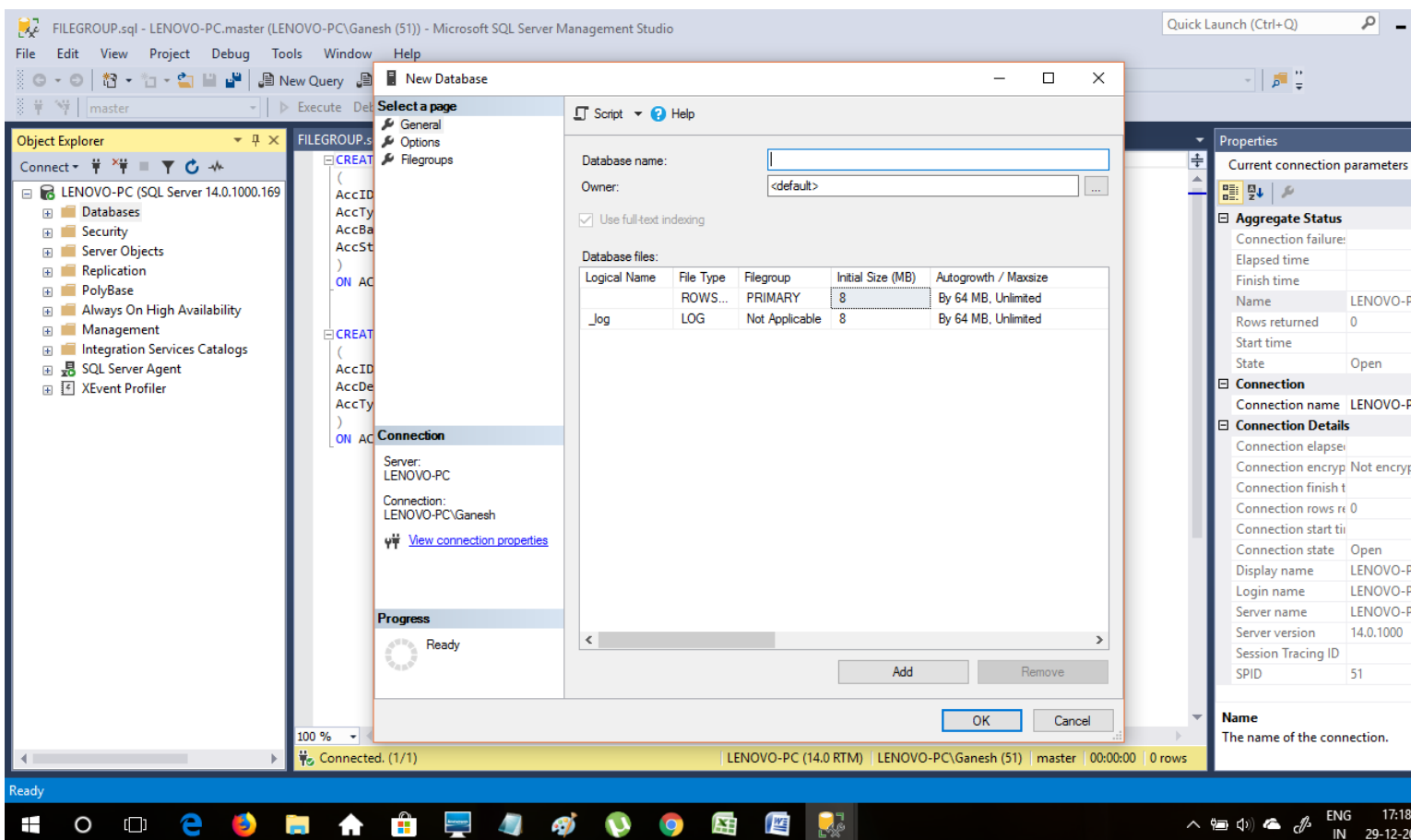
PRIMARY FILE-- ASSOCIATION OFFICE

SECONDARY FILES--ADDITIONAL FLATS

LOG FILES--WATCH MEN



USING GUI CREATE DATABASE AND FILE GROUPS.



HERE GIVE ANY NAME AS DATABASE NAME.

HERE DATABASE NAME=LOGICAL NAME.

FILE TYPE:

ROWS DATA IF IT IS PRIMARY OR SECONDARY FILE

WE CANT CHANGE FILETYPE OF PRIMARY FILEGROUP.

LOG IF IT IS A LOG FILE.

BY DEFAULT ONE PRIMARY FILE GROUP IS CREATED.

INITIAL SIZE AND AUTOGROWTH OF FILES IS ASSIGNED , ALSO YOU CAN CUSTOMIZE IT WITH SOME RESTRICTIONS.

PATH: YOU CAN CHANGE THE PATH OF PRIMARY FILE.

YOU CAN ADD SECONDARY FILES BY CLICKING ON ADD BUTTON

SECONDARY FILE GROUP CAN BE OF ANY FILE TYPE :ROWSDATA,LOG,FILESTREAM.

EXAMPLE:

```
CREATE TABLE tblAccounts
(
  AccID int,
  AccType varchar(30),
  AccBal float,
  AccSts bit
)
ON ACCOUNTS_FILEGROUP
```

Q1. LOGICALLY (FOR END USERS):

DATABASE IS A COLLECTION OF _____

Q2. PHYSICALLY (AT BACKEND, DISK LEVEL):

DATABASE IS A COLLECTION OF _____

Q3. A COLLECTION OF FILES IS CALLED _____

Q4. TO BRIDGE TABLES AND FILES WE NEED TO DEFINE _____

Q5. RECOMMENDED EXTENSION FOR PRIMARY DATA FILE IS _____

Q6. A COLLECTION OF DATA PAGES IS CALLED _____

Q7. WHAT IS THE PURPOSE OF MASTER DATABASE : _____

Q8. WHAT IS THE PURPOSE OF RESOURCE DATABASE : _____

CHECKPOINT : THE PROCESS OF WRITING DATA FROM LOG FILE TO DATA FILE.

WRITE AHEAD LOG (WAL) :

This thread is responsible to write every operation to database log file.

If one log file is full, then database operations are written to another log file (if available). If not, existing log file is TRUNCATED to get space for writing the log information.

LAZY WRITER :

This thread is responsible to write every operation (DML, DDL, SELECT) to LOG FILE and then to DATA FILE. Controlled by master database.

**** THREAD MEANS A BACKGROUND PROCESS/ACTIVITY.**

**** TRUNCATION MEANS ERASING / DELETING THE CONTENTS**

CONSTRAINTS:CONDITIONS SPECIFIED ON TABLE COLUMNS.

EX: PRD_PRICE >= 1 EX: AGE <= 99

"NULL" MEANS AN UNKNOWN VALUE. OR AN UNDEFINED VALUE.

TYPES OF CONSTRAINTS:

NOT NULL :THIS COLUMN DOES NOT ALLOW NULLs. MEANS, MANDATORY COLUMN VALUE

NULL :THIS COLUMN ALLOW NULLs. MEANS, COLUMN VALUE IS OPTIONAL.

UNIQUE :THIS COLUMN DOES NOT ALLOW DUPLICATES. ALLOWS UPTO 1 NULL VALUE

PRIMARY KEY :THIS COLUMN DOES NOT ALLOW DUPLICATES & DOES NOT ALLOW NULL VALUES

UNIQUE KEY + NOT NULL=PRIMARY KEY

FOREIGN KEY (REFERENCE):THIS IS USED TO reference / LINK ONE TABLE TO ANOTHER TABLE IN THE DATABASE.

SUCH DATABASES ARE CALLED "RELATIONAL DATABASES"

EX: PRODUCTS TABLE LINKED TO SALE ORDERS TABLE

EX: STUDENTS TABLE LINKED TO COURSE TABLE

EX: ORDERS TABLE LINKED TO MENU TABLE

CHECK CONSTRAINT :USED TO SPECIFY CONDITIONS ON COLUMN VALUES

EX: AGE BETWEEN 18 AND 99

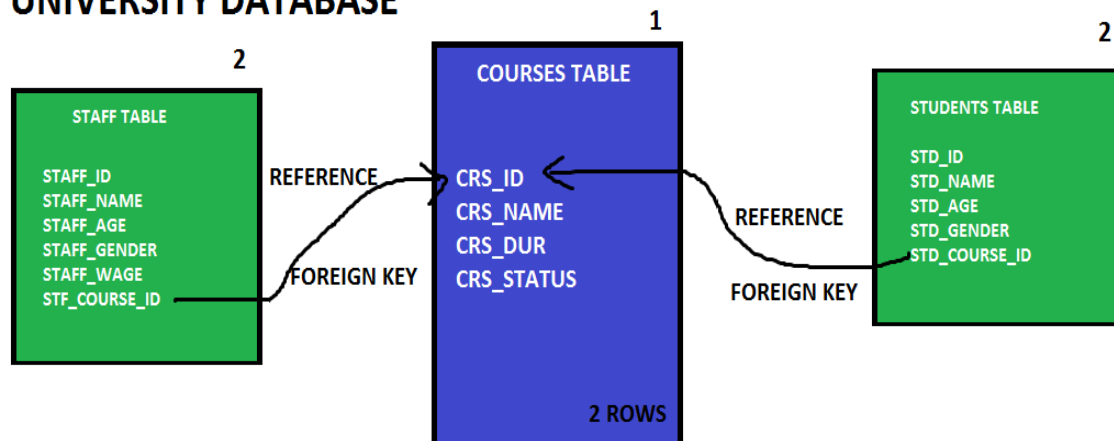
DEFAULT CONSTRAINT:USED TO INSERT A DEFAULT VALUE.

EX: GENDER = F

IDENTITY PROPERTY : USED TO INSERT AUTOMATED SEQUENCE OF INTEGER VALUES (EX: 1001,1002,1003....)

SQL SERVER

UNIVERSITY DATABASE



```
CREATE DATABASE UNIVDATABASE
```

```
USE UNIVDATABASE
```

```
-- REQUIREMENT: TO CREATE THREE TABLES:
```

```
-- ONE FOR COURSES                                BLUE
-- ONE FOR STUDENTS. THIS TABLE SHOULD BE RELATED TO COURSES
-- GREEN
-- ONE FOR STAFF. THIS TABLE SHOULD BE REALTED TO COURSES    GREEN
```

```
-- TABLE 1:
```

```
CREATE TABLE COURSES
```

```
(
  COURSE_ID INT PRIMARY KEY ,
  -- UNIQUE + NOT NULL = NO DUPLICATES + NO NULL VALUES
  COURSE_NAME VARCHAR(30) NOT NULL,
  -- MEANS THIS COLUMN DOES NOT ALLOW NULL VALUES = M
  COURSE_DUR SMALLINT CHECK (COURSE_DUR IN (180,240))
  -- MEANS THIS COLUMN ACCEPTS ONLY 180 OR 240
)
```

```
INSERT COURSES VALUES ( 101, 'COMPUTERS', 180 )
INSERT COURSES VALUES ( 102, 'CIVIL', 180 )
INSERT COURSES VALUES ( 103, 'ROBOTICS', 180 )
```

```
/****** HOW TO TEST ABOVE CONSTRAINTS *****/
```

```
-- TEST FOR PRIMARY KEY:
```

```
INSERT COURSES VALUES ( 101, 'COURSE4', 180 )    -- ERROR
```

```
-- TEST FOR NOT NULL:
INSERT COURSES VALUES ( 101, NULL, 180 )          -- ERROR
```

```
-- TEST FOR CHECK CONSTRAINT:
INSERT COURSES VALUES ( 1002, 'COURSE4', -180 ) -- ERROR
```

-- HOW TO REPORT ABOVE TABLE DATA?

```
SELECT * FROM COURSES
```

-- TABLE 2:

```
CREATE TABLE tblStudents          -- ORDER
(
  StdId INT          UNIQUE,
  -- THIS COLUMN DOES NOT ALLOW DUPLICATES. CAN ALLOW UPTO 1 NULL VALUE.
  StdName  VARCHAR(30) NOT NULL,
  StdAge   TINYINT CHECK ( STDAGE >= 18 ),
  StdGender CHAR    CHECK ( STDGENDER IN ( 'M','F' )),
  StdCourse_ID INT REFERENCES COURSES(COURSE_ID)
  -- THIS IS AN ADDITIONAL REFERENCE COLUMN IN TABLE
)                                     -- MENU
```

```
INSERT INTO tblStudents VALUES ( 100001, 'SAI', 34, 'M', 101 )
INSERT INTO tblStudents VALUES ( 100002, 'SAISH', 33, 'M', 101 )
INSERT INTO tblStudents VALUES ( 100003, 'JOHN', 34, 'M', 102 )
INSERT INTO tblStudents VALUES ( 100004, 'JOHNY', 33, 'M', 102 )
```

-- HOW TO REPORT ABOVE TABLE DATA?

```
SELECT * FROM tblStudents
```

-- HOW TO TEST FOREIGN KEY?

```
INSERT INTO tblStudents VALUES (100005, 'SAISHA', 39, 'F', 6666)
-- INSERT FAILS. INVALID COURSE ID
INSERT INTO tblStudents VALUES ( 100005, 'JENY', 24, 'F', 9999)
-- ERROR DUE TO INVALID COURSE ID
```

```
-- HOW TO TEST FOREIGN KEY?
INSERT INTO tblStudents VALUES (100005, 'SAISHA', 39, 'F', 6666) -- INSERT FAILS. INVALID COURSE ID
INSERT INTO tblStudents VALUES ( 100005, 'JENY', 24, 'F', 9999) -- ERROR DUE TO INVALID COURSE ID
```

