ASSIGNMENT 4

Database Migration

Caitlin Ross
#040750891
CST2355: Database Systems
Section 451

Contents

Process Focus

Choice of Approach: I will be using reverse engineering because the assignment explicitly states that some data and surrogate keys are missing and that the relationships are incorrect. Forward engineering would be useful if I had a copy of the complete database and the redesign process was relatively minor. However, this project calls for an overhaul of the entire design and structure, so I will need to use reverse engineering.

Step One: Not applicable, since the choice itself doesn't change which approach I choose.

Step Two: If I were to use forward engineering, it would theoretically be easy enough to derive a model based on the design diagram provided. Using reverse engineering, I will be looking at how the current database is structured and creating a completely new model that builds on that structure.

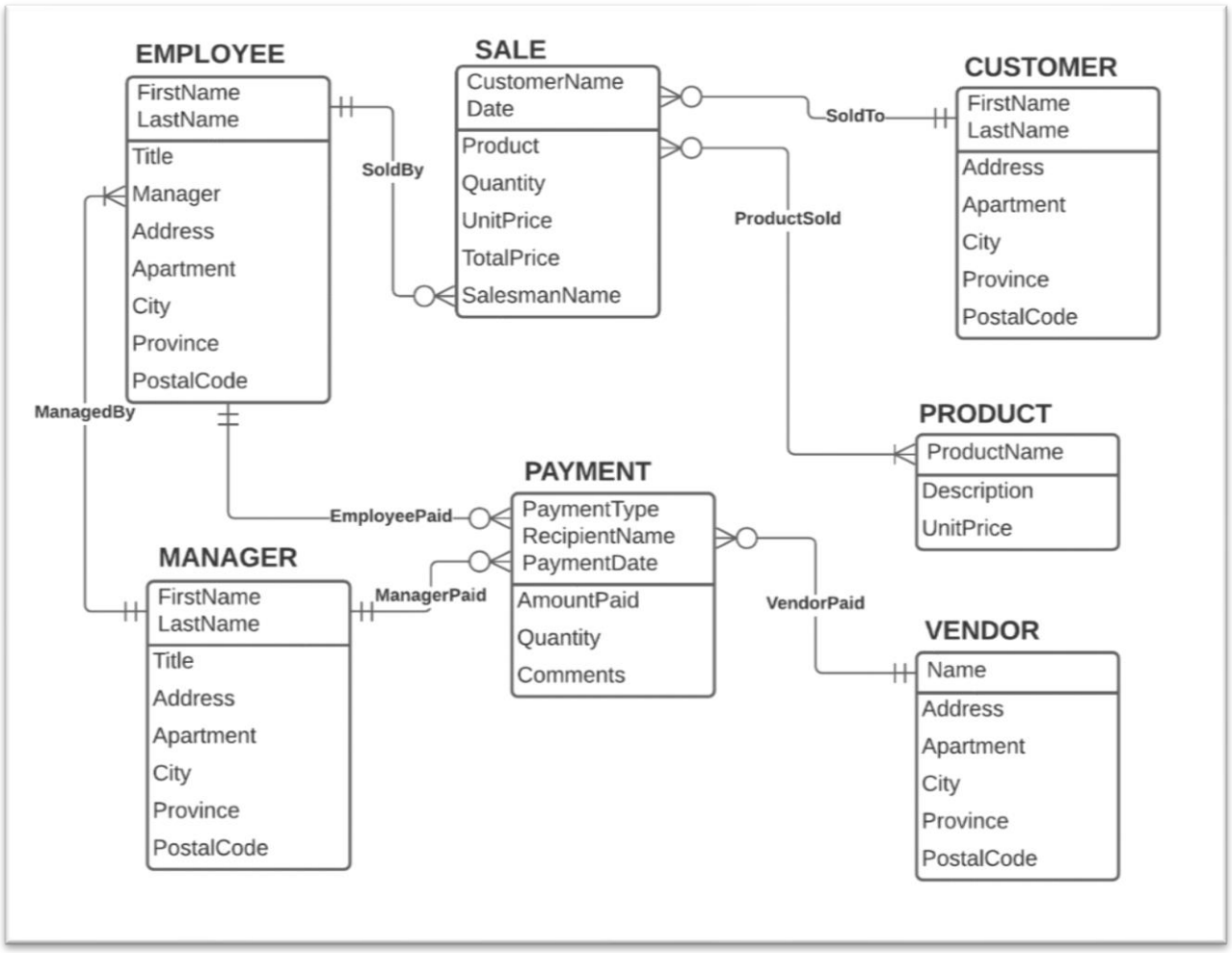
Step Three: The process used to create backups and restore them is the same for either approach. The main difference would be the number and structure of tables needing to be exported/imported.

Step Four: This step would be carried out the same either way, but the results of this step would look quite different. In a forward engineering scenario, the charts and diagrams would be very similar to the original design simply because the model from step two would be very similar to the original. In a reverse engineering situation, the charts and diagrams will be based on the model created in step two.

Step Five: Data cleanup would be minimal in a forward engineering situation. Since the data is non-existent and the current design is incomplete, the data cleanup required is assumed to be extensive. This is the main factor behind choosing reverse engineering.

Step Six: Like step four, the creation of constraints, relationships and views would be carried out in the same fashion for either approach. For the same reason, the result would differ.

Data Model



**Note: MANAGER is only separated from EMPLOYEE because of how the original database is structured. Also, original database doesn't keep track of which VENDORs sell which PRODUCTs. The final design will differ from this model.*

Database Backup

**Note: All walkthroughs below assume that the user is connected the database already. The Backup explanations all create the relevant CSV files (rather than backing up the databases as a whole) because despite being more time-consuming, using CSV files is more portable and robust. The Restore explanations assume that CSV files and relevant tables have already been created.*

MySQL Backup

1. Right-click on the table to be exported and choose Table Data Export Wizard.
2. Ensuring that the correct table is selected and all columns are checked, click Next.
3. Click Browse to either choose a file to export the data to or to create a new file. Click Save to close the Save File dialogue.
4. Ensure that csv is the option chosen and verify if the csv options are acceptable. Click Next.
5. Click Next to begin the export.
6. Click Next to view the results.
7. Click Finish.
8. Repeat steps 1-7 for each table.

MySQL Restore

1. Right-click on the table that the data will be imported to and choose Table Data Import Wizard.
2. Click Browse, select the file to be imported, and click Open to close the Open File dialogue.
3. Click Next to go to the next step.
4. Ensure that the correct table is displayed in the Use Existing Table drop down, select Truncate Table Before Import if necessary, and click Next.
5. Verify that the columns in the CSV file are properly mapped to the table's columns and click Next.
6. Click Next to begin the import.
7. Click Next to view the results.
8. Click Finish.
9. Repeat steps 1-8 for each table.

Oracle Backup

1. In the Connections explorer, find the relevant User and view the Tables (Filtered).
2. Right-click the table and select Export.
3. Uncheck Export DDL (unless desired). Leave Export Data checked, and select csv as the Format. Click Browse to decide where to name and save the file, ensuring that .csv is at the end of the file name. Click Save to close the Save File dialogue.
4. Click Next.
5. Verify that the proper table is displayed under Database Object and that an asterisk (*) is visible under Columns. Click Next.
6. Click Finish to complete the export.
7. Repeat steps 1-6 for each table.

Oracle Restore

1. In the Connections explorer, find the relevant User and view the Tables (Filtered).
2. Right-click on the table and select Import.
3. Click Browse to open the Open File dialogue and find the correct file. Click Open to close the Open File dialogue.
4. Look over the sample data to make sure it looks right, changing any CSV options if needed. Click Next.
5. With Insert as the Import Method, click Next.
6. With all column names in the right-hand box, click Next.
7. Going through each Source Data Column in turn, make sure they are properly mapped to the destination table's columns. Click Next.
8. Click Finish to complete the import.
9. Repeat steps 1-8 for each table.

SQL Server Backup

1. In the Object Explorer, right-click on the database containing the tables to be exported. Under the Tasks menu, choose Export Data.
2. Under the Data Source drop-down menu, choose SQL Server Native Client.
3. Verify that the Server Name and Database Name are correct and click Next. (This step assumes that Windows Authentication was used when setting up the database, and that it's selected now.)
4. Under the Destination drop-down menu, choose Flat File Destination.
5. Click the Browse button next to the File Name text field.
6. Ensure that CSV is the filetype chosen (drop-down menu in the bottom right), then either type in a new file name or select an existing file to overwrite. Click Open to close the Open File dialogue.
7. If all the CSV options are acceptable, click Next.
8. With the "Copy data from one or more tables or views" option selected, click Next.
9. Select the correct table and click Next.
10. With "Run immediately" checked (and "Save SSIS package" unchecked) click Finish.
11. Click Finish to begin the export.
12. Click Close to close the Wizard.
13. Repeat steps 1-12 for each table.

SQL Server Restore

1. In the Object Explorer, right-click on the database containing the tables that the data will be imported to. Under the Tasks menu, choose Import Data.
2. Choose Flat File Source. Click Browse, ensure that CSV filetype is selected (drop-down menu in the bottom right) and find the CSV file to be imported. Click Open to close the Open File dialogue.
3. Verify the CSV settings, then click Next.
4. Check the sample data shown then click Next.
5. Under Destination, choose SQL Server Native Client. Verify the Server Name and Database Name before clicking Next. (This step assumes that Windows Authentication was used when setting up the database, and that it's selected now.)
6. Click on the destination table and type in the name of the table to import the data into. Click Edit Mappings.
7. Verify the mappings then click OK.
8. Click Next.
9. Click Next to accept the error settings.
10. With "Run immediately" checked (and "Save SSIS package" unchecked) click Finish.
11. Click Finish to begin the import.
12. Click Close to close the Wizard.
13. Repeat steps 1-12 for each table.