Msg 547, Level 16, State 0, Line 65
The INSERT statement conflicted with the FOREIGN KEY constraint "FK__tblStuden__StdCo__15502E78".
The statement has been terminated.

```

-- TABLE 3:

CREATE TABLE tblStaff

(
StfId INT IDENTITY (1001, 1) PRIMARY KEY,
-- IDENTITY (SEED, INCREMENT)
StfName      VARCHAR(30) NOT NULL,
StfAge       TINYINT CHECK ( STFAGE >= 28 ) DEFAULT 28,
StfGender CHAR    CHECK ( STFGENDER IN ( 'M','F' )) DEFAULT 'F',
StfCourse_ID INT   REFERENCES COURSES(COURSE_ID)
)

-- NOTE: IDENTITY MEANS, AUTOMATING THE INSERTION OF SEQUENCE VALUES.
INSERT INTO tblStaff VALUES ( 'SAI', 34, 'M', 101 )
INSERT INTO tblStaff VALUES ( 'SAISH', 33, 'M', 102 )
INSERT INTO tblStaff VALUES ( 'JENY', 34, 'F', 102)
SELECT * FROM tblStaff

INSERT INTO tblStaff VALUES ( 'JENY2', 14, 'F', 102)
INSERT INTO tblStaff VALUES ( 'JENY2', 44, 'F', 102)

-- HOW TO TEST FOREIGN KEY CONSTRAINT?
INSERT INTO tblStaff VALUES ( 'JENYSH', 44, 'M', 222)

-- ERROR DUE TO INVALID COURSE ID

SELECT * FROM COURSES
SELECT * FROM tblStudents
SELECT * FROM tblStaff

```

```
-- TABLE #4:

CREATE TABLE tblExams
(
    ExmID int                IDENTITY PRIMARY KEY,
    ExmCrS int                REFERENCES COURSES(COURSE_ID),
    Duration tinyint          check (Duration in (180,240)),
    ExmDate datetime          default getdate()
)

INSERT INTO tblExams(ExmCrS,Duration) VALUES (101, 180)
INSERT INTO tblExams(ExmCrS,Duration) VALUES (102, 180)
INSERT INTO tblExams(ExmCrS,Duration) VALUES (102, 220)      --
ERROR

INSERT INTO tblExams(ExmCrS,Duration) VALUES (102, 180)

SELECT * FROM tblExams

TRUNCATE TABLE tblExams    -- THIS IS TO REMOVE TABLE DATA

INSERT INTO tblExams VALUES (101, 240,'2017-12-05')

SELECT * FROM tblExams

/*
```

SELF ASSESSMENT FROM YOUR CAREFUL PRACTICE OF ABOVE EXAMPLES:

1. WHAT IS THE ISSUE WITH IDENTITY PROPERTY?

POSSIBLE MISSING SEQUENCE.

WHENEVER THERE IS AN INSERT STATEMENT EXECUTION, IDENTITY IS AUTO GENERATED EVENTHOUGH THE INSERT MAY FAIL.

2. CAN A FOREIGN KEY CONSTRAINT ALLOW DUPLICATE VALUES?

YES

3. CAN A FOREIGN KEY CONSTRAINT ALLOW NULL VALUES?

YES

4. OUT OF PRIMARY KEY AND NULL/NOT NULL, WHICH ONE TAKES MORE PRIORITY?

NULL/NOT NULL

5. WHAT IS THE DIFFERENCE BETWEEN PRIMARY KEY CONSTRAINT AND UNIQUE KEY CONSTRAINT?

PRIMARY KEY : NO DUPLICATES AND NO NULL VALUES

UNIQUE : UPTO 1 NULL VALUE

6. WHAT IS THE DIFFERENCE BETWEEN DELETE AND TRUNCATE?

DELETE IS USED TO REMOVE TABLE DATA

DELETE IS LOGGED IN DATABASE LOG FILE. CAN INCLUDE CONDITIONS.

TRUNCATE IS ALSO USED TO REMOVE TABLE DATA.

TRUNCATE DOES NOT GET LOGGED IN DATABASE LOG FILE. DOES NOT INCLUDE CONDITIONS.

***/-- ADDITIONAL PRACTICE EXAMPLES:**

CREATE TABLE TEST_TABLE (COL1 INT, COL2 INT)

HOW TO ADD PRIMARY KEY TO EXISTING TABLE?

ALTER TABLE TEST_TABLE ALTER COLUMN COL1 INT NOT NULL

ALTER TABLE TEST_TABLE ADD CONSTRAINT PK_TEST_COL PRIMARY KEY(COL1,COL2)
-- ERROR

ALTER TABLE TEST_TABLE ALTER COLUMN COL2 INT NOT NULL

ALTER TABLE TEST_TABLE ADD CONSTRAINT PK_TEST_COL PRIMARY KEY(COL1,COL2)

-- HOW TO ADD FOREIGN KEY TO EXISTING TABLE?

ALTER TABLE TEST_TABLE ADD CONSTRAINT FK_TEST_COL FOREIGN KEY(COL2)
REFERENCES COURSES(COURSE_ID)

GROUPBY

GROUP BY : USED TO IDENTIFY UNIQUE VALUES OF THE GIVEN COLUMN(S) AND PERFORM AGGREGATIONS

AGGREGATIONS LIKE: SUM, AVG, COUNT, MIN, MAX ...

HAVING : USED TO SPECIFY CONDITION ON GROUP BY DATA

ROLLUP : USED TO GENERATE NEW ROWS IN THE OUTPUT THAT CONTAINS VALUE LEVEL AGREGATIONS, IN GROUP BY.

EX: CLASS WISE COUNT + total class count

CLASS WISE COLOR WISE COUNT + total class count

CUBE : USED TO GENERATE NEW ROWS IN THE OUTPUT THAT CONTAINS VALUE LEVEL AGGREGATIONS

FOR ALL POSSIBLE GROUP BY COLUMNS

EX: CLASS WISE COUNT

COLOR WISE COUNT

CLASS WISE COLOR WISE COUNT

COLOR WISE CLASS WISE COUNT

GROUPING : USED TO IDENTIFY THE NEW ROWS RESULTED FROM ABOVE GROUP BY & ROLLUP.

SCENARIO 1: YOU WERE GIVEN A DATABASE WITH SALES BASED ON TIME ENTRIES LIKE YEAR, QUARTER, MONTH.

YOU NEED TO REPORT SALES ?

ANSWER: ROLLUP

SCENARIO 2: YOU WERE GIVEN A DATABASE WITH SALES BASED ON PRODUCT PROPERTIES LIKE COLOR, CLASS.

YOU NEED TO REPORT SALES.

ANSWER: CUBE

EXAMPLE:

```
CREATE TABLE tblPopulation (  
Country VARCHAR(100),  
[State] VARCHAR(100),  
City VARCHAR(100),  
Population INT  
)
```

```
INSERT INTO tblPopulation VALUES('COUNTRY1', 'STATE1','CITY1',9 )
```

```
INSERT INTO tblPopulation VALUES('COUNTRY1', 'STATE1','CITY2',8 )
```

```
INSERT INTO tblPopulation VALUES('COUNTRY1', 'STATE2','CITY1',5.5)
```

```
INSERT INTO tblPopulation VALUES('COUNTRY1', 'STATE2','CITY2',7.5)
```

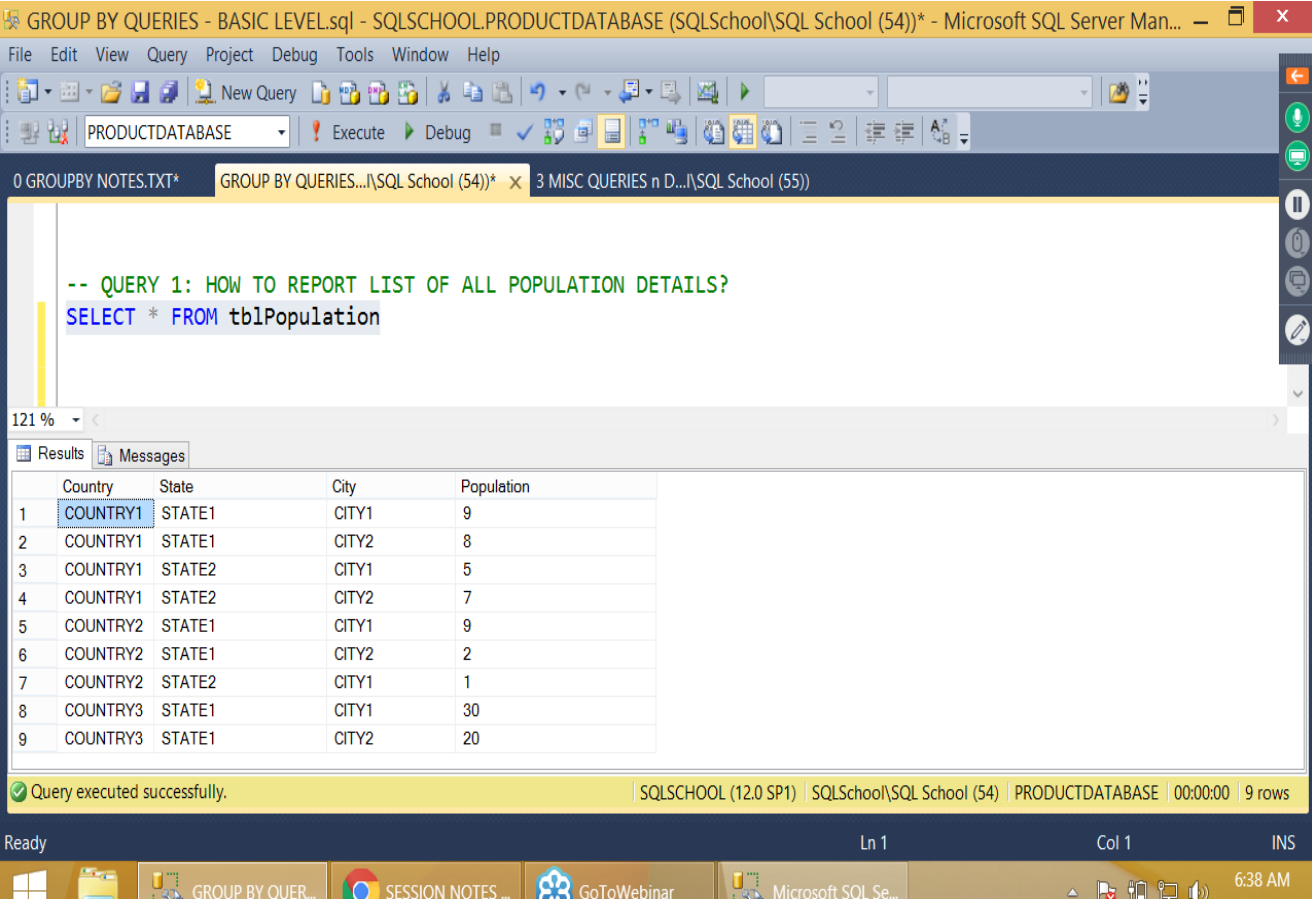
```

INSERT INTO tblPopulation VALUES('COUNTRY2', 'STATE1','CITY1',9.5)
INSERT INTO tblPopulation VALUES('COUNTRY2', 'STATE1','CITY2',2.5)
INSERT INTO tblPopulation VALUES('COUNTRY2', 'STATE2','CITY1',1.5)
INSERT INTO tblPopulation VALUES('COUNTRY3', 'STATE1','CITY1',30)
INSERT INTO tblPopulation VALUES('COUNTRY3', 'STATE1','CITY2',20)

```

-- QUERY 1: HOW TO REPORT LIST OF ALL POPULATION DETAILS?

```
SELECT * FROM tblPopulation
```



GROUP BY QUERIES - BASIC LEVEL.sql - SQLSCHOOL.PRODUCTDATABASE (SQLSchool\SQL School (54))* - Microsoft SQL Server Man...

File Edit View Query Project Debug Tools Window Help

PRODUCTDATABASE Execute Debug

0 GROUPBY NOTES.TXT* GROUP BY QUERIES... \SQL School (54))* 3 MISC QUERIES n D... \SQL School (55)

```
-- QUERY 1: HOW TO REPORT LIST OF ALL POPULATION DETAILS?
SELECT * FROM tblPopulation
```

121 %

Results Messages

	Country	State	City	Population
1	COUNTRY1	STATE1	CITY1	9
2	COUNTRY1	STATE1	CITY2	8
3	COUNTRY1	STATE2	CITY1	5
4	COUNTRY1	STATE2	CITY2	7
5	COUNTRY2	STATE1	CITY1	9
6	COUNTRY2	STATE1	CITY2	2
7	COUNTRY2	STATE2	CITY1	1
8	COUNTRY3	STATE1	CITY1	30
9	COUNTRY3	STATE1	CITY2	20

Query executed successfully. | SQLSCHOOL (12.0 SP1) | SQLSchool\SQL School (54) | PRODUCTDATABASE | 00:00:00 | 9 rows

Ready Ln 1 Col 1 INS

GROUP BY QUER... SESSION NOTES ... GoToWebinar Microsoft SQL Se...

6:38 AM 12/16/2017

-- QUERY 2: HOW TO REPORT LIST OF ALL UNIQUE COUNTRIES DETAILS?

```
SELECT DISTINCT Country FROM tblPopulation
```

-- QUERY 1: HOW TO REPORT LIST OF ALL POPULATION DETAILS?

SELECT * FROM tblPopulation

-- QUERY 2: HOW TO REPORT LIST OF ALL UNIQUE COUNTRIES DETAILS?