Database Creation and Import

Cardinality Chart

Relationships		Cardinality		
Parent	Child	Type	Min	Max
MANAGER	EMPLOYEE	Strong	O-M	1:N
EMPLOYEE	WAGE	Weak, WAGE is ID-dependent	M-O	1:N
EMPLOYEE	SALE	Weak, SALE is ID-dependent	M-O	1:N
CUSTOMER	SALE	Weak, SALE is ID-dependent	M-O	1:N
PRODUCT	SALE	Weak, SALE is ID-dependent	M-O	1:N
PRODUCT	PURCHASE_ORDER	Weak, PURCHASE_ORDER is ID-dependent	M-O	1:N
VENDOR	PURCHASE_ORDER	Weak, PURCHASE_ORDER is ID-dependent	M-O	1:N

Action Charts

Employee Managed By

MANAGER optional; EMPLOYEE required	Action on MANAGER (Parent)	Action on EMPLOYEE (Child)
Insert	None	None
Modify Key	Cannot change surrogate keys	None
Delete	Disallow for business reasons	Disallow for business reasons

Employee Paid

EMPLOYEE required; WAGE optional	Action on EMPLOYEE (Parent)	Action on WAGE (Child)
Insert	None	WAGE must go to a valid EMPLOYEE
Modify Key	Cannot change surrogate keys	Cannot change surrogate keys
Delete	Disallow for business reasons	Disallow for business reasons

Sold By Employee

EMPLOYEE required; SALE optional	Action on EMPLOYEE (Parent)	Action on SALE (Child)
Insert	None	SALE must be performed by a valid EMPLOYEE
Modify Key	Disallow for business reasons	Disallow for business reasons
Delete	Disallow for business reasons	Disallow for business reasons

Sold To Customer

CUSTOMER required; SALE optional	Action on CUSTOMER (Parent)	Action on SALE (Child)
Insert	None	SALE must match a valid CUSTOMER
Modify Key	Disallow for business reasons	Disallow for business reasons
Delete	Disallow for business reasons	Disallow for business reasons

Product Sold

PRODUCT required; SALE optional	Action on PRODUCT (Parent)	Action on SALE (Child)
Insert	None	SALE must be of a valid PRODUCT
Modify Key	Disallow for business reasons	Disallow for business reasons
Delete	Disallow for business reasons	Disallow for business reasons

Product Bought

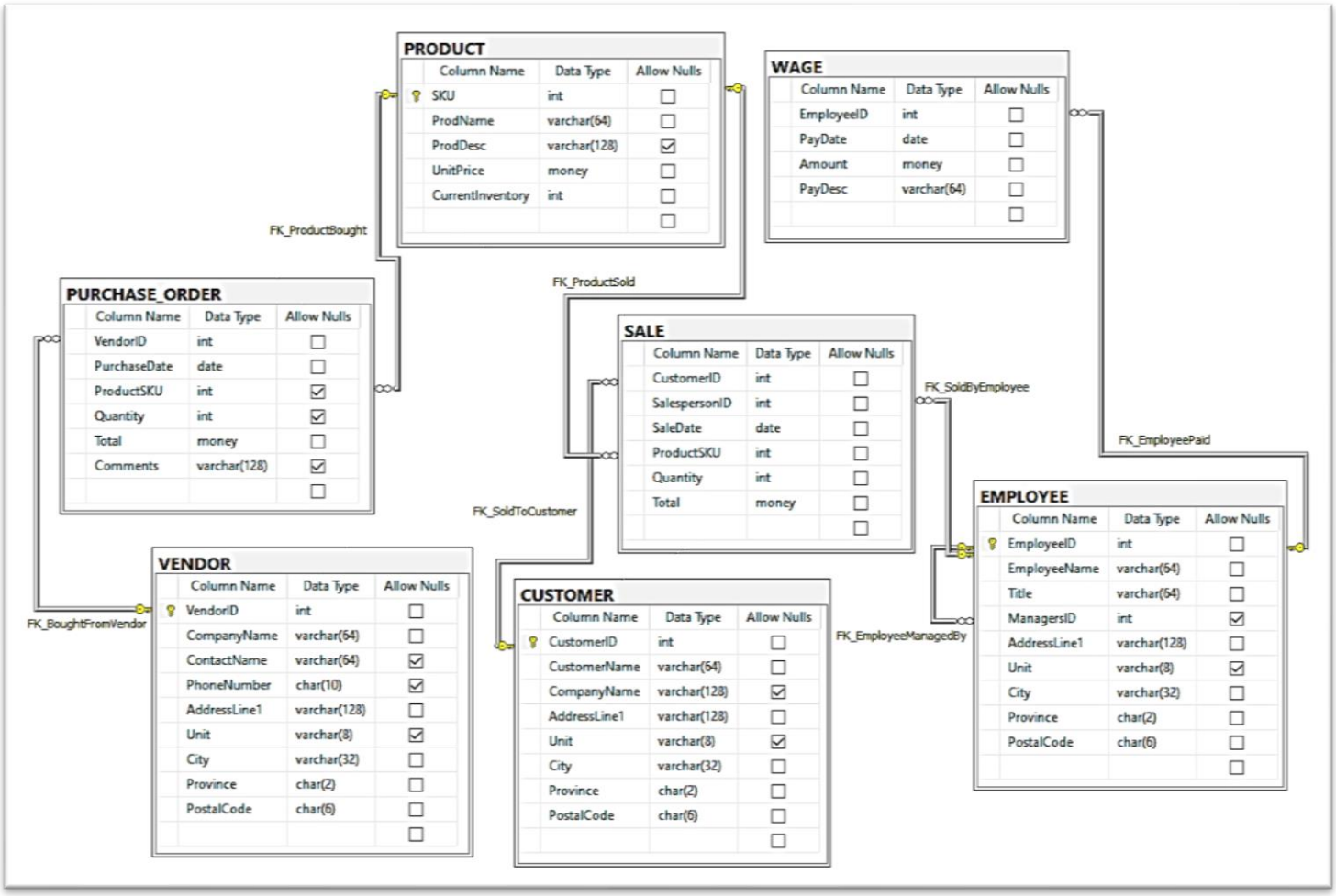
PRODUCT required; PURCHASE_ORDER optional	Action on PRODUCT (Parent)	Action on PURCHASE_ORDER (Child)
Insert	None	PURCHASE_ORDER must match a valid PRODUCT
Modify Key	Disallow for business reasons	Disallow for business reasons
Delete	Disallow for business reasons	Disallow for business reasons

Bought From Vendor

VENDOR required; PURCHASE_ORDER optional	Action on VENDOR (Parent)	Action on PURCHASE_ORDER (Child)
Insert	None	PURCHASE_ORDER must be from a valid VENDOR
Modify Key	Disallow for business reasons	Disallow for business reasons
Delete	Disallow for business reasons	Disallow for business reasons

**Note: I'm assuming that CUSTOMERs and VENDORs cannot be deleted after a transaction has occurred. For the same reason, EMPLOYEEs cannot be deleted after a WAGE has been paid and PRODUCTs cannot be deleted after a SALE has occurred. Essentially, after a transaction has taken place, no related entries can be deleted.*

Database Design Diagram



**Note: Diagram is shown after constraints have been added.*

**Note: In order to reduce complexity, I used SALE instead of INVOICE (InvoiceID, CustomerID, SalespersonID, Date, Total) and PRODUCT_SOLD (SKU, InvoiceID, Quantity), with PURCHASE_ORDER structured to match.*

**Note: I realize the original database used Salesman rather than Salesperson as a business term. In the workplace I would follow the business's terminology, but for a school assignment I feel uncomfortable using unnecessarily gendered language.*

**Note: For VENDOR, I added PhoneNumber and ContactName because that's how most businesses keep track of their suppliers. However, since the data wasn't in the original database, I'm allowing them to be null. Really, there should be PhoneNumber fields for EMPLOYEE and CUSTOMER as well. Probably emails for all three too.*

Table Characteristics

EMPLOYEE

Column ID	Constraints	Default	Oracle Datatype	SQL Server Datatype	MySQL Datatype
EmployeeID	PK		INTEGER	Int	INT
EmployeeName	NN		VARCHAR2(64)	Varchar(64)	VARCHAR(64)
Title	NN		VARCHAR2(64)	Varchar(64)	VARCHAR(64)
ManagersID	FK		INTEGER	Int	INT
AddressLine1	NN		VARCHAR2(128)	Varchar(128)	VARCHAR(128)
Unit			VARCHAR2(8)	Varchar(8)	VARCHAR()
City	NN		VARCHAR2(32)	Varchar(32)	VARCHAR(32)
Province	NN		CHAR2(2)	Char(2)	CHAR(2)
PostalCode	NN		CHAR2(6)	Char(6)	CHAR(6)

CUSTOMER

Column ID	Constraints	Default	Oracle Datatype	SQL Server Datatype	MySQL Datatype
CustomerID	PK		INTEGER	Int	INT
CustomerName	NN		VARCHAR2(64)	Varchar(64)	VARCHAR(64)
CompanyName			VARCHAR2(128)	Varchar(128)	VARCHAR(128)
AddressLine1	NN		VARCHAR2(128)	Varchar(128)	VARCHAR(128)
Unit			VARCHAR2(8)	Varchar(8)	VARCHAR()
City	NN		VARCHAR2(32)	Varchar(32)	VARCHAR(32)
Province	NN		CHAR2(2)	Char(2)	CHAR(2)
PostalCode	NN		CHAR2(6)	Char(6)	CHAR(6)

VENDOR

Column ID	Constraints	Default	Oracle Datatype	SQL Server Datatype	MySQL Datatype
VendorID	PK		INTEGER	Int	INT
CompanyName	NN		VARCHAR2(128)	Varchar(128)	VARCHAR(128)
ContactName			VARCHAR2(64)	Varchar(64)	VARCHAR(64)
PhoneNumber			CHAR2(10)	Char(10)	CHAR(10)
AddressLine1	NN		VARCHAR2(128)	Varchar(128)	VARCHAR(128)
Unit			VARCHAR2(8)	Varchar(8)	VARCHAR()
City	NN		VARCHAR2(32)	Varchar(32)	VARCHAR(32)
Province	NN		CHAR2(2)	Char(2)	CHAR(2)
PostalCode	NN		CHAR2(6)	Char(6)	CHAR(6)

PRODUCT

Column ID	Constraints	Default	Oracle Datatype	SQL Server Datatype	MySQL Datatype
SKU	PK		INTEGER	Int	INT
ProdName	NN		VARCHAR2(64)	Varchar(64)	VARCHAR(64)
ProdDesc			VARCHAR2(128)	Varchar(128)	VARCHAR(128)
UnitPrice	NN	0.00	NUMBER(8, 2)	Money	DECIMAL(6,2)
CurrentInventory	NN	0	INTEGER	Int	INT

SALE

Column ID	Constraints	Default	Oracle Datatype	SQL Server Datatype	MySQL Datatype
CustomerID	PK.1, FK, NN		INTEGER	Int	INT
SalespersonID	FK, NN		INTEGER	Int	INT
SaleDate	PK.2, NN		DATE	Date	DATE
ProductSKU	PK.3, FK, NN		INTEGER	Int	INT
Quantity	NN	1	INTEGER	Int	INT
Total	NN	0.00	NUMBER(8, 2)	Money	DECIMAL(8,2)

PURCHASE_ORDER

Column ID	Constraints	Default	Oracle Datatype	SQL Server Datatype	MySQL Datatype
VendorID	PK.1, FK, NN		INTEGER	Int	INT
PurchaseDate	PK.2		DATE	Date	DATE
ProductSKU	PK.3, FK, NN		INTEGER	Int	INT
Quantity	NN	1	INTEGER	Int	INT
Total	NN	0.00	NUMBER(8, 2)	Money	DECIMAL(8,2)
Comments			VARCHAR2(128)	Varchar(128)	VARCHAR(128)