SELECT DISTINCT Country FROM tblPopulation

SELECT Country FROM tblPopulation
group by Country

121 %

Results Messages

	Country
1	COUNTRY1
2	COUNTRY2
3	COUNTRY3

SELECT Country FROM tblPopulation

group by Country

THE ABOVE BOTH QUERIES GIVES SAME RESULT.

GROUPBY INTERNALLY PERFORMS DISTINCT SORT.

The screenshot displays the Microsoft SQL Server Management Studio interface. The main window shows a SQL script with three queries. Query 1 is a simple SELECT statement. Query 2 uses DISTINCT. Query 3 uses GROUP BY. The execution plan for Query 3 is visible, showing a Table Scan, a Sort operation (Distinct Sort), and a SELECT statement. The status bar at the bottom indicates the query was executed successfully.

GROUP BY QUERIES - BASIC LEVEL.sql - LENOVO-PC.master (LENOVO-PC\Ganesh (51)) - Microsoft SQL Server Management Studio

File Edit View Query Project Debug Tools Window Help

master | Execute | Debug | Generic Debugger

GROUP BY QUERIES...O-PC\Ganesh (51)

```
INSERT INTO tblPopulation VALUES('COUNTRY2', 'STATE1', 'CITY2', 2.5)
INSERT INTO tblPopulation VALUES('COUNTRY2', 'STATE2', 'CITY1', 1.5)
INSERT INTO tblPopulation VALUES('COUNTRY3', 'STATE1', 'CITY1', 30)
INSERT INTO tblPopulation VALUES('COUNTRY3', 'STATE1', 'CITY2', 20)

-- QUERY 1: HOW TO REPORT LIST OF ALL POPULATION DETAILS?
SELECT * FROM tblPopulation

-- QUERY 2: HOW TO REPORT LIST OF ALL UNIQUE COUNTRIES DETAILS?
SELECT DISTINCT Country FROM tblPopulation

SELECT Country FROM tblPopulation
group by Country

-- QUERY 3: HOW TO REPORT COUNTRY WISE TOTAL POPULATION DETAILS?
```

100 %

Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT Country FROM tblPopulation group by Country

Execution plan:

- SELECT (Cost: 0 %)
- Sort (Distinct Sort) (Cost: 78 %)
- Table Scan [tblPopulation] (Cost: 22 %)

Query executed successfully. LENOVO-PC (14.0 RTM) LENOVO-PC\Ganesh (51) master | 00:00:00 | 0 rows

Properties

Current connection parameters

Aggregate Status

- Connection
- Elapsed time: 00:00:00.085
- Finish time: 30-12-2017 09:20
- Name: LENOVO-PC
- Rows return: 0
- Start time: 30-12-2017 09:20
- State: Open

Connection

- Connection: LENOVO-PC (LENOVO-PC\Ganesh (51))

Connection Details

- Connection: 00:00:00.085
- Connection: Not encrypted
- Connection: 30-12-2017 09:20
- Connection: 0
- Connection: 30-12-2017 09:20
- Connection: Open
- Display name: LENOVO-PC
- Login name: LENOVO-PC\Ganesh (51)
- Server name: LENOVO-PC
- Server version: 14.0.1000
- Session Trace
- SPID: 51

Name

The name of the connection.

Ready | Ln 25 | Col 1 | Ch 1 | INS

QUERY 3: HOW TO REPORT COUNTRY WISE TOTAL POPULATION DETAILS?

```
SELECT Country, sum(Population) AS TOTAL_POP FROM tblPopulation
```

```
GROUP BY Country
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor contains the following SQL code:

```
-- QUERY 3: HOW TO REPORT COUNTRY WISE TOTAL POPULATION DETAILS?
SELECT Country, sum(Population) AS TOTAL_POP FROM tblPopulation
GROUP BY Country

-- RULE TO USE GROUP BY:
-- WHENEVER WE USE "GROUP BY" THE COLUMNS USED IN SELECT STATEMENT SHOULD BE
-- EITHER IN GROUP BY OR IN AN AGGREGATE (SUM/AVG/MIN/MAX/...)FUNCTION.

SELECT Country, STATE, sum(Population) AS TOTAL_POP FROM tblPopulation
GROUP BY Country -- ERROR

SELECT Country, STATE, sum(Population) AS TOTAL_POP FROM tblPopulation
GROUP BY Country, STATE -- NO ERROR

-- QUERY 4: HOW TO REPORT COUNTRY WISE TOTAL POPULATION AND GRAND TOTAL ?
```

The query was executed, and the results are displayed in the Results pane:

	Country	TOTAL_POP
1	COUNTRY1	29
2	COUNTRY2	12
3	COUNTRY3	50

The status bar at the bottom indicates: "Query executed successfully. LENOVO-PC (14.0 RTM) | LENOVO-PC\Ganesh (51) | master | 00:00:00 | 3 rows". The Properties pane on the right shows connection details for the current connection.

*NOTE:

RULE TO USE GROUP BY:

WHENEVER WE USE "GROUP BY" THE COLUMNS USED IN SELECT STATEMENT SHOULD BE EITHER IN GROUP BY OR IN AN AGGREGATE (SUM/AVG/MIN/MAX/...)FUNCTION.

EXAMPLE:

```
SELECT Country, STATE, sum(Population) AS TOTAL_POP FROM tblPopulation
```

```
GROUP BY Country -- ERROR
```

THE ABOVE QUERY RESULTS IN ERROR. WHY BECAUSE COLUMN 'STATE' IS NOT USED IN EITHER IN GROUPBY OR IN AGGREGATE FUNCTIONS.

```
SELECT Country, STATE, sum(Population) AS TOTAL_POP FROM tblPopulation
```

```
GROUP BY Country,STATE -- NO ERROR
```

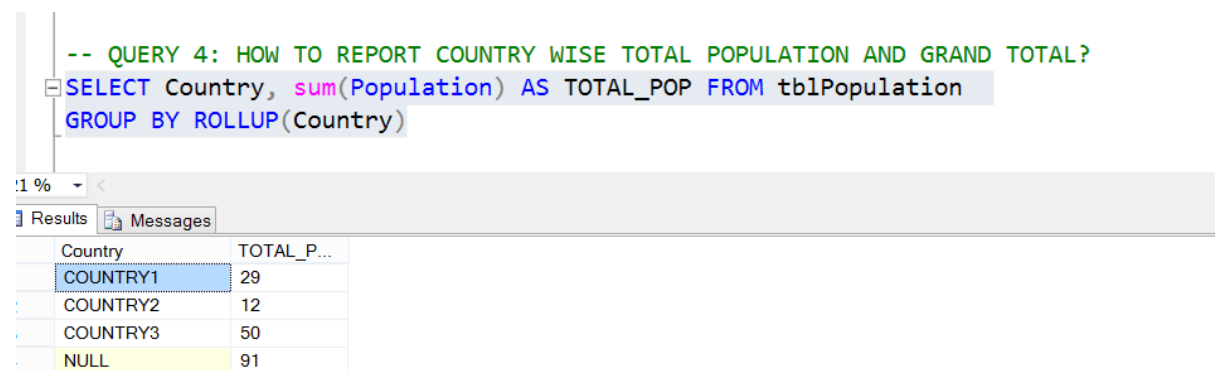
HERE 'Population' IS USED IN AGGREGATE FUNCTIONS.

QUERY 4: HOW TO REPORT COUNTRY WISE TOTAL POPULATION AND GRAND TOTAL?

```
SELECT Country, sum(Population) AS TOTAL_POP FROM tblPopulation
```

```
GROUP BY ROLLUP(Country)
```

OUTPUT:



The screenshot shows a SQL IDE interface. At the top, a query is entered in a text area: `-- QUERY 4: HOW TO REPORT COUNTRY WISE TOTAL POPULATION AND GRAND TOTAL?` followed by `SELECT Country, sum(Population) AS TOTAL_POP FROM tblPopulation` and `GROUP BY ROLLUP(Country)`. Below the text area, there are tabs for 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with two columns: 'Country' and 'TOTAL_P...'. The table contains four rows: 'COUNTRY1' with a value of 29, 'COUNTRY2' with 12, 'COUNTRY3' with 50, and 'NULL' with 91. The 'NULL' row is highlighted in yellow.

Country	TOTAL_P...
COUNTRY1	29
COUNTRY2	12
COUNTRY3	50
NULL	91

HERE THE LAST ROW WITH 'NULL' REPRESENTS TOTAL SUM OF ALL COUNTRIES,
MEANS ROLLUP GIVES INDIVIDUAL SUM AND GRAND TOTAL.

-

QUERY 5: HOW TO REPORT COUNTRY WISE, STATE WISE TOTAL POPULATION DETAILS?

```
SELECT Country, STATE, sum(Population) AS TOTAL_POP FROM tblPopulation
```

```
GROUP BY Country,STATE
```

QUERY 6: HOW TO REPORT COUNTRY WISE, STATE WISE TOTAL POPULATION AND GRAND TOTAL OF ALL COUNTRIES?

```
SELECT Country, STATE, sum(Population) AS TOTAL_POP FROM tblPopulation
```

```
GROUP BY ROLLUP(Country,STATE)
```

```
-- COUNTRY WISE TOTALS + GRAND TOTAL
```

QUERY 7: HOW TO REPORT COUNTRY, STATE AND ALL TOTAL POPULATIONS?

```
SELECT Country, STATE, sum(Population) AS TOTAL_POP FROM tblPopulation
```

```
GROUP BY CUBE(Country, STATE)
```

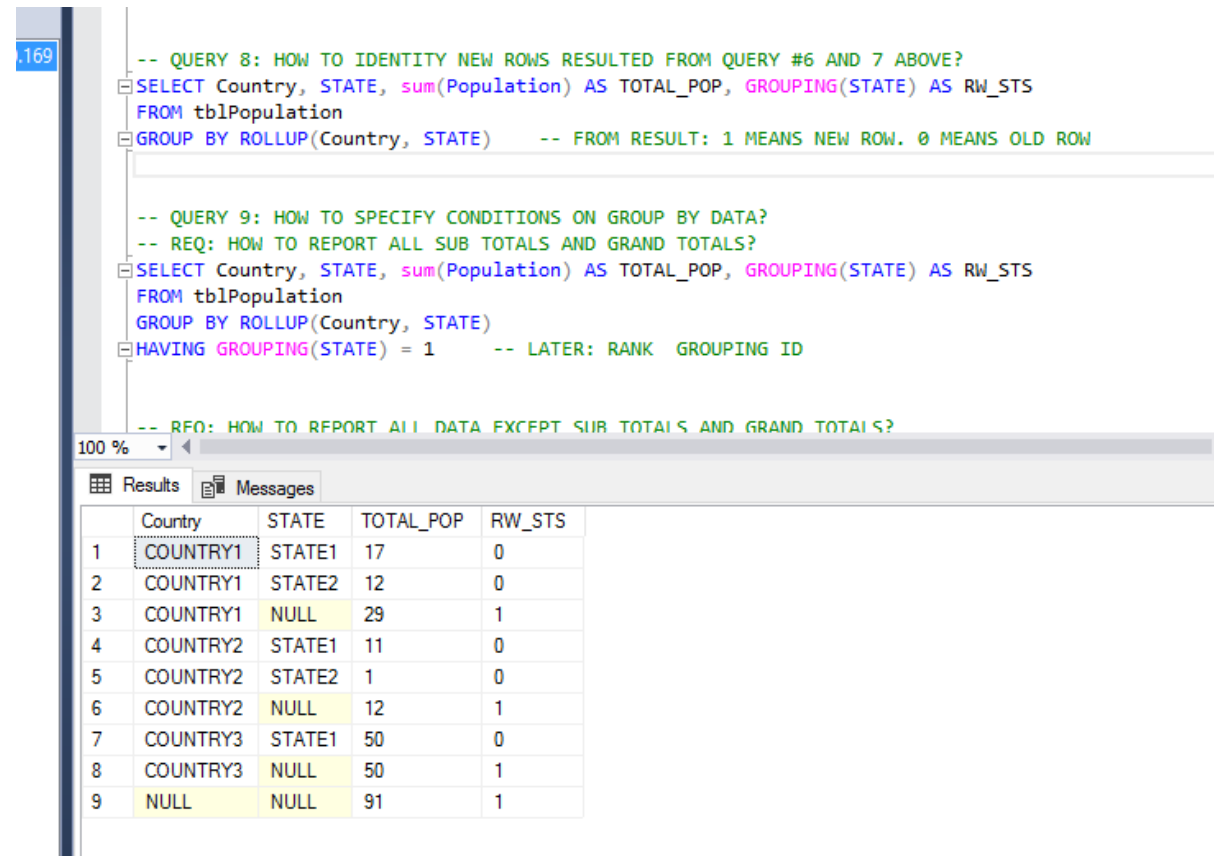
```
-- COUNTRY WISE TOTALS + STATE WISE TOTAL + GRAND TOTAL
```

QUERY 8: HOW TO IDENTITY NEW ROWS RESULTED FROM QUERY #6 AND 7 ABOVE?

```
SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS  
RW_STS FROM tblPopulation
```

```
GROUP BY ROLLUP(Country, STATE)
```

```
-- FROM RESULT: 1 MEANS NEW ROW. 0 MEANS OLD ROW
```



```
-- QUERY 8: HOW TO IDENTITY NEW ROWS RESULTED FROM QUERY #6 AND 7 ABOVE?  
SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS RW_STS  
FROM tblPopulation  
GROUP BY ROLLUP(Country, STATE) -- FROM RESULT: 1 MEANS NEW ROW. 0 MEANS OLD ROW
```

100 %

Results Messages

	Country	STATE	TOTAL_POP	RW_STS
1	COUNTRY1	STATE1	17	0
2	COUNTRY1	STATE2	12	0
3	COUNTRY1	NULL	29	1
4	COUNTRY2	STATE1	11	0
5	COUNTRY2	STATE2	1	0
6	COUNTRY2	NULL	12	1
7	COUNTRY3	STATE1	50	0
8	COUNTRY3	NULL	50	1
9	NULL	NULL	91	1

-- QUERY 9: HOW TO SPECIFY CONDITIONS ON GROUP BY DATA?

-- REQ: HOW TO REPORT ALL SUB TOTALS AND GRAND TOTALS?

```
SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS  
RW_STS
```

```
FROM tblPopulation
```

```
GROUP BY ROLLUP(Country, STATE)
```

```
HAVING GROUPING(STATE) = 1
```

```
-- LATER: RANK GROUPING ID
```

```
-- QUERY 9: HOW TO SPECIFY CONDITIONS ON GROUP BY DATA?
-- REQ: HOW TO REPORT ALL SUB TOTALS AND GRAND TOTALS?
SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS RW_STS
FROM tblPopulation
GROUP BY ROLLUP(Country, STATE)
HAVING GROUPING(STATE) = 1    -- LATER: RANK GROUPING ID

-- REQ: HOW TO REPORT ALL DATA EXCEPT SUB TOTALS AND GRAND TOTALS?
```

100 %

Results Messages

	Country	STATE	TOTAL_POP	RW_STS
1	COUNTRY1	NULL	29	1
2	COUNTRY2	NULL	12	1
3	COUNTRY3	NULL	50	1
4	NULL	NULL	91	1

REQ: HOW TO REPORT ALL DATA EXCEPT SUB TOTALS AND GRAND TOTALS?

```
SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS
RW_STS
```

```
FROM tblPopulation
```

```
GROUP BY ROLLUP(Country, STATE)
```

```
HAVING GROUPING(STATE) = 0
```

```
-- REQ: HOW TO REPORT ALL DATA EXCEPT SUB TOTALS AND GRAND TOTALS?
SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS RW_STS
FROM tblPopulation
GROUP BY ROLLUP(Country, STATE)
HAVING GROUPING(STATE) = 0
```

0 %

Results Messages

	Country	STATE	TOTAL_POP	RW_STS
1	COUNTRY1	NULL	29	1
2	COUNTRY2	NULL	12	1
3	COUNTRY3	NULL	50	1
4	NULL	NULL	91	1

-- REQ: HOW TO REPORT ALL BASE DATA + SUB TOTALS + GRAND TOTALS : IN ORDER?

CREATE VIEW VW_GROUPED_DATA

AS

SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS RW_STS

FROM tblPopulation

GROUP BY ROLLUP(Country, STATE)

HAVING GROUPING(STATE) = 0

UNION ALL

SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS RW_STS

FROM tblPopulation

GROUP BY ROLLUP(Country, STATE)

HAVING GROUPING(STATE) = 1

-- TO EXECUTE ABOVE VIEW:

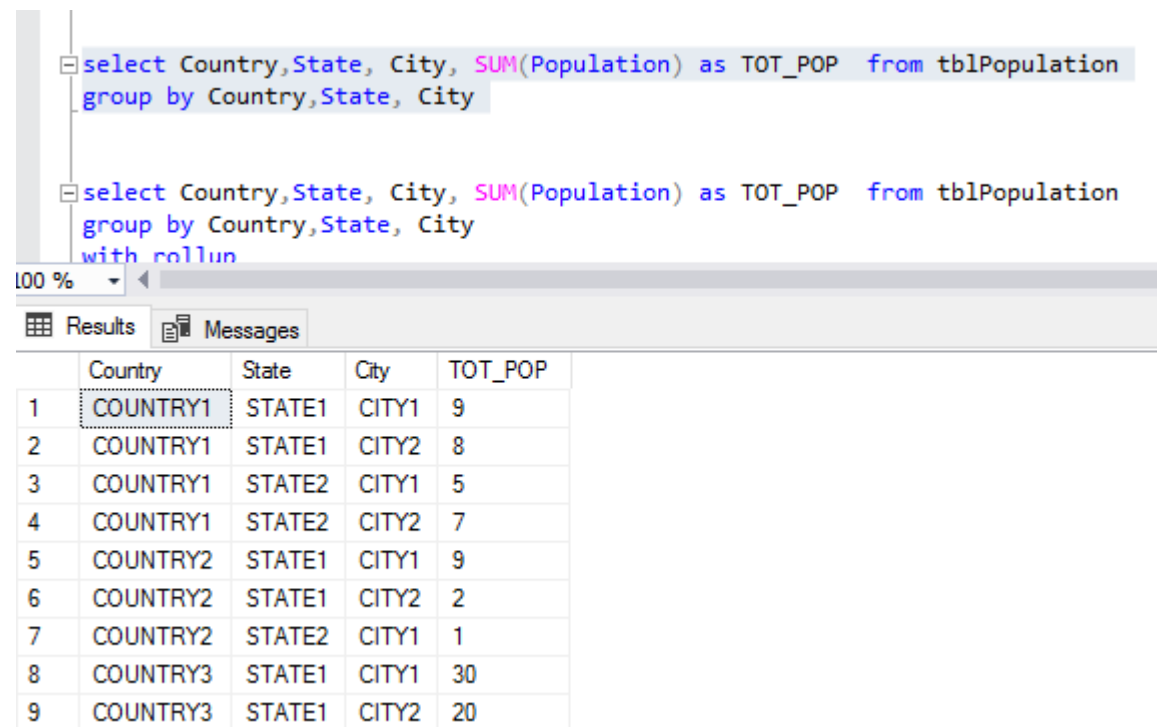
SELECT * FROM VW_GROUPED_DATA

* UNION ALL GIVES DUPLICATE VALUES.

```
GROUP BY QUERIES...O-PC\Ganesh (51))* -p X
-- REQ: HOW TO REPORT ALL BASE DATA + SUB TOTALS + GRAND TOTALS : IN ORDER?
CREATE VIEW VW_GROUPED_DATA
AS
SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS RW_STS
FROM tblPopulation
GROUP BY ROLLUP(Country, STATE)
HAVING GROUPING(STATE) = 0
UNION ALL
SELECT Country, STATE, sum(Population) AS TOTAL_POP, GROUPING(STATE) AS RW_STS
FROM tblPopulation
GROUP BY ROLLUP(Country, STATE)
HAVING GROUPING(STATE) = 1
-- TO EXECUTE ABOVE VIEW:
SELECT * FROM VW_GROUPED_DATA
```

	Country	STATE	TOTAL_POP	RW_STS
1	COUNTRY1	STATE1	17	0
2	COUNTRY1	STATE2	12	0
3	COUNTRY2	STATE1	11	0
4	COUNTRY2	STATE2	1	0
5	COUNTRY3	STATE1	50	0
6	COUNTRY1	NULL	29	1
7	COUNTRY2	NULL	12	1
8	COUNTRY3	NULL	50	1
9	NULL	NULL	91	1

```
select Country,State, City, SUM(Population) as TOT_POP from tblPopulation
group by Country,State, City
```



```
select Country,State, City, SUM(Population) as TOT_POP from tblPopulation
group by Country,State, City

select Country,State, City, SUM(Population) as TOT_POP from tblPopulation
group by Country,State, City
with rollup
```

	Country	State	City	TOT_POP
1	COUNTRY1	STATE1	CITY1	9
2	COUNTRY1	STATE1	CITY2	8
3	COUNTRY1	STATE2	CITY1	5
4	COUNTRY1	STATE2	CITY2	7
5	COUNTRY2	STATE1	CITY1	9
6	COUNTRY2	STATE1	CITY2	2
7	COUNTRY2	STATE2	CITY1	1
8	COUNTRY3	STATE1	CITY1	30
9	COUNTRY3	STATE1	CITY2	20

```
select Country,State, City, SUM(Population) as TOT_POP from tblPopulation
group by Country,State, City
with rollup
```

```
select Country,State, City, SUM(Population) as TOT_POP from tblPopulation
group by Country,State, City
with cube
```

```
select Country,State, City, SUM(Population) as TOT_POP from tblPopulation
group by rollup(Country,State, City) -- recommended
```

/*

GROUPING is an aggregate function that causes an additional column to be output with a value of 1 when the

row is added by either the CUBE or ROLLUP operator, or 0 when the row is not the result of CUBE or ROLLUP. */

```
select Country,State, City, SUM(Population) as TOT_POP, GROUPING(city) as 'Grouping' from tblPopulation
```

```
group by rollup(Country,State, City)
```

```
select Country,State, City, SUM(Population) as TOT_POP, GROUPING(city) as 'Grouping' from tblPopulation
group by rollup(Country,State, City)

select Country,State,
CASE GROUPING(City)
WHEN 1 THEN 'ALL CITIES'
ELSE isnull(City, 'Unknown') END ,SUM(Population) as TOT_POP from tblPopulation
group by rollup(Country,State, City)
```

) %				
Results Messages				
Country	State	City	TOT_POP	Grouping
COUNTRY1	STATE1	CITY1	9	0
COUNTRY1	STATE1	CITY2	8	0
COUNTRY1	STATE1	NULL	17	1
COUNTRY1	STATE2	CITY1	5	0
COUNTRY1	STATE2	CITY2	7	0
COUNTRY1	STATE2	NULL	12	1
COUNTRY1	NULL	NULL	29	1
COUNTRY2	STATE1	CITY1	9	0
COUNTRY2	STATE1	CITY2	2	0
0 COUNTRY2	STATE1	NULL	11	1

```
select Country,State,
```

```
CASE GROUPING(City)
```

```
WHEN 1 THEN 'ALL CITIES'
```

```
ELSE isnull(City, 'Unknown') END ,SUM(Population) as TOT_POP from
tblPopulation
```

```
group by rollup(Country,State, City)
```

```

select Country, State,
CASE GROUPING(City)
WHEN 1 THEN 'ALL CITIES'
ELSE isnull(City, 'Unknown') END, SUM(Population) as TOT_POP from tblPopulation
group by rollup(Country, State, City)

```

Country	State	(No column name)	TOT_POP
COUNTRY1	STATE1	CITY1	9
COUNTRY1	STATE1	CITY2	8
COUNTRY1	STATE1	ALL CITIES	17
COUNTRY1	STATE2	CITY1	5
COUNTRY1	STATE2	CITY2	7
COUNTRY1	STATE2	ALL CITIES	12
COUNTRY1	NULL	ALL CITIES	29
COUNTRY2	STATE1	CITY1	9
COUNTRY2	STATE1	CITY2	2
COUNTRY2	STATE1	ALL CITIES	11
COUNTRY2	STATE2	CITY1	1

-- CASE 1: YOU WERE GIVEN TIME DATA.

NEED TO WRITE QURIES TO REPORT SALES BASED ON TIME (YEAR, QTR, MONTH

** ROLLUP == YEARLY SALES, YEARLY QUATERLY SALES
 CUBE == YEARLY SALES, QUATERLY SALES, YEARLY
 QUATERLY SALES

-- CASE 2: YOU WERE GIVEN PRODUCT DATA.

NEED TO WRITE QURIES TO REPORT SALES BASED ON PRODUCT CLASS &

PRODUCT COLOR

** ROLLUP == CLASS SALES, COLOR SALES
 CUBE == CLASS SALES, COLOR SALES, CLASS & COLOR
 SALES

VIEWS

VIEWS ARE DATABASE OBJECTS USED TO STORE SELECT QUERY IN THE DATABASE. FOR REPEATED ACCESS. REUSABILITY OF CODE.

HOW TO STORE DATA IN THE DATABASE? TABLES

HOW TO STORE QUERIES IN THE DATABASE? VIEWS

PURPOSE OF VIEWS:

1. ROW LEVEL SECURITY

USING VIEWS, WE CAN IMPLEMENT SECURITY FOR INDIVIDUAL TABLE ROWS.

EXAMPLE: ASSUME A TABLE OF STUDENTS WITH MULTIPLE COURSES.

WE DESIGN A QUERY TO REPORT ALL STUDENTS OF A SPECIFIC COURSE (COMPUTERS) AND STORE THIS QUERY IN A "VIEW".

NOW, WHOEVER ACCESS THE VIEW WOULD SEE THE STUDENTS OF THAT SPECIFIC COURSE ONLY. NOT OTHER COURSE STUDENTS.

2. EASY ACCESS

END USERS NO NEED TO WRITE QUERIES. WE WRITE THE QUERIES ONCE

AND STORE THEM IN THE DATABASE. FOR EASY, FUTURE ACCESS BY END USERS USING REPORTING TOOL

TYPES OF VIEWS:

1. USER DEFINED VIEWS

2. SYSTEM PREDEFINED VIEWS

3. DYNAMIC VIEWS: THESE ARE A SPECIAL TYPE OF VIEWS. USED FOR METADATA ACCESS. PROPERTIES.

SYNTAX OF VIEW DEFINITION:

CREATE VIEW <VIEWNAME>

WITH

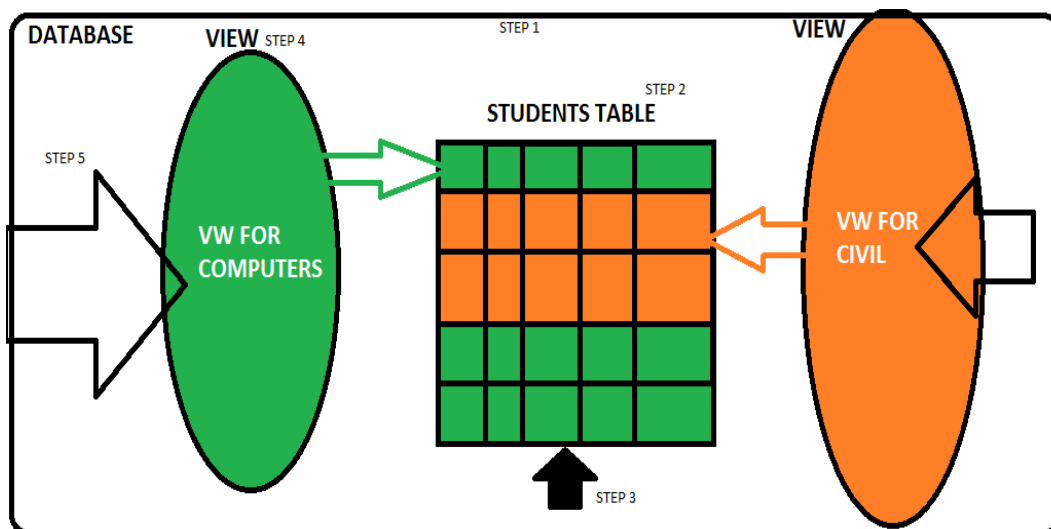
SCHEMABINDING : TO PREVENT ORPHAN VIEWS. ORPHAN VIEWS ARE SUCH VIEWS WITH UNKNOWN TABLES

, ENCRYPTION : TO hide DEFINITION OF THE VIEW IN DATABASE

AS

<SELECT QUERY>

WITH CHECK OPTION : TO PREVENT UNAUTHORIZED DML OPERATIONS USING THE VIEW.



EXAMPLE:

```
CREATE DATABASE STUDENTDATABASE
```

```
USE STUDENTDATABASE
```

```
CREATE TABLE STUDENTS
```

```
(
```

```
STD_ID INT PRIMARY KEY,
```

```
STD_NAME VARCHAR(30),
```

```
STD_AGE FLOAT,
```

```
STD_COURSE VARCHAR(30)
```

```
)
```

```
INSERT STUDENTS VALUES
```

```
(1001, 'AMIN', 23.5, 'COMPUTERS'), (1002, 'SAM', 23.5, 'CIVIL'),
```

```
(1003, 'AMINI', 23.5, 'COMPUTERS')
```

HOW TO REPORT LIST OF ALL STUDENTS FOR COMPUTERS COURSE?

```
SELECT * FROM STUDENTS
```

```
WHERE
```

```
STD_COURSE = 'COMPUTERS'
```

1. HOW TO STORE ABOVE QUERY IN THE DATABASE?

SOLUTION: USING "VIEWS"

NOW CREATE A VIEW FOR COMPUTER STUDENTS

```
CREATE VIEW VW_COMP_STUDENTS
```

```
AS
```

```
SELECT * FROM STUDENTS
```

```
WHERE
```

```
STD_COURSE = 'COMPUTERS'
```

2. HOW TO ACCESS ABOVE VIEW?

```
SELECT * FROM VW_COMP_STUDENTS
```

```
-- COMPUTERS DATA IS REPORTED FROM TABLE
```

3. WHAT OPERATIONS CAN BE PERFORMED ON THE VIEW?

SELECT, INSERT, UPDATE, DELETE

WHENEVER WE INSERT/UPDATE/DELETE FROM VIEW ACTUAL DATA IS AFFECTED ON BASE TABLE.

UPDATE VW_COMP_STUDENTS SET STD_AGE = 30

-- COMPUTERS DATA IS UPDATED IN TABLE

HERE WE ARE UPDATING COLUMN VALUE USING VIEW.

DELETE FROM VW_COMP_STUDENTS -- COMPUTERS DATA IS REMOVED FROM TABLE

-- ISSUE 1: INVALID DATA INSERTS. INSERT STATEMENT IS NOT CONDITIONAL.

INSERT VW_COMP_STUDENTS VALUES (1005, 'AMINISHA', 23.5, 'CIVIL')

THE ABOVE INSERT STATEMENT WILL BE EXECUTED SUCCESSFULLY.

SINCE WE HAVE CREATED A VIEW FOR COMPUTER STUDENTS

'VW_COMP_STUDENTS', INSERTED DATA IS CIVIL-MEANS WE CAN'T RESTRICT IT.

SOLUTION:

ALTER THE TABLE , MAKE IT 'WITH CHECK OPTION'

CODE:

ALTER VIEW VW_COMP_STUDENTS

AS

SELECT * FROM STUDENTS

WHERE

STD_COURSE = 'COMPUTERS'

WITH CHECK OPTION

--THE ABOVE WITH CHECK OPTION RESTRICTS STD_COURSE ENTERING INVALID INPUTS.

-- VERIFY THE SOLUTION:

INSERT VW_COMP_STUDENTS VALUES (1009, 'AMINISHA', 23.5, 'CIVIL') -- ERROR

INSERT VW_COMP_STUDENTS VALUES (1010, 'AMINISHA', 23.5, 'COMPUTERS')

-- NO ERROR


```
-- VERIFY THE SOLUTION:
INSERT VW_COMP_STUDENTS VALUES (1009, 'AMINISHA', 23.5, 'CIVIL') -- ERROR
INSERT VW_COMP_STUDENTS VALUES (1010, 'AMINISHA', 23.5, 'COMPUTERS') -- NO ERROR
```

100 %

Messages

Msg 550, Level 16, State 1, Line 61
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view
The statement has been terminated.

(1 row affected)

ISSUE 2: ORPHAN VIEWS

DROP TABLE STUDENTS

SUPPOSE THE TABLE IS DROPPED, VIEW ON THE TABLE BECOMES ORPHAN.

SELECT * FROM VW_COMP_STUDENTS --ERROR

THE ABOVE STATEMENT GIVES ERROR.

INSERT VW_COMP_STUDENTS VALUES (1010, 'AMINISHA', 23.5, 'COMPUTERS') --
ERROR

BOTH STATEMENTS GIVE ERROR, BECAUSE THE VIEW IS DROPPED.

```
-- ISSUE 2: ORPHAN VIEWS
DROP TABLE STUDENTS

SELECT * FROM VW_COMP_STUDENTS -- ERROR

-- SOLUTION: HOW TO PREVENT ORPHAN VIEWS?
-- MEANS, HOW TO PREVENT "DROP" OF BASE TABLE?

CREATE TABLE STUDENTS
(
    STD_ID INT PRIMARY KEY,
```

100 %

Messages

Msg 208, Level 16, State 1, Procedure VW_COMP_STUDENTS, Line 3 [Batch Start Line 68]
Invalid object name 'STUDENTS'.
Msg 4413, Level 16, State 1, Line 69
Could not use view or function 'VW_COMP_STUDENTS' because of binding errors.

-- SOLUTION: HOW TO PREVENT ORPHAN VIEWS?
-- MEANS, HOW TO PREVENT "DROP" OF BASE TABLE?
NOW CREATE A TABLE

```
CREATE TABLE STUDENTS  
(  
  STD_ID INT PRIMARY KEY,  
  STD_NAME VARCHAR(30),  
  STD_AGE FLOAT,  
  STD_COURSE VARCHAR(30)  
)
```

-- SOLUTION:
ALTER VIEW VW_COMP_STUDENTS
WITH SCHEMABINDING -- SCHEMA means structure
AS
SELECT STD_ID, STD_NAME, STD_COURSE, STD_AGE
FROM DBO.STUDENTS -- DBO is the default namespace. Database Owner
WHERE
STD_COURSE = 'COMPUTERS'
WITH CHECK OPTION

drop table STUDENTS -- error

```
-- final view creation statement:  
CREATE VIEW VW_CIVIL_STUDENTS  
WITH SCHEMABINDING , ENCRYPTION  
AS  
SELECT STD_ID, STD_NAME, STD_COURSE, STD_AGE  
FROM DBO.STUDENTS  
WHERE  
STD_COURSE = 'CIVIL'  
WITH CHECK OPTION
```

Messages

Msg 3701, Level 11, State 5, Line 94
Cannot drop the table 'STUDENTS', because it does not exist or you do not have permission.

-- FINAL VIEW CREATION STATEMENT:

```
CREATE VIEW VW_CIVIL_STUDENTS
WITH SCHEMABINDING, ENCRYPTION
AS
SELECT STD_ID, STD_NAME, STD_COURSE, STD_AGE
FROM dbo.STUDENTS
WHERE
STD_COURSE = 'CIVIL'
WITH CHECK OPTION
```

IMPORTANT SYSTEM VIEWS

HOW TO REPORT ALL DATABASES IN THE CURRENT INSTANCE / SERVER?

```
SELECT * FROM SYS.DATABASES
```

HOW TO REPORT ALL TABLES IN CURRENT DATABASE?

```
SELECT * FROM SYS.TABLES
```

HOW TO REPORT ALL VIEWS IN CURRENT DATABASE?

```
SELECT * FROM SYS.views
```

HOW TO REPORT ALL VIEWS & DEFINITIONS IN CURRENT DATABASE?

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS
```

HOW TO REPORT ALL OBJECTS IN CURRENT DATABASE ?

```
SELECT * FROM SYS.objects --TABLES, VIEWS, KEYS, CONSTRAINTS, ETC..
```

EVERY VIEW DEFINITION IS STORED IN PRIMARY FILE OF THE DATABASE.

EVERY VIEW COMPILED FORMAT (COMPILED PLAN) IS STORED IN A MEMORY LOCATION CALLED "PROCEDURE CACHE"

HOW TO QUERY PROCEDURE CACHE?

```
SELECT * FROM SYS.DM_EXEC_CACHED_PLANS
```

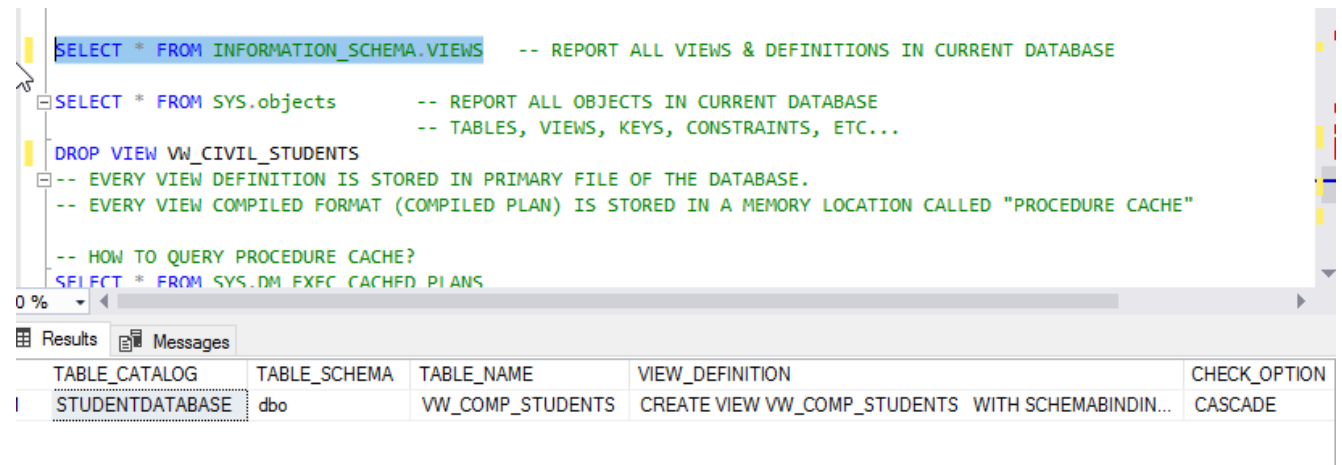
AN INTERNAL COMPONENT OF SQL SERVER "QUERY OPTIMIZER (QO)" CHECKS FOR AVAILABLE COMPILED PLAN TO EXECUTE THE QUERY.

HOW TO HIDE VIEW DEFINITION OR ENCRYPT THE VIEW DEFINITION?

WE USE A KEYWORD CALLED 'ENCRYPTION' FOR A VIEW TO HIDE THE VIEW DEFINITION.

HOW TO SEE VIEW DEFINITION?