WAGE

Column ID	Constraints	Default	Oracle Datatype	SQL Server Datatype	MySQL Datatype
EmployeeID	PK.1, FK, NN		INTEGER	Int	INT
PayDate	PK.2, NN		DATE	Date	DATE
Amount	PK.3, NN	0.00	NUMBER(8, 2)	Money	DECIMAL(8,2)
PayDesc	NN	"Bonus"	VARCHAR2(64)	Varchar(64)	VARCHAR(64)

**Note: NN refers to Not Null, PK to Primary Key, FK to Foreign Key, AI to Auto Increment (in MySQL, Identity in SQL Server). UQ and NN are redundant for primary keys.*

**Note: WAGE.PayDescription was originally WAGE.PayPeriod, but since bonuses, commissions and expense reimbursements may not fit that model I changed it to Description.*

Table Creation Scripts

EMPLOYEE

```
1.    CREATE TABLE dbo.EMPLOYEE(
2.      EmployeeID Int,
3.      EmployeeName Varchar(64),
4.      Title Varchar(64),
5.      ManagersID Int,
6.      AddressLine1 Varchar(128),
7.      Unit Varchar(8),
8.      City Varchar(32),
9.      Province Char(2),
10.     PostalCode Char(6)
11.   );
```

CUSTOMER

```
1.    CREATE TABLE dbo.CUSTOMER(
2.      CustomerID Int,
3.      CustomerName Varchar(64),
4.      CompanyName Varchar(128),
5.      AddressLine1 Varchar(128),
6.      Unit Varchar(8),
7.      City Varchar(32),
8.      Province Char(2),
9.      PostalCode Char(6)
10.   );
```

VENDOR

```
1.    CREATE TABLE dbo.VENDOR(
2.      VendorID Int,
3.      CompanyName Varchar(128),
4.      ContactName Varchar(64),
5.      PhoneNumber Char(10),
6.      AddressLine1 Varchar(128),
7.      Unit Varchar(8),
8.      City Varchar(32),
9.      Province Char(2),
10.     PostalCode Char(6)
11.   );
```

PRODUCT

```
1.    CREATE TABLE dbo.PRODUCT(
2.      SKU Int,
3.      ProdName Varchar(64),
4.      ProdDesc Varchar(128),
5.      UnitPrice Money,
6.      CurrentInventory Int
7.    );
```

SALE

```
1.    CREATE TABLE dbo.SALE(
2.      CustomerID Int,
3.      SalespersonID Int,
4.      SaleDate Date,
5.      ProductSKU Int,
6.      Quantity Int,
7.      Total Money
8.    );
```

PURCHASE_ORDER

```
1.    CREATE TABLE dbo.PURCHASE_ORDER(
2.      VendorID Int,
3.      PurchaseDate Date,
4.      ProductSKU Int,
5.      Quantity Int,
6.      Total Money,
7.      Comments Varchar(128)
8.    );
```

WAGE

```
1.    CREATE TABLE dbo.WAGE(
2.      EmployeeID Int,
3.      PayDate Date,
4.      Amount Money,
5.      PayDesc Varchar(64)
6.    );
```

Data Cleanup

EMPLOYEE Scripts

Data Validation

Primary Key Validation: Single Column

EMPLOYEE records would be assigned a surrogate key. However, if some sort of Employee ID already exists in the data, the following script would be used to verify it.

```
1.    -- PK Validation: Check for NULL values, if found assign new ID
2.    UPDATE dbo.EMPLOYEE
3.      SET EmployeeID = 1
4.        WHERE EMPLOYEE.EmployeeID IS NULL;
5.
6.    -- PK Validation: Use a cursor to find duplicate IDs and reassign them
7.    DECLARE @MaxID Int;
8.    SELECT @MaxID = MAX(EMPLOYEE.EmployeeID) FROM EMPLOYEE;
9.    DECLARE @TempID Int;
10.   DECLARE @TempName Varchar(64);
11.   DECLARE @TempTitle Varchar(64);
12.   DECLARE @TempManager Int;
13.   DECLARE @TempAddress Varchar(128);
14.   DECLARE @TempUnit Varchar(8);
15.   DECLARE @TempCity Varchar(32);
16.   DECLARE @TempProvince Char(2);
17.   DECLARE @TempPostCode Char(6);
18.   DECLARE Cursor1CURSOR LOCAL FOR
19.     SELECT
20.       EmployeeID,
21.       EmployeeName,
22.       Title,
23.       ManagersID,
24.       AddressLine1,
25.       Unit,
26.       City,
27.       Province,
28.       PostalCode
29.     FROM dbo.EMPLOYEE
30.     WHERE EmployeeID IN (
31.       SELECT EmployeeID
32.       FROM dbo.EMPLOYEE
33.       GROUP BY EmployeeID
34.       HAVING COUNT(EmployeeID) > 1
35.     );
36.   OPEN Cursor1;
```

```
37.   FETCH NEXT FROM Cursor1
38.     INTO
39.       @TempID,
40.       @TempName,
41.       @TempTitle,
42.       @TempManager,
43.       @TempAddress,
44.       @TempUnit,
45.       @TempCity,
46.       @TempProvince,
47.       @TempPostCode;
48.   WHILE @@FETCH_STATUS = 0
49.   BEGIN
50.     SET @MaxID = @MaxID + 1;
51.     DELETE FROM dbo.EMPLOYEE
52.       WHERE
53.         EMPLOYEE.EmployeeID = @TempID AND
54.         EMPLOYEE.EmployeeName = @TempName AND
55.         EMPLOYEE.Title = @TempTitle AND
56.         EMPLOYEE.AddressLine1 = @TempAddress AND
57.         EMPLOYEE.City = @TempCity AND
58.         EMPLOYEE.Province = @TempProvince AND
59.         EMPLOYEE.PostalCode = @TempPostCode;
60.     INSERT INTO dbo.EMPLOYEE
61.        VALUES (
62.        @MaxID,
63.        @TempName,
64.        @TempTitle,
65.        @TempManager,
66.        @TempAddress,
67.        @TempUnit,
68.        @TempCity,
69.        @TempProvince,
70.        @TempPostCode
71.       );
72.     FETCH NEXT FROM Cursor1
73.       INTO
74.         @TempID,
75.         @TempName,
76.         @TempTitle,
77.         @TempManager,
78.         @TempAddress,
79.         @TempUnit,
80.         @TempCity,
81.         @TempProvince,
82.         @TempPostCode;
83.   END;
84.   CLOSE Cursor1;
85.   DEALLOCATE Cursor1;
```

```
86.   -- PK Validation: Check for duplicate records with different IDs
87.   DELETE FROM dbo.EMPLOYEE
88.   WHERE EmployeeID NOT IN (
89.     SELECT MAX(EmployeeID) AS MaxRecordID
90.     FROM dbo.EMPLOYEE
91.     GROUP BY
92.       EmployeeName,
93.       Title,
94.       AddressLine1,
95.       City,
96.       Province,
97.       PostalCode
98.   );
```

Foreign Key Validation

```
1.    -- FK Validation: ManagersID
2.    UPDATE dbo.EMPLOYEE
3.      SET EMPLOYEE.ManagersID = NULL
4.      WHERE EMPLOYEE.ManagersID NOT IN
5.        (SELECT EMPLOYEE.EmployeeID FROM dbo.EMPLOYEE);
```

Uniqueness Validation

```
1.    -- Manually verify duplicate Employee names
2.    SELECT *
3.    FROM dbo.EMPLOYEE
4.    WHERE EmployeeName IN
5.      (SELECT EmployeeName
6.      FROM dbo.EMPLOYEE
7.      GROUP BY EmployeeName
8.      HAVING COUNT(EmployeeName) > 1);
```

Null/Not Null Validation

```
1.    -- NULL validation: Null values will be assigned placeholder values
2.    -- Columns contained in the Primary Key have been checked already
3.    -- Assign placeholder values where EmployeeName is NULL
4.    UPDATE dbo.EMPLOYEE
5.      SET EMPLOYEE.EmployeeName = 'Unknown Employee'
6.      WHERE EMPLOYEE.EmployeeName IS NULL;
7.
8.    -- Assign placeholder values where Title is NULL
9.    UPDATE dbo.EMPLOYEE
10.     SET EMPLOYEE.Title = 'Sales'
11.     WHERE EMPLOYEE.Title IS NULL;
12.
13.   -- Assign placeholder values where AddressLine1 is NULL
14.   UPDATE dbo.EMPLOYEE
15.     SET EMPLOYEE.AddressLine1 = 'Not Applicable'
16.     WHERE EMPLOYEE.AddressLine1 IS NULL;
17.
18.   -- Assign placeholder values where City is NULL
19.   UPDATE dbo.EMPLOYEE
20.     SET EMPLOYEE.City = 'Unknown'
21.     WHERE EMPLOYEE.City IS NULL;
22.
23.   -- Assign placeholder values where Province is NULL
24.   UPDATE dbo.EMPLOYEE
25.     SET EMPLOYEE.Province = 'UK'
26.     WHERE EMPLOYEE.Province IS NULL;
27.
28.   -- Assign placeholder values where PostalCode is NULL
29.   UPDATE dbo.EMPLOYEE
30.     SET EMPLOYEE.PostalCode = 'A0A0A0'
31.     WHERE EMPLOYEE.PostalCode IS NULL;
```

Default Values Validation

EMPLOYEE table doesn't use any default values.