SELECT * FROM INFORMATION_SCHEMA.VIEWS



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following text:

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS -- REPORT ALL VIEWS & DEFINITIONS IN CURRENT DATABASE
SELECT * FROM SYS.objects -- REPORT ALL OBJECTS IN CURRENT DATABASE
-- TABLES, VIEWS, KEYS, CONSTRAINTS, ETC...
DROP VIEW VW_CIVIL_STUDENTS
-- EVERY VIEW DEFINITION IS STORED IN PRIMARY FILE OF THE DATABASE.
-- EVERY VIEW COMPILED FORMAT (COMPILED PLAN) IS STORED IN A MEMORY LOCATION CALLED "PROCEDURE CACHE"
-- HOW TO QUERY PROCEDURE CACHE?
SELECT * FROM SYS.DM_EXEC_CACHED_PLANS
```

The bottom pane shows the 'Results' tab with a grid containing one row of data:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	VIEW_DEFINITION	CHECK_OPTION
STUDENTDATABASE	dbo	VW_COMP_STUDENTS	CREATE VIEW VW_COMP_STUDENTS WITH SCHEMABINDIN...	CASCADE

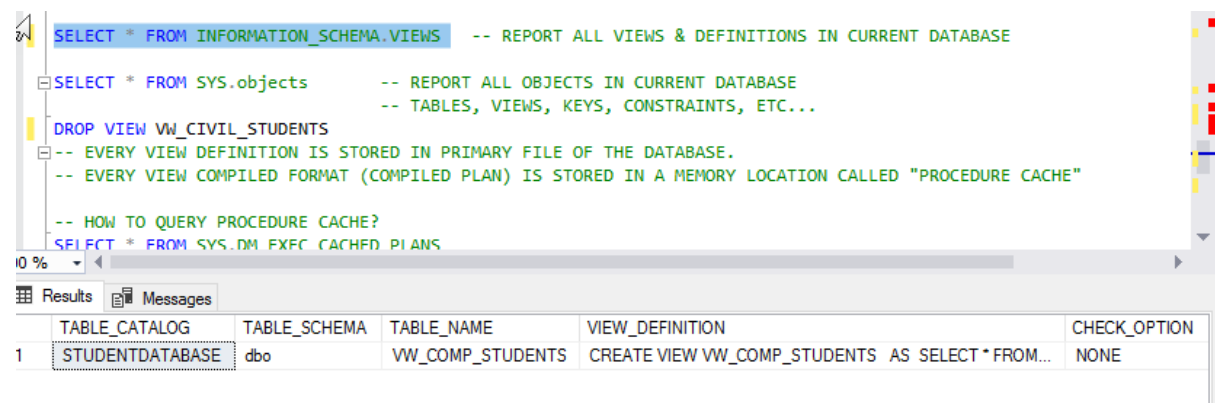
IF YOU SEE VIEW DEFINITION, THE EXECUTED QUERY STATEMENT IS PRESENT.

YOU CAN SEE VIEW DEFINITION OF A VIEW, ONLY IF VIEW IS NOT ENCRYPTED.

IF IT IS ENCRYPTED YOU CAN SEE NULL VALUE.

IF WITH CHECKOPTION , YOU CAN SEE 'CASCADE'

IF 'WITH CHECKOPTION' IS NOT MENTIONED ,YOU CAN SEE 'NONE'



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following text:

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS -- REPORT ALL VIEWS & DEFINITIONS IN CURRENT DATABASE
SELECT * FROM SYS.objects -- REPORT ALL OBJECTS IN CURRENT DATABASE
-- TABLES, VIEWS, KEYS, CONSTRAINTS, ETC...
DROP VIEW VW_CIVIL_STUDENTS
-- EVERY VIEW DEFINITION IS STORED IN PRIMARY FILE OF THE DATABASE.
-- EVERY VIEW COMPILED FORMAT (COMPILED PLAN) IS STORED IN A MEMORY LOCATION CALLED "PROCEDURE CACHE"
-- HOW TO QUERY PROCEDURE CACHE?
SELECT * FROM SYS.DM_EXEC_CACHED_PLANS
```

The bottom pane shows the 'Results' tab with a grid containing one row of data:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	VIEW_DEFINITION	CHECK_OPTION
STUDENTDATABASE	dbo	VW_COMP_STUDENTS	CREATE VIEW VW_COMP_STUDENTS AS SELECT * FROM...	NONE

SQL SERVER ARCHITECTURE 1:

SQL SERVER ARCHITECTURE HAS THREE COMPONENTS:

1. PROTOCOLS : NETWORK PROVIDERS USED TO COMMUNICATE TDS PACKETS.

TCP : **TRANSMISSION CONTROL PROTOCOL**

TCP USED FOR CLIENT - SERVER COMMUNICATION OVER INTERNET.

NAMED PIPES

NAMED PIPES USED FOR CLIENT - SERVER COMMUNICATION OVER LAN.

SHARED MEMORY

SHARED MEMORY USED FOR CLIENT - SERVER COMMUNICATION WITH THE SAME OS.

VIRTUAL INTERFACE ADAPTOR -VIA

VIA USED FOR CLIENT - SERVER COMMUNICATION OVER INTERNET.

2. SQL ENGINE : ALSO CALLED DATABASE ENGINE

TWO COMPONENTS IN THIS DATABASE ENGINE

a. QUERY PROCESSOR b. STORAGE ENGINE

2.a. PARSER : USED FOR QUERY CHECK AND QUERY COMPILATION.

2.b. OPTIMIZER : USED FOR QUERY ANALYSIS AND LOOKING FOR THE BEST INDEX TO BE USED FOR QUERY EXECUTION = QUERY OPTIMIZER (QO).

2.c. SQL MANAGER : USED TO ANALYSE DATABASE HEALTH AND VERIFY THE DATABASE AVAILABILITY FOR REQUIRED OPERATIONS.

2.d. DATABASE MANAGER : LOCKS THE DATABASE LOG FILE AND IF REQUIRED.

DATA FILE TO PERFORM THE OPERATIONS.

2.e. QUERY EXECUTION ENGINE : TRIGGERS THE ACTUAL EXECUTION.

CONTROL GOES TO STORAGE ENGINE.

b. STORAGE ENGINE

2.f. TRANSACTION MANAGER : TO MANAGE/MONITOR MEMORY

2.g. MEMORY MANAGER : ALLOCATE MEMORY TO QUERIES.

2.h. LOCK MANAGER : QUERY LOCKS & BLOCKING

2.i. FILE MANAGER : DATABASE DATA FILES, LOG FILES

2.j. UTILITIES : IMPORT/EXPORT, SSIS, ETC.

2.k. STORAGE ACCESS UNITS : FILES, PAGES, RID, EXTENT

ROW ID

ROW IDENTIFIER

3. SQL OS : APPLICABLE FOR OPERATING SYSTEM TO CONTROL SQL SERVER SERVICE

SERVICE IS A COMMUNICATION BETWEEN SQL SERVER AND OPERATING SYSTEM.

TO MONITOR THIS SERVICE, ONE "SERVICE ACCOUNT" IS AUTO CREATED DURING SQL SERVER INSTALLATION.

MEMORY MANAGER: USED TO ALLOCATE MEMORY TO SQL SERVER.

DYNAMIC ALLOCATION

BUFFER MANAGER: USED TO "BUFFER" OR "CACHE" THE DATABASE DATA.

FOR FASTER DATA ACCESS & COMPUTATIONS.

LOCK MANAGER: USED TO ALLOCATE LOCKS ON PAGES AND FILES.

FOR EFFICIENT DATA STORAGE AND ACCESS.

IO MANAGER: USED TO ALLOCATE I/O INTERRUPTS TO ACCEPT PARAMETER VALUES AND ALSO TO PRODUCE THE OUTPUT AT THE CLIENT.

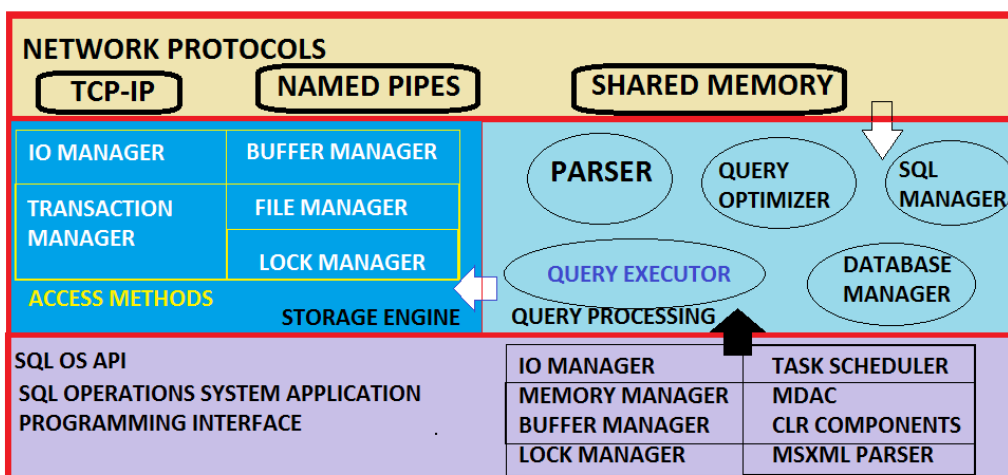
TASK SCHEDULER: USED TO "SCHEDULE" / "AUTOMATE" ANY SQL SERVER OPERATION.

MDAC: MICROSOFT DATA ACCESS COMPONENT.

RESPONSIBLE FOR CLIENT - SERVER COMMUNICATION WITH SSMS TOOL. USED TO "ENCRYPT" DATA BEING SENT OVER TDS PACKETS TO THE SERVER.

CLR: COMMON LANGUAGE RUNTIME. THESE ARE LIBRARY FILES.

USED TO INTEREACT WITH SQL SERVER USING SPECIFIC SUB ROUTINES FROM .NET AND OTHER PROGRAMMING PLATFORMS.



Q&A

Q1: WHICH ONE YOU PREFER? CHAR OR VARCHAR? FOR DESINING TABLES? WHY?

CHAR : TO STORE STATIC DATA EX: USERNAME

VARCHAR : TO STORE VARYING / DYNAMIC DATA. EX: PASSWORDS

Q2: WHAT IS THE DIFFERENCE BETWEEN A BATCH AND A GO STATEMENT?

A BATCH IS A MECHANISM : TO DENOTE A COLLECTION OF SQL STATEMENTS.

GO IS A SQL STATEMENT TO SEPERATE MULTIPLE BATCHES.

NUMBER OF BATCHES = NUMBER OF GOs + 1

Q3: YOU WERE GIVEN BELOW CODE. FIND THE NUMBER OF ERRORS?

```
CREATE VIEW VW_TEST
```

```
AS
```

```
SELECT * FROM TABLE1
```

```
JOIN
```

```
TABLE2 ON
```

```
TABLE1.COL1 IN TABLE2.COL2      -- ALL TABLES ARE VALID
```

NO ERRORS

Q4. WHAT IS THE ADVANTAGE OF SCHEMAS IN A DATABASE?

USED FOR EASY SECURITY MANAGMENT AND EASY GROUPING OF TABLES

Q5. WHAT SHOULD BE THE MINIMUM SIZE OF FILE IN A DATABASE?

1MB

Q6. CAN A PRIMARY FILE BE OF EXTENSION "NDF" ?

YES

Q7. IF YOU CREATE A DATABASE WITH PRIMARY FILE IN D DRIVE. THEN WHERE WOULD BE THE LOG FILE CREATED?

OPTION 1: ON D DRIVE

OPTION 2: ON DEFAULT INSTALLATION LOCATION

OPTION 1

Q8. TABLE STRUCTURE IS STORED IN WHICH FILEGROUP?

OPTION 1: IN THE SPECIFIED FILEGROUP WHILE CREATING TABLE

OPTION 2: IN THE PRIMARY FILEGROUP

OPTION 3: IN LOG FILE

OPTION 2

Q9.WHAT IS THE DIFFERENCE BETWEEN WHERE AND HAVING? WHAT ARE THE PRECONDITIONS TO USE HAVING?

WHERE : TO SPECIFY CONDITIONS BASED ON DATA REPORTED FROM TABLES & JOINS

HAVING: TO SPECIFY CONDITIONS BASED ON DATA REPOTED FROM GROUP BY

GROUP BY IS THE PRECONDITION FOR HAVING.

STORED PROCEDURES

STORED PROCEDURES ARE DATABASE OBJECTS TO STORE ANY TYPE OF T-SQL STATEMENTS IN THE DATABASE

PURPOSE:

1. REUSABILITY OF CODE
2. FOR DATA VALIDATIONS
3. FOR DYNAMIC SQL PROGRAMMING
4. FOR DATA FORMATTING & REPORTING
5. FOR QUERY TUNING [MAKING QUERIES TO EXECUTE FASTER]

TYPES OF STORED PROCEDURES:

1. USER DEFINED STORED PROCEDURES
2. SYSTEM PREDEFINED STORED PROCEDURES
3. EXTENDED STORED PROCEDURES : These SPs can be used for Dynamic SQL Queries.

SYNTAX FOR USER DEFINED STORED PROCEDURES:

```
CREATE PROCEDURE <NAME OF PROCEDURE> (@PARAMETER1 DEFN, @PARAMETER2  
DEFN,...)
```

```
AS
```

```
BEGIN
```

```
STATEMENT 1
```

```
STATEMENT 2
```

```
STATEMENT 3.....
```

```
END
```

HOW TO EXECUTE STORED PROCEDURES?

```
EXECUTE <PROCEDURENAME> @PARAMETER1=VALUE, @PARAMETER2=VALUE.....
```

IMPORTANT SYSTEM STORED PROCEDURES:

1. SP_HELP - To report metadata of any object (table, view, procedure, etc..)
2. SP_HELPDB- To report metadata / properties of any database.
3. SP_WHO2- To report list of all sessions & SPIDs currently available in server.
4. SP_RENAMEDB- To rename a database
5. SP_RENAME - To rename any database object (table / view / procedure..)
6. SP_HELPTEXT -To report text / definition of View or Procedure
7. SP_RECOMPILE- To recompile a procedure
8. SP_REFRESH - To recompile a view

EXAMPLE:

```
CREATE TABLE tblProducts  
(  
  PrdId int PRIMARY KEY,  
  PrdName varchar(30) NOT NULL,  
  PrdSize char,  
  PrdPrice float  
)
```

REQ : FOR SMALL SIZE PRODUCTS, MINIMUM PRICE SHOULD BE 1\$
 FOR MEDIUM SIZE PRODUCTS, MINIMUM PRICE SHOULD BE 2\$
 FOR LARGE SIZE PRODUCTS, MINIMUM PRICE SHOULD BE 3\$

```

CREATE PROCEDURE usp_ValidateInsert

    @id int, @name varchar(30), @size char, @price float    -- 4
PARAMETERS

    AS

    BEGIN

    IF @size = 'S' AND @PRICE >= 1

    INSERT INTO tblProducts VALUES (@ID, @NAME, @SIZE, @PRICE)

    ELSE IF @size = 'M' AND @PRICE >= 2

    INSERT INTO tblProducts VALUES (@ID, @NAME, @SIZE, @PRICE)

    ELSE IF @size = 'L' AND @PRICE >= 3

    INSERT INTO tblProducts VALUES (@ID, @NAME, @SIZE, @PRICE)

    ELSE

    PRINT 'INVALID INPUT VALUES'

    END

```

AS PER OUR REQUIREMENT WE HAVE CREATED A STORED PROCEDURE TO VALIDATE THE INPUTS IN OUR TABLE.

-- HOW TO EXECUTE ABOVE STORED PROCEDURE?

```
EXECUTE usp_ValidateInsert @ID = 1001, @NAME = 'CAPS', @SIZE = 'S', @PRICE = 1
```

WE HAVE TO GIVE INPUT THROUGH THE STORED PROCEDURE.

-- VERIFY:

```
SELECT * FROM tblProducts
```

TO CHECK ABOVE PROCEDURE IS FUNCTIONALITY?

```
EXECUTE usp_ValidateInsert @ID = 1002, @NAME = 'CAPS', @SIZE = 'L', @PRICE = 1
```

THE ABOVE STATEMENT GIVES ERROR, IT VIOLATES THE CONDITION SIZE 'L' SHOULD BE MINIMUM OF \$3

BENEFITS OF PROCEDURES (STORED PROCEDURES / SPs):

1. PRECOMPILED :

COMPILATION MEANS CONVERSION OF T-SQL STATEMENTS INTO MACHINE CODE DURING EXECUTION, NO ADDITIONAL PRECOMPILE REQUIRED.
MEANS, FASTER QUERY EXECUTION.

2. PREOPTIMIZED:

OPTIMIZATION MEANS TRACKING / LOOKUP FOR BEST EXECUTION PLAN
"QUERY OPTIMIZER" (QO) COMPONENT TAKES CARE OF THIS OPERATION.

USE CASE SCENARIO 1 : DATA CONSISTENCY

```
CREATE TABLE PROD_DETAILS
```

```
(
```

```
PROD_ID INT IDENTITY(1001,1) PRIMARY KEY,
```

```
PROD_NAME VARCHAR(30),
```

```
PROD_SIZE CHAR(2) CHECK (PROD_SIZE IN ('S', 'M', 'L', 'NA')),
```

```
PROD_PRICE FLOAT
```

```
)
```

```
INSERT INTO PROD_DETAILS VALUES ('CAPS', 'S',1)
```

```
INSERT INTO PROD_DETAILS VALUES ('CAPS', 'M',2)
```

```
INSERT INTO PROD_DETAILS VALUES ('CAPS', 'X',1)  -- INVALID SIZE. MISSING  
IDENTITY VALUE!!
```

```
INSERT INTO PROD_DETAILS VALUES ('CHAINS', 'S',1)
```

```
INSERT INTO PROD_DETAILS VALUES ('CHAINS', 'M',2)
```

IN THE ABOVE 5 INSERT STATEMENTS, 3RD STATE IS INVALID, BUT STILL THE OTHER INSERT STATEMENTS CONTINUE TO EXECUTE.

```

CREATE TABLE PROD_DETAILS
(
    PROD_ID INT IDENTITY(1001,1) PRIMARY KEY,
    PROD_NAME VARCHAR(30),
    PROD_SIZE CHAR(2) CHECK (PROD_SIZE IN ('S', 'M', 'L', 'NA')),
    PROD_PRICE FLOAT
)

INSERT INTO PROD_DETAILS VALUES ('CAPS', 'S', 1)
INSERT INTO PROD_DETAILS VALUES ('CAPS', 'M', 2)
INSERT INTO PROD_DETAILS VALUES ('CAPS', 'X', 1) -- INVALID SIZE. MISSING IDENTITY VALUE!!
INSERT INTO PROD_DETAILS VALUES ('CHAINS', 'S', 1)
INSERT INTO PROD_DETAILS VALUES ('CHAINS', 'M', 2)

```

Messages

```

(1 row affected)

(1 row affected)
Msg 547, Level 16, State 0, Line 33
The INSERT statement conflicted with the CHECK constraint "CK__PROD_DETA__PROD__405A880E". The conflict occurred
The statement has been terminated.

(1 row affected)

(1 row affected)

```

SELECT * FROM PROD_DETAILS

```

INSERT INTO PROD_DETAILS VALUES ('CAPS', 'S', 1)
INSERT INTO PROD_DETAILS VALUES ('CAPS', 'M', 2)
INSERT INTO PROD_DETAILS VALUES ('CAPS', 'X', 1) -- INVALID SIZE. MISSING IDENTITY VALUE!!
INSERT INTO PROD_DETAILS VALUES ('CHAINS', 'S', 1)
INSERT INTO PROD_DETAILS VALUES ('CHAINS', 'M', 2)

SELECT * FROM PROD_DETAILS
TRUNCATE TABLE PROD_DETAILS

-- SOLUTION TO ABOVE ISSUE : MISSING IDENTITY (DATA CONSISTENCY ISSUE)

CREATE PROCEDURE uspInsertCheck (@PrdName varchar(30), @prdSize char(2), @PrdPrice float)

```

Results

PROD_ID	PROD_NAME	PROD_SIZE	PROD_PRICE
1001	CAPS	S	1
1002	CAPS	M	2
1004	CHAINS	S	1
1005	CHAINS	M	2

BY OBSERVING OUTPUT, WE CAN SEE THE MISSING OF SEQUENCE BECAUSE OF INVALID INSERTS.

HERE PROD_ID IS IDENTITY DATATYPE. NOW TRUNCATE THE TABLE AND VALIDATE THE TABLE USING SP'S

TRUNCATE TABLE PROD_DETAILS

```
-- SOLUTION TO ABOVE ISSUE : MISSING IDENTITY (DATA CONSISTENCY ISSUE)

CREATE PROCEDURE uspInsertCheck (@PrdName varchar(30), @prdSize char(2),
@PrdPrice float)

AS

BEGIN /* START OF THE STORED PROCEDURE*/

IF @prdSize IN ('S','M','L','NA')

INSERT INTO PROD_DETAILS VALUES (@PrdName, @prdSize, @PrdPrice)

ELSE

PRINT 'INVALID INPUT'

END /* END OF STORED PROCEDURE */
```

HOW TO EXECUTE ABOVE STORED PROCEDURE?

```
EXECUTE uspInsertCheck 'CAPS', 'S',1

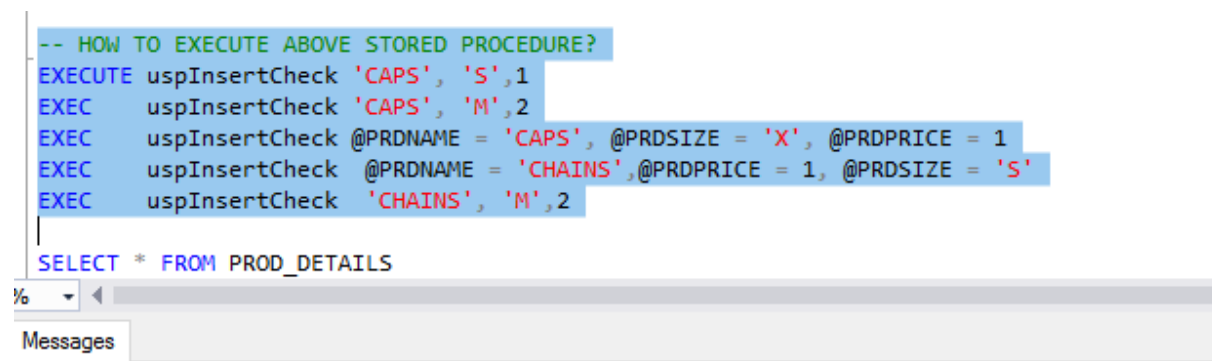
EXEC uspInsertCheck 'CAPS', 'M',2

EXEC uspInsertCheck @PRDNAME = 'CAPS', @PRDSIZE = 'X', @PRDPRICE = 1

-- THE ABOVE PRDSIZE IS INVALID.

EXEC uspInsertCheck @PRDNAME = 'CHAINS',@PRDPRICE = 1, @PRDSIZE = 'S'

EXEC uspInsertCheck 'CHAINS', 'M',2
```



```
-- HOW TO EXECUTE ABOVE STORED PROCEDURE?
EXECUTE uspInsertCheck 'CAPS', 'S',1
EXEC uspInsertCheck 'CAPS', 'M',2
EXEC uspInsertCheck @PRDNAME = 'CAPS', @PRDSIZE = 'X', @PRDPRICE = 1
EXEC uspInsertCheck @PRDNAME = 'CHAINS',@PRDPRICE = 1, @PRDSIZE = 'S'
EXEC uspInsertCheck 'CHAINS', 'M',2
SELECT * FROM PROD_DETAILS
```

Messages

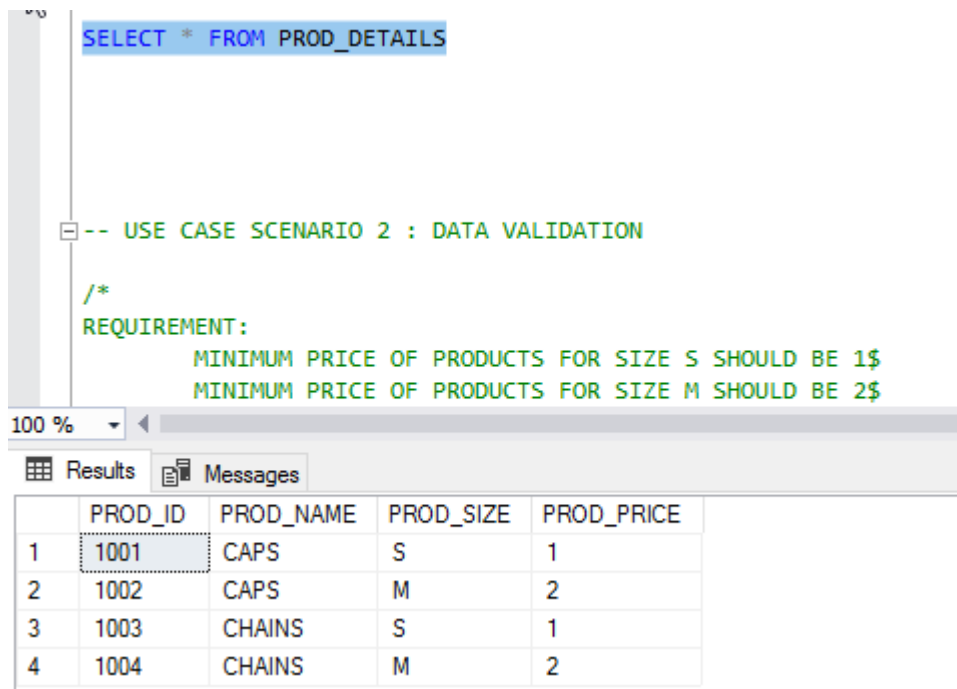
```
(1 row affected)

(1 row affected)
INVALID INPUT

(1 row affected)

(1 row affected)
```

NOW CHECK THE SEQUENCE



The screenshot shows a SQL query window with the following content:

```
SELECT * FROM PROD_DETAILS
```

Below the query window, there is a tab labeled "Results" which displays the following data:

	PROD_ID	PROD_NAME	PROD_SIZE	PROD_PRICE
1	1001	CAPS	S	1
2	1002	CAPS	M	2
3	1003	CHAINS	S	1
4	1004	CHAINS	M	2

-- HOW TO MODIFY THE STRUCTURE OF A GIVEN STORED PROCEDURE?

```
ALTER PROC usp_ValidateInsert (@name varchar(30), @size char(2), @price float)
```

AS

BEGIN

```
    IF @SIZE = 'S' AND @PRICE >= 1
```

```
        INSERT INTO PROD_DETAILS VALUES (@NAME, @SIZE, @PRICE)
```

```
    ELSE IF @SIZE = 'M' AND @PRICE >= 2
```

```
        INSERT INTO PROD_DETAILS VALUES (@NAME, @SIZE, @PRICE)
```

```
    ELSE IF @SIZE = 'L' AND @PRICE >= 3
```

```
        INSERT INTO PROD_DETAILS VALUES (@NAME, @SIZE, @PRICE)
```

```
    ELSE IF @SIZE NOT IN ('S','M','L') AND @PRICE >= 1
```

```
        INSERT INTO PROD_DETAILS VALUES (@NAME, @SIZE, @PRICE)
```

```
    ELSE
```

```
        PRINT 'INVALID INPUT'
```

END

-- TEST ABOVE SP:

EXEC usp_ValidateInsert 'CHAINS', 'L', 1

```
ALTER PROC usp_ValidateInsert @name varchar(30), @size char(2) = 'S', @price float =
1
AS
BEGIN
    IF @SIZE = 'S' AND @PRICE >= 1
        INSERT INTO PROD_DETAILS VALUES (@NAME, 'S', @PRICE)
    ELSE IF @SIZE = 'M' AND @PRICE >= 2
        INSERT INTO PROD_DETAILS VALUES (@NAME, 'M', @PRICE)
    ELSE IF @SIZE = 'L' AND @PRICE >= 3
        INSERT INTO PROD_DETAILS VALUES (@NAME, 'L', @PRICE)
    ELSE IF @SIZE NOT IN ('S', 'M', 'L') AND @PRICE >= 1
        INSERT INTO PROD_DETAILS VALUES (@NAME, @SIZE, @PRICE)
    ELSE
        PRINT 'INVALID INPUT'
END
```

/*

POINT #1: WHENEVER POSSIBLE, USE STATIC VALUES.