Values Validation

```
1.    -- Check Province values against list of province codes
2.    -- (add UK for unknown)
3.    UPDATE dbo.EMPLOYEE
4.      SET EMPLOYEE.Province = 'UK'
5.      WHERE EMPLOYEE.Province NOT IN ('YT', 'NT', 'NU', 'BC', 'AB', 'SK',
        'MB', 'ON', 'QC', 'NB', 'PE', 'NS', 'NL', 'UK');
6.
7.    -- Check PostalCode is a valid pattern
8.    UPDATE dbo.EMPLOYEE
9.      SET EMPLOYEE.PostalCode = 'A0A0A0'
10.     WHERE EMPLOYEE.PostalCode NOT LIKE '[A-Z][0-9][A-Z][0-9][A-Z][0-9]';
```

Create Test Data

```
1.    INSERT INTO dbo.EMPLOYEE
2.     VALUES
3.       (2, 'Stan Powers', 'VP Sales', NULL, '992 Fancypants Avenue', NULL,
         'Vancouver', 'BC', 'W8B3G2'),
4.       (1, 'Bobby Wonderboy', 'Salesgnome', 2, '111 Simple Lane',
         'Basement', 'St John', 'NB', 'B0B1C3'),
5.       (1, 'Johnny Newton', 'Salesgnome', 2, '4328 Nouveau Street', NULL,
         'Chateaneuf', 'QC', 'J8E2J4'),
6.       (1, 'Johnny Newton', 'Salesgnome', 2, '4328 Nouveau Street', NULL,
         'Chateaneuf', 'QC', 'J8E2J4'),
7.       (3, 'Johnny Newton', 'Salesgnome', 2, '4328 Nouveau Street', NULL,
         'Chateaneuf', 'QC', 'J8E2J4'),
8.       (NULL, 'Casper Nonexistent', 'Absent', NULL, '00 Zero Road', NULL,
         'Nowhere', 'NN', 'N0N0N0'),
9.       (3, 'Susan Hossenpfeffer', 'Human Resources', 81, '4523 Strict
         Street', 3, 'Toronto', 'ON', 'M3K9W2');
```

CUSTOMER Scripts

Data Validation

Primary Key Validation: Single Column

CUSTOMER records will be assigned a surrogate key. However, if some sort of Customer ID already exists in the data, the following script would be used to verify it.

```
1.    -- PK Validation: Check for NULL values, if found assign new ID
2.    UPDATE dbo.CUSTOMER
3.      SET CustomerID = 1
4.        WHERE CUSTOMER.CustomerID IS NULL;
5.
6.    -- PK Validation: Use a cursor to find duplicate IDs and reassign them
7.    DECLARE @MaxID Int;
8.    SELECT @MaxID = MAX(CUSTOMER.CustomerID) FROM CUSTOMER;
9.    DECLARE @TempID Int;
10.   DECLARE @TempCustName Varchar(64);
11.   DECLARE @TempCompName Varchar(128);
12.   DECLARE @TempAddress Varchar(128);
13.   DECLARE @TempUnit Varchar(8);
14.   DECLARE @TempCity Varchar(32);
15.   DECLARE @TempProvince Char(2);
16.   DECLARE @TempPostCode Char(6);
17.   DECLARE Cursor1 CURSOR LOCAL FOR
18.     SELECT
19.       CustomerID,
20.       CustomerName,
21.       CompanyName,
22.       AddressLine1,
23.       Unit,
24.       City,
25.       Province,
26.       PostalCode
27.     FROM dbo.CUSTOMER
28.     WHERE CustomerID IN (
29.       SELECT CustomerID
30.       FROM dbo.CUSTOMER
31.       GROUP BY CustomerID
32.       HAVING COUNT(CustomerID) > 1
33.     );
34.   OPEN Cursor1;
```

```
35.   FETCH NEXT FROM Cursor1
36.     INTO
37.       @TempID,
38.       @TempCustName,
39.       @TempCompName,
40.       @TempAddress,
41.       @TempUnit,
42.       @TempCity,
43.       @TempProvince,
44.       @TempPostCode;
45.   WHILE @@FETCH_STATUS = 0
46.   BEGIN
47.     SET @MaxID = @MaxID + 1;
48.     DELETE FROM dbo.CUSTOMER
49.       WHERE
50.         CUSTOMER.CustomerID = @TempID AND
51.         CUSTOMER.CustomerName = @TempCustName AND
52.         CUSTOMER.AddressLine1 = @TempAddress AND
53.         CUSTOMER.City = @TempCity AND
54.         CUSTOMER.Province = @TempProvince AND
55.         CUSTOMER.PostalCode = @TempPostCode;
56.     INSERT INTO dbo.CUSTOMER
57.       VALUES (
58.         @MaxID,
59.         @TempCustName,
60.         @TempCompName,
61.         @TempAddress,
62.         @TempUnit,
63.         @TempCity,
64.         @TempProvince,
65.         @TempPostCode
66.       );
67.     FETCH NEXT FROM Cursor1
68.       INTO
69.         @TempID,
70.         @TempCustName,
71.         @TempCompName,
72.         @TempAddress,
73.         @TempUnit,
74.         @TempCity,
75.         @TempProvince,
76.         @TempPostCode;
77.   END;
78.   CLOSE Cursor1;
79.   DEALLOCATE Cursor1;
```

```
80.    -- PK Validation: Check for duplicate records with different IDs
81.    DELETE FROM dbo.CUSTOMER
82.      WHERE CustomerID NOT IN (
83.        SELECT MAX(CustomerID) AS MaxRecordID
84.        FROM dbo.CUSTOMER
85.        GROUP BY
86.          CustomerName,
87.          AddressLine1,
88.          City,
89.          Province,
90.          PostalCode
91.      );
```

Foreign Key Validation

CUSTOMER table contains no foreign keys.

Uniqueness Validation

```
1.     -- Manually check duplicates of CustomerName
2.     SELECT *
3.     FROM dbo.CUSTOMER
4.     WHERE CustomerName IN
5.       (SELECT CustomerName
6.       FROM dbo.CUSTOMER
7.       GROUP BY CustomerName
8.       HAVING COUNT(CustomerName) > 1);
9.
10.    -- Manually check duplicates of CompanyName
11.    SELECT *
12.    FROM dbo.CUSTOMER
13.    WHERE CompanyName IN
14.      (SELECT CompanyName
15.      FROM dbo.CUSTOMER
16.      GROUP BY CompanyName
17.      HAVING COUNT(CompanyName) > 1);
```

Null/Not Null Validation

```
1.    -- NULL validation: Null values will be assigned placeholder values
2.    -- Columns contained in the Primary Key have been checked already
3.    -- Assign placeholder values where CustomerName is NULL
4.    UPDATE dbo.CUSTOMER
5.      SET CUSTOMER.CustomerName = 'Unknown Customer
6.      WHERE CUSTOMER.CustomerName IS NULL;
7.
8.    -- Assign placeholder values where AddressLine1 is NULL
9.    UPDATE dbo.CUSTOMER
10.     SET CUSTOMER.AddressLine1 = 'Not Applicable'
11.     WHERE CUSTOMER.AddressLine1 IS NULL;
12.
13.   -- Assign placeholder values where City is NULL
14.   UPDATE dbo.CUSTOMER
15.     SET CUSTOMER.City = 'Unknown'
16.     WHERE CUSTOMER.City IS NULL;
17.
18.   -- Assign placeholder values where Province is NULL
19.   UPDATE dbo.CUSTOMER
20.     SET CUSTOMER.Province = 'UK'
21.     WHERE CUSTOMER.Province IS NULL;
22.
23.   -- Assign placeholder values where PostalCode is NULL
24.   UPDATE dbo.CUSTOMER
25.     SET CUSTOMER.PostalCode = 'A0A0A0'
26.     WHERE CUSTOMER.PostalCode IS NULL;
```

Default Values Validation

CUSTOMER table doesn't use any default values.

Values Validation

```
1.    -- Check Province values against list of province codes
2.    -- (add UK for unknown)
3.    UPDATE dbo.CUSTOMER
4.      SET CUSTOMER.Province = 'UK'
5.      WHERE CUSTOMER.Province NOT IN ('YT', 'NT', 'NU', 'BC', 'AB', 'SK',
        'MB', 'ON', 'QC', 'NB', 'PE', 'NS', 'NL', 'UK');
6.
7.    -- Check PostalCode is a valid pattern
8.    UPDATE dbo.CUSTOMER
9.      SET CUSTOMER.PostalCode = 'A0A0A0'
10.     WHERE CUSTOMER.PostalCode NOT LIKE '[A-Z][0-9][A-Z][0-9][A-Z][0-9]';
```

Create Test Data

```
1.    INSERT INTO dbo.CUSTOMER
2.      VALUES ('Sally', 'Commons', '2124 Everywhere Road', NULL, 'Hamilton',
        'ON', 'L2D4J2', NULL);
```

VENDOR Scripts

Data Validation

Primary Key Validation: Single Column

VENDOR records will be assigned a surrogate key. However, if some sort of Vendor ID already exists in the date, the following script would be used to verify it.

```
1.    -- PK Validation: Check for NULL values, if found assign new ID
2.    UPDATE dbo.VENDOR
3.      SET VendorID = 1
4.      WHERE VENDOR.VendorID IS NULL;
5.
6.    -- PK Validation: Use a cursor to find duplicate IDs and reassign them
7.    DECLARE @MaxID Int;
8.    SELECT @MaxID = MAX(VENDOR.VendorID) FROM VENDOR;
9.    DECLARE @TempID Int;
10.   DECLARE @TempCompName Varchar(128);
11.   DECLARE @TempContName Varchar(64);
12.   DECLARE @TempPhone Char(10);
13.   DECLARE @TempAddress Varchar(128);
14.   DECLARE @TempUnit Varchar(8);
15.   DECLARE @TempCity Varchar(32);
16.   DECLARE @TempProvince Char(2);
17.   DECLARE @TempPostCode Char(6);
18.   DECLARE Cursor1 CURSOR LOCAL FOR
19.     SELECT
20.       VendorID,
21.       CompanyName,
22.       ContactName,
23.       PhoneNumber
24.       AddressLine1,
25.       Unit,
26.       City,
27.       Province,
28.       PostalCode
29.     FROM dbo.VENDOR
30.     WHERE VendorID IN (
31.       SELECT VendorID
32.       FROM dbo.VENDOR
33.       GROUP BY VendorID
34.       HAVING COUNT(VendorID) > 1
35.     );
36.   OPEN Cursor1;
```

```
37.   FETCH NEXT FROM Cursor1
38.     INTO
39.       @TempID,
40.       @TempCompName,
41.       @TempContName,
42.       @TempPhone,
43.       @TempAddress,
44.       @TempUnit,
45.       @TempCity,
46.       @TempProvince,
47.       @TempPostCode;
48.   WHILE @@FETCH_STATUS = 0
49.   BEGIN
50.     SET @MaxID = @MaxID + 1;
51.     DELETE FROM dbo.VENDOR
52.       WHERE
53.         VENDOR.VendorID = @TempID AND
54.         VENDOR.CompanyName = @TempCompName AND
55.         VENDOR.AddressLine1 = @TempAddress AND
56.         VENDOR.City = @TempCity AND
57.         VENDOR.Province = @TempProvince AND
58.         VENDOR.PostalCode = @TempPostCode;
59.     INSERT INTO dbo.VENDOR
60.       VALUES (
61.         @MaxID,
62.         @TempCompName,
63.         @TempContName,
64.         @TempPhone,
65.         @TempAddress,
66.         @TempUnit,
67.         @TempCity,
68.         @TempProvince,
69.         @TempPostCode
70.       );
71.     FETCH NEXT FROM Cursor1
72.       INTO
73.         @TempID,
74.         @TempCompName,
75.         @TempContName,
76.         @TempPhone,
77.         @TempAddress,
78.         @TempUnit,
79.         @TempCity,
80.         @TempProvince,
81.         @TempPostCode;
82.   END;
83.   CLOSE Cursor1;
84.   DEALLOCATE Cursor1;
85.
```

```
86.   -- PK Validation: Check for duplicate records with different IDs
87.   DELETE FROM dbo.VENDOR
88.     WHERE VendorID NOT IN (
89.       SELECT MAX(VendorID) AS MaxRecordID
90.       FROM dbo.VENDOR
91.       GROUP BY
92.         CompanyName,
93.         AddressLine1,
94.         City,
95.         Province,
96.         PostalCode
97.     );
```

Foreign Key Validation

VENDOR Table contains no foreign keys.

Uniqueness Validation

```
1.    -- Manually check duplicates of CompanyName
2.    SELECT *
3.    FROM dbo.VENDOR
4.    WHERE CompanyName IN
5.      (SELECT CompanyName
6.      FROM dbo.VENDOR
7.      GROUP BY CompanyName
8.      HAVING COUNT(CompanyName) > 1);
9.
10.   -- Manually check duplicates of ContactName
11.   SELECT *
12.   FROM dbo.VENDOR
13.   WHERE ContactName IN
14.     (SELECT ContactName
15.     FROM dbo.VENDOR
16.     GROUP BY ContactName
17.     HAVING COUNT(ContactName) > 1);
```

Null/Not Null Validation

```
1.    -- NULL validation: Null values will be assigned placeholder values
2.    -- Columns contained in the Primary Key have been checked already
3.    -- Assign placeholder values where CompanyName is NULL
4.    UPDATE dbo.VENDOR
5.      SET VENDOR.CompanyName = 'Unknown Vendor
6.      WHERE VENDOR. CompanyName IS NULL;
7.
8.    -- Assign placeholder values where AddressLine1 is NULL
9.    UPDATE dbo.VENDOR
10.     SET VENDOR.AddressLine1 = 'Not Applicable'
11.     WHERE VENDOR.AddressLine1 IS NULL;
12.
13.   -- Assign placeholder values where City is NULL
14.   UPDATE dbo.VENDOR
15.     SET VENDOR.City = 'Unknown'
16.     WHERE VENDOR.City IS NULL;
17.
18.   -- Assign placeholder values where Province is NULL
19.   UPDATE dbo.VENDOR
20.     SET VENDOR.Province = 'UK'
21.     WHERE VENDOR.Province IS NULL;
22.
23.   -- Assign placeholder values where PostalCode is NULL
24.   UPDATE dbo.VENDOR
25.     SET VENDOR.PostalCode = 'A0A0A0'
26.     WHERE VENDOR.PostalCode IS NULL;
```

Default Values Validation

VENDOR table doesn't use any default values.

Values Validation

```
1.    -- Check Province values against list of province codes
2.    -- (add UK for unknown)
3.    UPDATE dbo.VENDOR
4.      SET VENDOR.Province = 'UK'
5.      WHERE VENDOR.Province NOT IN ('YT', 'NT', 'NU', 'BC', 'AB', 'SK',
        'MB', 'ON', 'QC', 'NB', 'PE', 'NS', 'NL', 'UK');
6.
7.    -- Check PostalCode is a valid pattern
8.    UPDATE dbo.VENDOR
9.      SET VENDOR.PostalCode = 'A0A0A0'
10.     WHERE VENDOR.PostalCode NOT LIKE '[A-Z][0-9][A-Z][0-9][A-Z][0-9]';
11.
12.   -- Check PhoneNumber is all numbers
13.   UPDATE dbo.VENDOR
14.     SET VENDOR.PhoneNumber = NULL
15.     WHERE VENDOR.PhoneNumber NOT LIKE
16.       '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]';
```

Create Test Data

```
1.    INSERT INTO dbo.VENDOR
2.      VALUES ('Acme Whizzbangs Inc.', 'Maria Luisa Vasquez DeSantos', NULL,
       '98 Rue des Forets', 'Ste Anne de Bellevue', 'QC', 'J0V4R8');
```

PRODUCT Scripts

Data Validation

Primary Key Validation: Single Column

PRODUCT records will be assigned a surrogate key. However, if some sort of SKU already exists in the data, the following script would be used to verify it.

```
1.    -- PK Validation: Check for NULL values, if found assign new ID
2.    UPDATE dbo.PRODUCT
3.      SET SKU = 1
4.      WHERE PRODUCT.SKU IS NULL;
5.
6.    -- PK Validation: Use a cursor to find duplicate IDs and reassign them
7.    DECLARE @MaxID Int;
8.    SELECT @MaxID = MAX(PRODUCT.SKU) FROM PRODUCT;
9.    DECLARE @TempID Int;
10.   DECLARE @TempName Varchar(64);
11.   DECLARE @TemDesc Varchar(128);
12.   DECLARE @TempPrice Money;
13.   DECLARE @TempInventory Int;
14.   DECLARE Cursor1 CURSOR LOCAL FOR
15.     SELECT
16.       SKU,
17.       ProdName,
18.       ProdDesc,
19.       UnitPrice,
20.       CurrentInventory
21.     FROM dbo.PRODUCT
22.     WHERE SKU IN (
23.       SELECT SKU
24.       FROM dbo.PRODUCT
25.       GROUP BY SKU
26.       HAVING COUNT(SKU) > 1
27.     );
28.   OPEN Cursor1;
29.   FETCH NEXT FROM Cursor1
30.     INTO
31.       @TempID,
32.       @TempName,
33.       @TempDesc,
34.       @TempPrice,
35.       @TempInventory;
```

```
36.   WHILE @@FETCH_STATUS = 0
37.   BEGIN
38.     SET @MaxID = @MaxID + 1;
39.     DELETE FROM dbo.PRODUCT
40.       WHERE
41.         PRODUCT.SKU = @TempID AND
42.         PRODUCT.ProdName = @TempName AND
43.         PRODUCT.UnitPrice = @TempPrice AND
44.         PRODUCT.CurrentInventory = @TempInventory;
45.     INSERT INTO dbo.PRODUCT
46.       VALUES (
47.         @MaxID,
48.         @TempName,
49.         @TempDesc,
50.         @TempPrice,
51.         @TempInventory
52.       );
53.     FETCH NEXT FROM Cursor1
54.     INTO
55.     @TempID,
56.     @TempName,
57.     @TempDesc,
58.     @TempPrice,
59.     @TempInventory;
60.   END;
61.   CLOSE Cursor1;
62.   DEALLOCATE Cursor1;
63.
64.   -- PK Validation: Check for duplicate records with different IDs
65.   DELETE FROM dbo.PRODUCT
66.     WHERE SKU NOT IN (
67.       SELECT MAX(SKU) AS MaxRecordID
68.       FROM dbo.PRODUCT
69.       GROUP BY
70.         ProdName,
71.         ProdDesc,
72.         UnitPrice
73.     );
```

Foreign Key Validation

PRODUCT Table contains no foreign keys.

Uniqueness Validation

```
1.    -- Manually check duplicates of ProdName
2.    SELECT *
3.    FROM dbo.PRODUCT
4.    WHERE ProdName IN
5.      (SELECT ProdName
6.      FROM dbo.PRODUCT
7.      GROUP BY ProdName
8.      HAVING COUNT(ProdName) > 1);
```

Null/Not Null Validation

```
1.    -- NULL validation: Null values will be assigned placeholder values
2.    -- Columns contained in the Primary Key have been checked already
3.    -- Verify that all ProdName values are NOT NULL
4.    UPDATE dbo.PRODUCT
5.      SET PRODUCT.ProdName = 'Unknown Product'
6.      WHERE PRODUCT.ProdName IS NULL;
7.
8.    -- Verify that all UnitPrice values are NOT NULL
9.    UPDATE dbo.PRODUCT
10.     SET PRODUCT.UnitPrice = 0.00
11.     WHERE PRODUCT.UnitPrice IS NULL;
12.
13.   -- Verify that all CurrentInventory values are NOT NULL
14.   UPDATE dbo.PRODUCT
15.     SET PRODUCT.CurrentInventory = 0
16.     WHERE PRODUCT.CurrentInventory IS NULL;
```

Default Values Validation

Data that doesn't contain values for the columns with default values would be assigned those default values in the Null/Not Null validation step. This would apply to the UnitPrice and CurrentInventory columns.

Values Validation

```
1.    -- Check that UnitPrice is non-negative
2.    UPDATE dbo.PRODUCT
3.      SET PRODUCT.UnitPrice = 0.00
4.      WHERE PRODUCT.UnitPrice < 0.00;
5.
6.    -- Check that CurrentInventory is non-negative
7.    UPDATE dbo.PRODUCT
8.      SET PRODUCT.CurrentInventory = 0
9.      WHERE PRODUCT.CurrentInventory < 0;
```

Create Test Data

```
1.    INSERT INTO dbo.PRODUCT
2.     VALUES ('Thingamajigger', 'This type is blue and very large', 46.99,
       3);
```

SALE Scripts

Data Validation

Primary Key Validation: Multicolumn

```
1.    -- Check for NULL CustomerID values, create and assign placeholder
      value
2.    SELECT *
3.      FROM dbo.SALE
4.      WHERE SALE.CustomerID IS NULL;
5.    IF @@ROWCOUNT > 0
6.    BEGIN
7.      DECLARE @NewID Int;
8.      SELECT CustomerID
9.        FROM dbo.CUSTOMER
10.       WHERE CustomerName = 'Unknown Customer';
11.     IF @@ROWCOUNT = 0
12.     BEGIN
13.       SELECT @NewID = MAX(CUSTOMER.CustomerID) FROM dbo.CUSTOMER;
14.       SET @NewID = @NewID + 1;
15.       INSERT INTO dbo.CUSTOMER
16.         VALUES (@NewID, 'Unknown Customer', NULL, 'Not Applicable', NULL,
          'Unknown', 'UK', 'A0A0A0');
17.     END;
18.     ELSE
19.       SELECT @NewID = MAX(CUSTOMER.CustomerID) FROM dbo.CUSTOMER WHERE
          CustomerName = 'Unknown Customer';
20.     UPDATE dbo.SALE
21.       SET CustomerID = @NewID
22.       WHERE CustomerID IS NULL;
23.   END;
24.
25.   -- Check for NULL ProductSKU values, create and assign placeholder
      value
26.   SELECT * FROM dbo.SALE WHERE SALE.ProductSKU IS NULL;
27.   IF @@ROWCOUNT > 0
28.   BEGIN
29.     DECLARE @NewID Int;
30.     SELECT SKU FROM dbo.PRODUCT WHERE ProdName = 'Unknown Product';
31.     IF @@ROWCOUNT = 0
32.     BEGIN
33.       SELECT @NewID = MAX(PRODUCT.SKU) FROM dbo.PRODUCT;
34.       SET @NewID = @NewID + 1;
35.       INSERT INTO dbo.PRODUCT
36.         VALUES (@NewID, 'Unknown Product', 'This is a placeholder', 0.00,
          0);
37.     END;
38.     ELSE
39.       SELECT @NewID = MAX(PRODUCT.SKU) FROM dbo.PRODUCT WHERE ProdName =
          'Unknown Product';
```

```
40.     UPDATE dbo.SALE
41.       SET ProductSKU = @NewID
42.       WHERE ProductSKU IS NULL;
43.   END;
44.
45.   -- Check for NULL SaleDate values, assign placeholder value
46.   SELECT * FROM dbo.SALE WHERE SALE.SaleDate IS NULL;
47.   IF @@ROWCOUNT > 0
48.   BEGIN
49.     UPDATE dbo.SALE
50.       SET SaleDate = '0001-01-01'
51.       WHERE SALE.SaleDate IS NULL;
52.   END;
53.
54.   -- PK Validation: Find duplicate PKs so user can manually validate
55.   SELECT CustomerID, SalespersonID, ProductSKU, SaleDate
56.   FROM dbo.SALE
57.   GROUP BY CustomerID, SalespersonID, ProductSKU, SaleDate
58.   HAVING COUNT(CustomerID) > 1 AND
59.     COUNT(SalespersonID) > 1 AND
60.     COUNT(ProductSKU) > 1 AND
61.     COUNT(SaleDate) > 1;
```

Foreign Key Validation

```
1.    -- FK Validation: SalespersonID
2.    UPDATE dbo.SALE
3.      SET SALE.SalespersonID = NULL
4.      WHERE SALE.SalespersonID NOT IN
5.        (SELECT EMPLOYEE.EmployeeID FROM dbo.EMPLOYEE);
6.
7.    -- FK Validation: CustomerID
8.    UPDATE dbo.SALE
9.      SET SALE.CustomerID = NULL
10.     WHERE SALE.CustomerID NOT IN
11.       (SELECT CUSTOMER.CustomerID FROM dbo.CUSTOMER);
12.
13.   -- FK Validation: ProductSKU
14.   UPDATE dbo.SALE
15.     SET SALE.ProductSKU = NULL
16.     WHERE SALE.ProductSKU NOT IN
17.       (SELECT PRODUCT.SKU FROM dbo.PRODUCT);
```

Uniqueness Validation

No single column needs to be unique in the SALE table.

Null/Not Null Validation

```
1.    -- NULL validation: Null values will be assigned placeholder values
2.    -- Columns contained in the Primary Key have been checked already
3.    -- Check for NULL SalespersonID values, create and assign placeholder
      value
4.    SELECT * FROM dbo.SALE WHERE SALE.SalespersonID IS NULL;
5.    IF @@ROWCOUNT > 0
6.    BEGIN
7.      DECLARE @NewID Int;
8.      SELECT EmployeeID
9.        FROM dbo.EMPLOYEE
10.       WHERE EmployeeName = 'Unknown Employee;
11.     IF @@ROWCOUNT = 0
12.     BEGIN
13.       SELECT @NewID = MAX(EMPLOYEE.EmployeeID) FROM dbo.EMPLOYEE;
14.       SET @NewID = @NewID + 1;
15.       INSERT INTO dbo.EMPLOYEE
16.         VALUES (@NewID, 'Unknown Salesperson', 'Sales', NULL, 'Not
            Applicable', NULL, 'Unknown', 'UK', 'A0A0A0');
17.     END;
18.     ELSE
19.       SELECT @NewID = MAX(EMPLOYEE.EmployeeID) FROM dbo.EMPLOYEE WHERE
          EmployeeName = 'Unknown Salesperson';
20.     UPDATE dbo.SALE
21.       SET SalespersonID = @NewID
22.       WHERE SalespersonID IS NULL;
23.   END;
24.
25.   -- Verify that all Quantity values are NOT NULL
26.   UPDATE dbo.SALE
27.     SET SALE.Quantity = 0
28.     WHERE SALE.Quantity IS NULL;
29.
30.   -- Verify that all Total values are NOT NULL
31.   UPDATE dbo.SALE
32.     SET SALE.Total = 0.00
33.     WHERE SALE.Total IS NULL;
```

Default Values Validation

Data that doesn't contain values for the columns with default values would be assigned those default values in the Null/Not Null validation step. This would apply to the Quantity and Total columns.

Values Validation

```
1.    -- Check that SaleDate is not in the future
2.    UPDATE dbo.SALE
3.      SET SALE.SaleDate = CONVERT(date, GETDATE())
4.      WHERE SALE.SaleDate > CONVERT(date, GETDATE());
5.
6.    -- Check that Quantity is non-negative
7.    UPDATE dbo.SALE
8.      SET SALE.Quantity = 0
9.      WHERE SALE.Quantity < 0;
10.
11.   -- Check that Total is non-negative
12.   UPDATE dbo.SALE
13.     SET SALE.Total = 0.00
14.     WHERE SALE.Total < 0.00;
```

Create Test Data

```
1.    INSERT INTO dbo.SALE
2.      VALUES
3.        (1, 1, '02-21-2019', 1, 2, 93.98),
4.        (NULL, NULL, NULL, NULL, NULL, NULL),
5.        (1, 2, '03-18-2018', 1, 1, 46.99);
```

PURCHASE_ORDER Scripts

Data Validation

Primary Key Validation

```
1.    -- Check for NULL VendorID values, create and assign placeholder value
2.    SELECT * FROM dbo.PURCHASE_ORDER WHERE PURCHASE_ORDER.VendorID IS NULL;
3.    IF @@ROWCOUNT > 0
4.    BEGIN
5.      DECLARE @NewID Int;
6.      SELECT VendorID FROM dbo.VENDOR WHERE CompanyName = 'Unknown Vendor';
7.      IF @@ROWCOUNT = 0
8.      BEGIN
9.        SELECT @NewID = MAX(VENDOR.VendorID) FROM dbo.VENDOR;
10.       SET @NewID = @NewID + 1;
11.       INSERT INTO dbo.VENDOR
12.         VALUES (@NewID, 'Unknown Vendor', NULL, NULL, 'Not Applicable',
          NULL, 'Unknown', 'UK', 'A0A0A0');
13.     END;
14.     ELSE
15.       SELECT @NewID = MAX(VENDOR.VendorID) FROM dbo.VENDOR WHERE
          CompanyName = 'Unknown Vendor';
16.     UPDATE dbo.PURCHASE_ORDER
17.       SET VendorID = @NewID
18.       WHERE VendorID IS NULL;
19.   END;
20.
21.   -- Check for NULL ProductSKU values, create and assign placeholder
      value
22.   SELECT *
23.     FROM dbo.PURCHASE_ORDER
24.     WHERE PURCHASE_ORDER.ProductSKU IS NULL;
25.   IF @@ROWCOUNT > 0
26.   BEGIN
27.     DECLARE @NewID Int;
28.     SELECT SKU FROM dbo.PRODUCT WHERE ProdName = 'Unknown Product';
29.     IF @@ROWCOUNT = 0
30.     BEGIN
31.       SELECT @NewID = MAX(PRODUCT.SKU) FROM dbo.PRODUCT;
32.       SET @NewID = @NewID + 1;
33.       INSERT INTO dbo.PRODUCT
34.         VALUES (@NewID, 'Unknown Product', 'This is a placeholder', 0.00,
          0);
35.     END;
```

```
    ELSE
      SELECT @NewID = MAX(PRODUCT.SKU) FROM dbo.PRODUCT WHERE ProdName =
      'Unknown Product';
    UPDATE dbo.PURCHASE_ORDER
      SET ProductSKU = @NewID
      WHERE ProductSKU IS NULL;
  END;

  -- Check for NULL PurchaseDate values, assign placeholder value
  SELECT *
    FROM dbo.PURCHASE_ORDER
    WHERE PURCHASE_ORDER.PurchaseDate IS NULL;
  IF @@ROWCOUNT > 0
  BEGIN
    UPDATE dbo.PURCHASE_ORDER
      SET PurchaseDate = '0001-01-01'
      WHERE PURCHASE_ORDER.PurchaseDate IS NULL;
  END;

  -- PK Validation: Find duplicate PKs so user can manually validate
  SELECT VendorID, ProductSKU, PurchaseDate
  FROM dbo.PURCHASE_ORDER
  GROUP BY VendorID, ProductSKU, PurchaseDate
  HAVING COUNT(VendorID) > 1 AND
  COUNT(ProductSKU) > 1 AND
  COUNT(PurchaseDate) > 1;
```

Foreign Key Validation

```
  -- FK Validation: VendorID
  UPDATE dbo.PURCHASE_ORDER
    SET PURCHASE_ORDER.VendorID = NULL
    WHERE PURCHASE_ORDER.VendorID NOT IN
     (SELECT VENDOR.VendorID FROM dbo.VENDOR);

  -- FK Validation: ProductSKU
  UPDATE dbo.PURCHASE_ORDER
    SET PURCHASE_ORDER.ProductSKU = NULL
    WHERE PURCHASE_ORDER.ProductSKU NOT IN
      (SELECT PRODUCT.SKU FROM dbo.PRODUCT);
```

Uniqueness Validation

No single column needs to be unique in the PURCHASE_ORDER table.

Null/Not Null Validation

```
1.    -- NULL validation: Null values will be assigned placeholder values
2.    -- Columns contained in the Primary Key have been checked already
3.    -- Verify that all Quantity values are NOT NULL
4.    UPDATE dbo.PURCHASE_ORDER
5.      SET PURCHASE_ORDER.Quantity = 0
6.      WHERE PURCHASE_ORDER.Quantity IS NULL;
7.
8.    -- Verify that all Total values are NOT NULL
9.    UPDATE dbo.PURCHASE_ORDER
10.     SET PURCHASE_ORDER.Total = 0.00
11.     WHERE PURCHASE_ORDER.Total IS NULL;
```

Default Values Validation

Data that doesn't contain values for the columns with default values would be assigned those default values in the Null/Not Null validation step. This would apply to the Quantity and Total columns.

Values Validation

```
1.    -- Check that PurchaseDate is not in the future
2.    UPDATE dbo.PURCHASE_ORDER
3.      SET PURCHASE_ORDER.PurchaseDate = CONVERT(date, GETDATE())
4.      WHERE PURCHASE_ORDER.PurchaseDate > CONVERT(date, GETDATE());
5.
6.    -- Check that Quantity is non-negative
7.    UPDATE dbo.PURCHASE_ORDER
8.      SET PURCHASE_ORDER.Quantity = 0
9.      WHERE PURCHASE_ORDER.Quantity < 0;
10.
11.   -- Check that Total is non-negative
12.   UPDATE dbo.PURCHASE_ORDER
13.     SET PURCHASE_ORDER.Total = 0.00
14.     WHERE PURCHASE_ORDER.Total < 0.00;
```

Create Test Data

```
1.    INSERT INTO dbo.PURCHASE_ORDER
2.          VALUES (1, 04/12/2017, 1, 5, 115.00, NULL);
```

WAGE Scripts

Data Validation

Primary Key Validation

```
1.    -- Check for NULL EmployeeID values, create and assign placeholder
      value
2.    SELECT * FROM dbo.WAGE WHERE WAGE.EmployeeID IS NULL;
3.    IF @@ROWCOUNT > 0
4.    BEGIN
5.      DECLARE @NewID Int;
6.      SELECT EmployeeID
7.        FROM dbo.EMPLOYEE
8.        WHERE EmployeeName = 'Unknown Employee';
9.      IF @@ROWCOUNT = 0
10.     BEGIN
11.       SELECT @NewID = MAX(EMPLOYEE.EmployeeID) FROM dbo.EMPLOYEE;
12.       SET @NewID = @NewID + 1;
13.       INSERT INTO dbo.EMPLOYEE
14.         VALUES (@NewID, 'Unknown Employee', 'Sales', NULL, 'Not
            Applicable', NULL, 'Unknown', 'UK', 'A0A0A0');
15.     END;
16.     ELSE
17.       SELECT @NewID = MAX(EMPLOYEE.EmployeeID)
18.         FROM dbo.EMPLOYEE
19.         WHERE EmployeeName = 'Unknown Employee';
20.     UPDATE dbo.WAGE
21.       SET EmployeeID = @NewID
22.       WHERE EmployeeID IS NULL;
23.   END;
24.
25.   -- Check for NULL PayDate values, assign placeholder value
26.   SELECT * FROM dbo.WAGE WHERE WAGE.PayDate IS NULL;
27.   IF @@ROWCOUNT > 0
28.   BEGIN
29.     UPDATE dbo.WAGE
30.       SET PayDate = '0001-01-01'
31.       WHERE WAGE.PayDate IS NULL;
32.   END;
33.
34.   -- Check for NULL PayDesc values, assign placeholder value
35.   SELECT * FROM dbo.WAGE WHERE WAGE.PayDesc IS NULL;
36.   IF @@ROWCOUNT > 0
37.   BEGIN
38.     UPDATE dbo.WAGE
39.       SET PayDesc = 'Bonus'
40.       WHERE WAGE.PayDesc IS NULL;
41.   END;
```

```
42.    -- PK Validation: Find duplicate PKs so user can manually validate
43.    SELECT EmployeeID, PayDate, PayDesc
44.      FROM dbo.WAGE
45.      GROUP BY EmployeeID, PayDate, PayDesc
46.      HAVING
47.        COUNT(EmployeeID) > 1 AND
48.        COUNT(PayDate) > 1 AND
49.        COUNT(PayDesc) > 1;
```

Foreign Key Validation

```
1.     -- FK Validation: EmployeeID
2.     UPDATE dbo.WAGE
3.       SET WAGE.EmployeeID = NULL
4.       WHERE WAGE.EmployeeID NOT IN
5.         (SELECT EMPLOYEE.EmployeeID FROM dbo.EMPLOYEE);
```

Uniqueness Validation

No single column needs to be unique in the WAGE table.

Null/Not Null Validation

```
1.     -- NULL validation: Null values will be assigned placeholder values
2.     -- Columns contained in the Primary Key have been checked already
3.     -- Verify that all Amount values are NOT NULL
4.     UPDATE dbo.WAGE
5.       SET WAGE.Amount = 0.00
6.       WHERE WAGE.Amount IS NULL;
```

Default Values Validation

Data that doesn't contain values for the columns with default values would be assigned those default values in the Null/Not Null validation step. This would apply to the Amount column.

Values Validation

```
1.     -- Check that PayDate is not in the future
2.     UPDATE dbo.WAGE
3.       SET WAGE.PayDate = CONVERT(date, GETDATE())
4.       WHERE WAGE.PayDate > CONVERT(date, GETDATE());
5.
6.     -- Check that Amount is non-negative
7.     UPDATE dbo.WAGE
8.       SET WAGE.Amount = 0.00
9.       WHERE WAGE.Amount < 0.00;
```

Create Test Data

```
1.     INSERT INTO dbo.WAGE
2.       VALUES(1, 02/06/2019, 1834.22, 'Pay period Jan 18 2019 to Jan 31
         2019');
```

Database Structure

EMPLOYEE Table Constraints

```
1.    -- Add NOT NULL constraints to EMPLOYEE Table
2.    ALTER TABLE dbo.EMPLOYEE
3.      ALTER COLUMN EmployeeID Int NOT NULL;
4.    ALTER TABLE dbo.EMPLOYEE
5.      ALTER COLUMN EmployeeName Varchar(64) NOT NULL;
6.    ALTER TABLE dbo.EMPLOYEE
7.      ALTER COLUMN Title Varchar(64) NOT NULL;
8.    ALTER TABLE dbo.EMPLOYEE
9.      ALTER COLUMN AddressLine1 Varchar(128) NOT NULL;
10.   ALTER TABLE dbo.EMPLOYEE
11.     ALTER COLUMN City Varchar(32) NOT NULL;
12.   ALTER TABLE dbo.EMPLOYEE
13.     ALTER COLUMN Province Char(2) NOT NULL;
14.   ALTER TABLE dbo.EMPLOYEE
15.     ALTER COLUMN PostalCode Char(6) NOT NULL;
16.
17.   -- Add Primary Key to EMPLOYEE Table
18.   ALTER TABLE dbo.EMPLOYEE
19.     ADD CONSTRAINT PK_Employee
20.       PRIMARY KEY (EmployeeId);
21.
22.   -- Add Foreign Key to EMPLOYEE Table: Employee Managed By
23.   ALTER TABLE dbo.EMPLOYEE
24.     ADD CONSTRAINT FK_EmployeeManagedBy
25.       FOREIGN KEY (ManagersId)
26.         REFERENCES dbo.EMPLOYEE(EmployeeID);
```

CUSTOMER Table Constraints

```
1.    -- Add NOT NULL constraints to CUSTOMER Table
2.    ALTER TABLE dbo.CUSTOMER
3.      ALTER COLUMN CustomerID Int NOT NULL;
4.    ALTER TABLE dbo.CUSTOMER
5.      ALTER COLUMN CustomerName Varchar(64) NOT NULL;
6.    ALTER TABLE dbo.CUSTOMER
7.      ALTER COLUMN AddressLine1 Varchar(128) NOT NULL;
8.    ALTER TABLE dbo.CUSTOMER
9.      ALTER COLUMN City Varchar(32) NOT NULL;
10.   ALTER TABLE dbo.CUSTOMER
11.     ALTER COLUMN Province Char(2) NOT NULL;
12.   ALTER TABLE dbo.CUSTOMER
13.     ALTER COLUMN PostalCode Char(6) NOT NULL;
14.
15.   -- Add Primary Key to CUSTOMER Table
16.   ALTER TABLE dbo.CUSTOMER
17.     ADD CONSTRAINT PK_Customer
18.       PRIMARY KEY (CustomerId);
```

VENDOR Table Constraints

```
1.    -- Add NOT NULL constraints to VENDOR Table
2.    ALTER TABLE dbo.VENDOR
3.      ALTER COLUMN VendorID Int NOT NULL;
4.    ALTER TABLE dbo.VENDOR
5.      ALTER COLUMN CompanyName Varchar(64) NOT NULL;
6.    ALTER TABLE dbo.VENDOR
7.      ALTER COLUMN AddressLine1 Varchar(128) NOT NULL;
8.    ALTER TABLE dbo.VENDOR
9.      ALTER COLUMN City Varchar(32) NOT NULL;
10.   ALTER TABLE dbo.VENDOR
11.     ALTER COLUMN Province Char(2) NOT NULL;
12.   ALTER TABLE dbo.VENDOR
13.     ALTER COLUMN PostalCode Char(6) NOT NULL;
14.
15.   -- Add Primary Key to VENDOR Table
16.   ALTER TABLE dbo.VENDOR
17.     ADD CONSTRAINT PK_Vendor
18.       PRIMARY KEY (VendorId);
```

PRODUCT Table Constraints

```
1.    -- Add NOT NULL constraints to PRODUCT Table
2.    ALTER TABLE dbo.PRODUCT
3.      ALTER COLUMN SKU Int NOT NULL;
4.    ALTER TABLE dbo.PRODUCT
5.      ALTER COLUMN ProdName Varchar(64) NOT NULL;
6.    ALTER TABLE dbo.PRODUCT
7.      ALTER COLUMN UnitPrice Money NOT NULL;
8.    ALTER TABLE dbo.PRODUCT
9.      ALTER COLUMN CurrentInventory Int NOT NULL;
10.
11.   -- Add Default values to PRODUCT Table
12.   ALTER TABLE dbo.PRODUCT
13.     ADD CONSTRAINT DF_UnitPrice DEFAULT 0.00 FOR UnitPrice;
14.   ALTER TABLE dbo.PRODUCT
15.     ADD CONSTRAINT DF_Inventory DEFAULT 0 FOR CurrentInventory;
16.
17.   -- Add Primary Key to PRODUCT Table
18.   ALTER TABLE dbo.PRODUCT
19.     ADD CONSTRAINT PK_Product
20.       PRIMARY KEY (SKU);
```

SALE Table Constraints

```
1.    -- Add NOT NULL constraints to SALE Table
2.    ALTER TABLE dbo.SALE
3.      ALTER COLUMN CustomerID Int NOT NULL;
4.    ALTER TABLE dbo.SALE
5.      ALTER COLUMN SalespersonID Int NOT NULL;
6.    ALTER TABLE dbo.SALE
7.      ALTER COLUMN SaleDate Date NOT NULL;
8.    ALTER TABLE dbo.SALE
9.      ALTER COLUMN ProductSKU Int NOT NULL;
10.   ALTER TABLE dbo.SALE
11.     ALTER COLUMN Quantity Int NOT NULL;
12.   ALTER TABLE dbo.SALE
13.     ALTER COLUMN Total Money NOT NULL;
14.
15.   -- Add Default values to SALE Table
16.   ALTER TABLE dbo.SALE
17.     ADD CONSTRAINT DF_SaleQuantity DEFAULT 0 FOR Quantity;
18.   ALTER TABLE dbo.SALE
19.     ADD CONSTRAINT DF_SaleTotal DEFAULT 0.00 FOR Total;
20.
21.   -- Add Primary Key to SALE Table
22.   ALTER TABLE dbo.SALE
23.     ADD CONSTRAINT PK_Sale
24.       PRIMARY KEY (CustomerID, SaleDate, ProductSKU);
25.
26.   -- Add Foreign Key to SALE Table: Sold To Customer
27.   ALTER TABLE dbo.SALE
28.     ADD CONSTRAINT FK_SoldToCustomer
29.       FOREIGN KEY (CustomerID)
30.         REFERENCES dbo.CUSTOMER(CustomerID);
31.
32.   -- Add Foreign Key to SALE Table: Sold By Employee
33.   ALTER TABLE dbo.SALE
34.     ADD CONSTRAINT FK_SoldByEmployee
35.       FOREIGN KEY (SalespersonID)
36.         REFERENCES dbo.EMPLOYEE(EmployeeID);
37.
38.   -- Add Foreign Key to SALE Table: Product Sold
39.   ALTER TABLE dbo.SALE
40.     ADD CONSTRAINT FK_ProductSold
41.       FOREIGN KEY (ProductSKU)
42.         REFERENCES dbo.PRODUCT(SKU);
```

PURCHASE_ORDER Table Constraints

```
1.    -- Add NOT NULL constraints to PURCHASE_ORDER Table
2.    ALTER TABLE dbo.PURCHASE_ORDER
3.      ALTER COLUMN VendorID Int NOT NULL;
4.    ALTER TABLE dbo.PURCHASE_ORDER
5.      ALTER COLUMN PurchaseDate Date NOT NULL;
6.    ALTER TABLE dbo.PURCHASE_ORDER
7.      ALTER COLUMN ProductSKU Int NOT NULL;
8.    ALTER TABLE dbo.PURCHASE_ORDER
9.      ALTER COLUMN Quantity Int NOT NULL;
10.   ALTER TABLE dbo.PURCHASE_ORDER
11.     ALTER COLUMN Total Money NOT NULL;
12.
13.   -- Add Default values to PURCHASE_ORDER Table
14.   ALTER TABLE dbo.PURCHASE_ORDER
15.     ADD CONSTRAINT DF_PurchaseQuantity DEFAULT 0 FOR Quantity;
16.   ALTER TABLE dbo.PURCHASE_ORDER
17.     ADD CONSTRAINT DF_PurchaseTotal DEFAULT 0.00 FOR Total;
18.
19.   -- Add Primary Key to PURCHASE_ORDER Table
20.   ALTER TABLE dbo.PURCHASE_ORDER
21.   ADD CONSTRAINT PK_PurchaseOrder
22.   PRIMARY KEY (VendorID, PurchaseDate, SKU);
23.
24.   -- Add Foreign Key to PURCHASE_ORDER Table: Bought From Vendor
25.   ALTER TABLE dbo.PURCHASE_ORDER
26.   ADD CONSTRAINT FK_BoughtFromVendor
27.     FOREIGN KEY (VendorID)
28.       REFERENCES dbo.VENDOR(VendorID);
29.
30.   -- Add Foreign Key to PURCHASE_ORDER Table: Product Bought
31.   ALTER TABLE dbo.PURCHASE_ORDER
32.   ADD CONSTRAINT FK_ProductBought
33.     FOREIGN KEY (ProductSKU)
34.       REFERENCES dbo.PRODUCT(SKU);
```

WAGE Table Constraints

```
1.    -- Add NOT NULL constraints to WAGE Table
2.    ALTER TABLE dbo.WAGE
3.      ALTER COLUMN EmployeeID Int NOT NULL;
4.    ALTER TABLE dbo.WAGE
5.      ALTER COLUMN PayDate Date NOT NULL;
6.    ALTER TABLE dbo.WAGE
7.      ALTER COLUMN Amount Money NOT NULL;
8.    ALTER TABLE dbo.WAGE
9.      ALTER COLUMN PayDesc Varchar(64) NOT NULL;
10.
11.   -- Add Default values to WAGE Table
12.   ALTER TABLE dbo.WAGE
13.     ADD CONSTRAINT DF_Amount DEFAULT 0.00 FOR Amount;
14.   ALTER TABLE dbo.WAGE
15.     ADD CONSTRAINT DF_PayDescription DEFAULT 'Bonus' FOR PayDesc;
16.
17.   -- Add Primary Key to WAGE Table
18.   ALTER TABLE dbo.WAGE
19.   ADD CONSTRAINT PK_Wage
20.   PRIMARY KEY (EmployeeID, PayDate, PayDesc);
21.
22.   -- Add Foreign Key to WAGE Table: Employee Paid
23.   ALTER TABLE dbo.WAGE
24.     ADD CONSTRAINT FK_EmployeePaid
25.       FOREIGN KEY (EmployeeID)
26.         REFERENCES dbo.EMPLOYEE(EmployeeID);
```

Sales View

```
1.    CREATE VIEW dbo.VW_Sales AS
2.     SELECT
3.       custSale.CompanyName,
4.       custSale.CustomerName,
5.       prodSoldBy.ProdName,
6.       custSale.Quantity,
7.       prodSoldBy.UnitPrice,
8.       custSale.Total,
9.       custSale.AddressLine1,
10.      custSale.Unit,
11.      custSale.City,
12.      custSale.Province,
13.      custSale.PostalCode,
14.      prodSoldBy.EmployeeName
15.    FROM
16.      (SELECT
17.       CUSTOMER.CustomerID,
18.       CUSTOMER.CompanyName,
19.       CUSTOMER.CustomerName,
20.       CUSTOMER.AddressLine1,
21.       CUSTOMER.Unit,
22.       CUSTOMER.City,
23.       CUSTOMER.Province,
24.       CUSTOMER.PostalCode,
25.       SALE.Quantity,
26.       SALE.Total,
27.       SALE.ProductSKU
28.      FROM dbo.CUSTOMER INNER JOIN dbo.SALE
29.       ON CUSTOMER.CustomerID = SALE.CustomerID
30.      ) custSale
31.    INNER JOIN
32.      (SELECT
33.       PRODUCT.SKU,
34.       PRODUCT.ProdName,
35.       PRODUCT.UnitPrice,
36.       soldBy.EmployeeName
37.      FROM dbo.PRODUCT
38.      INNER JOIN
39.       (SELECT
40.         EMPLOYEE.EmployeeName,
41.         EMPLOYEE.EmployeeID,
42.         SALE.ProductSKU
43.        FROM dbo.EMPLOYEE INNER JOIN dbo.SALE
44.         ON EMPLOYEE.EmployeeID = SALE.SalespersonID
45.       ) soldBy
46.      ON PRODUCT.SKU = soldBy.ProductSKU
47.      ) prodSoldBy
48.    ON custSale.ProductSKU = prodSoldBy.SKU;
```

Managers View

```
1.    CREATE VIEW dbo.VW_Managers AS
2.     SELECT
3.      man.ManagerName,
4.      man.ManagerTitle,
5.      emp.EmployeeName,
6.      emp.Title
7.     FROM EMPLOYEE AS emp
8.      INNER JOIN
9.       (SELECT
10.        EmployeeName AS ManagerName,
11.        Title AS ManagerTitle,
12.        EmployeeID AS ManagersEmpID
13.       FROM EMPLOYEE) man
14.     ON emp.ManagersID = man.ManagersEmpID;
```

Operation Expenses View

```
1.    CREATE VIEW dbo.VW_OpExpenses AS
2.      SELECT
3.        EMPLOYEE.EmployeeName AS [Name],
4.        'Wage' AS [Payment Type],
5.        WAGE.Amount AS [Amount],
6.        WAGE.PayDate AS [Date],
7.        0 AS [Quantity],
8.        EMPLOYEE.AddressLine1 AS [Address],
9.        EMPLOYEE.Unit AS [Unit],
10.       EMPLOYEE.City AS [City],
11.       EMPLOYEE.Province AS [Province],
12.       EMPLOYEE.PostalCode AS [Postal Code],
13.       WAGE.PayDesc AS [Comments]
14.     FROM dbo.EMPLOYEE INNER JOIN dbo.WAGE
15.       ON EMPLOYEE.EmployeeID = WAGE.EmployeeID
16.     UNION
17.     SELECT
18.       VENDOR.CompanyName AS [Name],
19.       'Purchase Order' AS [PaymentType],
20.       PURCHASE_ORDER.Total AS [Amount],
21.       PURCHASE_ORDER.PurchaseDate AS [Date],
22.       PURCHASE_ORDER.Quantity AS [Quantity],
23.       VENDOR.AddressLine1 AS [Address],
24.       VENDOR.Unit AS [Unit],
25.       VENDOR.City AS [City],
26.       VENDOR.Province AS [Province],
27.       VENDOR.PostalCode AS [Postal Code],
28.       PURCHASE_ORDER.Comments AS [Comments]
29.     FROM dbo.VENDOR INNER JOIN dbo.PURCHASE_ORDER
30.       ON VENDOR.VendorID = PURCHASE_ORDER.VendorID;
```

Integrations

<Create stored procedures using transactional boundaries to ensure that data is properly added to the new database. Assume the batches are in the same format as before in ACCESS. Specify changes between MySQL, Oracle and SQL Server.>

Permissions

Permission Chart

User/Role	Data Access				
	Customers	Products	Purchases	Employees	Wages
Sales	Read/write	Read/write	Read/write	Read	No access
Operations	Read	Read	Read	Read	No access
Management	Read	Read	Read	Read/write	Read/write

**Note: I used CONTROL for read/write permissions because it implies all other permissions. I changed DENY ALL to DENY CONTROL because DENY ALL will be deprecated soon. DENY SELECT is implied by DENY CONTROL.*

Sales Role

```
1.    -- Create sales role and grant permissions
2.    CREATE ROLE sales;
3.    -- Read only on EMPLOYEE table
4.    DENY CONTROL ON dbo.EMPLOYEE TO sales;
5.    GRANT SELECT ON dbo.EMPLOYEE TO sales;
6.    -- Read/write on CUSTOMER table
7.    GRANT CONTROL ON dbo.CUSTOMER TO sales;
8.    -- Read/write on VENDOR table
9.    GRANT CONTROL ON dbo.VENDOR TO sales;
10.   -- Read/write on PRODUCT table
11.   GRANT CONTROL ON dbo.PRODUCT TO sales;
12.   -- Read/write on SALE table
13.   GRANT CONTROL ON dbo.SALE TO sales;
14.   -- Read/write on PURCHASE_ORDER table
15.   GRANT CONTROL ON dbo.PURCHASE_ORDER TO sales;
16.   -- No access on WAGE table
17.   DENY CONTROL ON dbo.WAGE TO sales;
```

Operations Role

```
1.    -- Create operations role and grant permissions
2.    CREATE ROLE operations;
3.    -- Read only on EMPLOYEE table
4.    DENY CONTROL ON dbo.EMPLOYEE TO operations;
5.    GRANT SELECT ON dbo.EMPLOYEE TO operations;
6.    -- Read only on CUSTOMER table
7.    DENY CONTROL ON dbo.CUSTOMER TO operations;
8.    GRANT SELECT ON dbo.CUSTOMER TO operations;
9.    -- Read only on VENDOR table
10