POINT #2: USE "ELSE IF" WHEREVER POSSIBLE. THIS AVOID UNWANTED CONDITIONAL CHECKS.

POINT #3: THE USE OF PARANTHESIS IS OPTIONAL FOR PARAMETERS DURING PROCEDURE CREATION.

WE SHOULD NOT USED PARAMETERS DURING PROCEDURE EXECUTION.

POINT #4: THE USE OF BEGIN ... END IS OPTIONAL. BUT RECOMMENDED.

*/

/* SYSTEM STORED PROCEDURES :

THESE ARE SYSTEM LEVEL, PREGENERATED PROCEDURES.

AUTO CREATED DURING DATABASE CREATION.

TO REPORT PROPERTIES OF DATABASES, TABLES, OTHER OBJECTS.

TO CHANGE PROPERTIES OF DATABASES, TABLES, COLUMNS, ETC..

*/

EXEC SP_HELPDB -- REPORTS LIST OF ALL DATABASES ON THE SERVER

EXEC SP_HELPDB 'UNIVDATABASE' -- REPORTS PROPERTIES OF SPECIFIED DATABASE

EXEC SP_HELP 'PROD_DETAILS' -- REPORTS PROPERTIES OF THE TABLE

EXEC SP_PKEYS 'PROD_DETAILS' -- REPORTS PRIMARY KEY DETAILS OF THE TABLE

EXEC SP_DEPENDS 'PROD_DETAILS' -- REPORTS DEPENDANCIES OF THE TABLE

EXEC SP_RENAME 'PROD_DETAILS', 'PROD_DETAILS_NEW' -- TO RENAME THE TABLE

EXEC SP_RENAMEDB 'UNIVDATABASE', 'University_DBase' -- TO RENAME THE
DATABASE

EXEC SP_HELPTEXT 'uspInsertCheck' -- TO RETREIVE PROCEDURE DEFINITION

```
ALTER PROCEDURE uspInsertCheck (@PrdName varchar(30), @prdSize char(2),  
@PrdPrice float)
```

```
AS
```

```
BEGIN          /* START OF THE STORED PROCEDURE*/
```

```
IF @prdSize IN ('S','M','L','NA')
```

```
    INSERT INTO PROD_DETAILS VALUES (@PrdName, @prdSize, @PrdPrice)
```

```
ELSE
```

```
    PRINT 'INVALID INPUT'
```

```
END            /* END OF STORED PROCEDURE */
```

HOW TO SEE PROCEDURE DEFINITION?

EXEC SP_HELPTEXT 'uspInsertCheck' -- TO RETREIVE PROCEDURE DEFINITION

The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays the SQL code for the stored procedure `uspInsertCheck`. The bottom pane shows the results of the `EXEC SP_HELPTEXT 'uspInsertCheck'` command, which returns the definition of the procedure.

```
EXEC SP_HELPTEXT 'uspInsertCheck' -- TO RETREIVE PROCEDURE DEFINITION
```

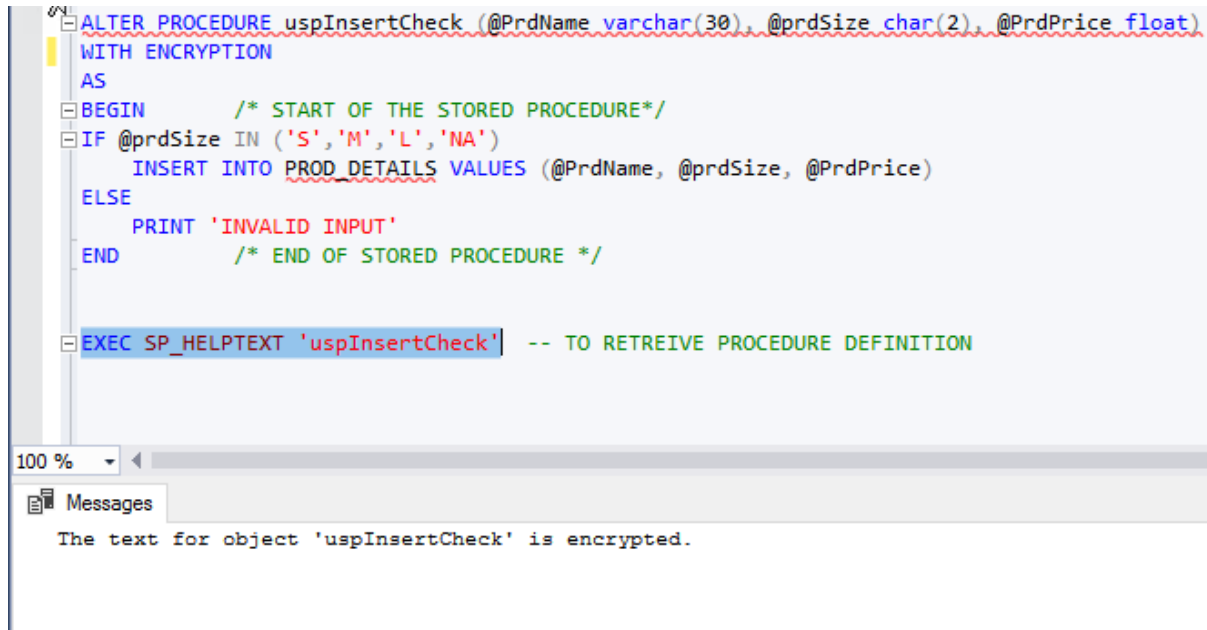
-- IN GENERAL, WE WORK WITH "ASCII" VALUES.
-- AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE

-- BELOW SYSTEM SP IS USED TO EXECUTE ANY TYPE OF T-SQL STATEMENT.

	Text
1	CREATE PROCEDURE uspInsertCheck (@PrdName varchar(...
2	
3	AS
4	BEGIN /* START OF THE STORED PROCEDURE*/
5	IF @prdSize IN ('S','M','L','NA')
6	INSERT INTO PROD_DETAILS VALUES (@PrdName, @prd...
7	ELSE
8	PRINT 'INVALID INPUT'
9	END /* END OF STORED PROCEDURE */

HOW TO HIDE PROCEDURE DEFINITION?

BY USING ENCRYPTION KEYWORD WE CAN HIDE THE PROCEDURE.



The screenshot shows a SQL Server Enterprise Manager interface. The main pane displays the definition of a stored procedure named `uspInsertCheck`. The procedure is defined with parameters `@PrdName varchar(30)`, `@prdSize char(2)`, and `@PrdPrice float`. It is created with the `WITH ENCRYPTION` keyword. The procedure body includes a `BEGIN` block with a comment `/* START OF THE STORED PROCEDURE*/`, followed by an `IF` statement checking if `@prdSize` is in ('S', 'M', 'L', 'NA'). If true, it inserts into `PROD_DETAILS`. Otherwise, it prints 'INVALID INPUT'. The block ends with `END` and a comment `/* END OF STORED PROCEDURE */`. Below the procedure definition, there is a comment `-- TO RETREIVE PROCEDURE DEFINITION` followed by the command `EXEC SP_HELPTEXT 'uspInsertCheck'`. The bottom pane shows the output of this command: "The text for object 'uspInsertCheck' is encrypted."

```
ALTER PROCEDURE uspInsertCheck (@PrdName varchar(30), @prdSize char(2), @PrdPrice float)
WITH ENCRYPTION
AS
BEGIN
    /* START OF THE STORED PROCEDURE*/
    IF @prdSize IN ('S', 'M', 'L', 'NA')
        INSERT INTO PROD_DETAILS VALUES (@PrdName, @prdSize, @PrdPrice)
    ELSE
        PRINT 'INVALID INPUT'
    END
    /* END OF STORED PROCEDURE */

EXEC SP_HELPTEXT 'uspInsertCheck' -- TO RETREIVE PROCEDURE DEFINITION
```

100 %

Messages

The text for object 'uspInsertCheck' is encrypted.

IN GENERAL, WE WORK WITH "ASCII" VALUES.

AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE

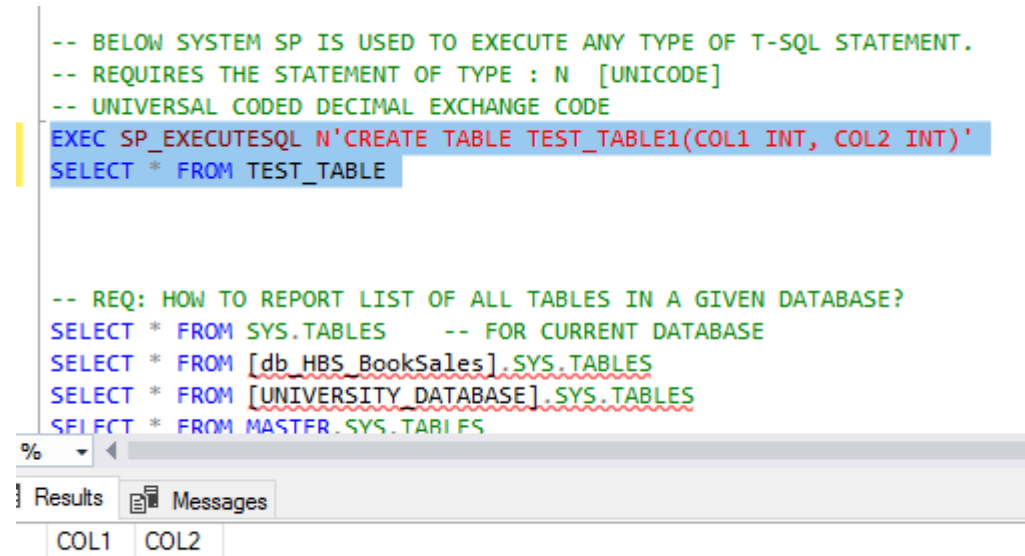
BELOW SYSTEM SP IS USED TO EXECUTE ANY TYPE OF T-SQL STATEMENT.

-- REQUIRES THE STATEMENT OF TYPE : N [UNICODE]

-- UNIVERSAL CODED DECIMAL EXCHANGE CODE

HOW TO CREATE A TABLE USING SYSTEM PROCEDURES?

```
EXEC SP_EXECUTESQL N'CREATE TABLE TEST_TABLE(COL1 INT, COL2 INT)'
```



```
SELECT * FROM TEST_TABLE
```

REQ: HOW TO REPORT LIST OF ALL TABLES IN A GIVEN DATABASE?

```
SELECT * FROM SYS.TABLES -- FOR CURRENT DATABASE
```

```
SELECT * FROM [db_HBS_BookSales].SYS.TABLES
```

```
SELECT * FROM [UNIVERSITY_DATABASE].SYS.TABLES
```

```
SELECT * FROM MASTER.SYS.TABLES
```

```
REQUIRED FORMAT: SELECT * FROM ???????.SYS.TABLES
```

REQUIREMENT:

CREATE A PROCEDURE TO GET DATABASE NAME AS INPUT AND REPORT ALL THE TABLES IN THE DATABASE?

```
CREATE PROC USP_REP_TABLES @DB_NAME VARCHAR(30) = MASTER
AS
BEGIN
DECLARE @QUERY NVARCHAR(100)
SET @QUERY = 'SELECT * FROM '+@DB_NAME+'.SYS.TABLES'
EXEC SP_EXECUTESQL @QUERY
END
```

HOW TO EXECUTE ABOVE USER SP?

```
EXEC USP_REP_TABLES -- DEFAULT PARAMETER VALUE IS CONSIDERED.
```

```
EXEC USP_REP_TABLES 'Banking' --database name
```

```
-- ISSUE:
```

```
EXEC USP_REP_TABLES 'MASTERS_TABLE' -- ERROR
```

HOW TO REPORT A CUSTOMIZED MESSAGE IN CASE OF ANY ERROR?

```
ALTER PROC USP_REP_TABLES @DB_NAME VARCHAR(30) = MASTER
AS
BEGIN
BEGIN TRY -- MEANS, IF THERE IS ANY ERROR INSIDE THE TRY BLOCK THEN
CATCH IS EXECUTED AUTOMATICALLY
DECLARE @QUERY NVARCHAR(100)
SET @QUERY = 'SELECT * FROM '+@DB_NAME+'.SYS.TABLES'
EXEC SP_EXECUTESQL @QUERY
END TRY
BEGIN CATCH
PRINT 'INVALID DATABASE NAME SUPPLIED.'
END CATCH
END
```

HOW TO REPORT A CUSTOMIZED MESSAGE IN CASE OF ANY ERROR + ACTUAL ERROR ALSO?

```
ALTER PROC USP_REP_TABLES @DB_NAME VARCHAR(30) = MASTER
AS
BEGIN
BEGIN TRY  -- MEANS, IF THERE IS ANY ERROR INSIDE THE TRY BLOCK THEN
CATCH IS EXECUTED AUTOMATICALLY
DECLARE @QUERY NVARCHAR(100)
SET @QUERY = 'SELECT * FROM '+@DB_NAME+'.SYS.TABLES'
EXEC SP_EXECUTESQL @QUERY
END TRY
BEGIN CATCH
PRINT 'INVALID DATABASE NAME SUPPLIED.'
;THROW  -- FROM SQL SERVER 2012 VERSION. USED TO REPORT THE REASON FOR
ERROR.
END CATCH
END
EXEC USP_REP_TABLES 'MASTERS_TABLE'  -- ERROR
```

TASKS >>>

1. Write a Stored Procedure to generate alpha numeric sequence of values with a dynamic prefix?
2. What is Grouping of Procedures? Advantages? Explain with an example?
3. When to RECOMPILE a procedure and what are the options to do this?
4. What are nested Stored Procedures? How they impact Performance of query execution?

TRANSACTIONS: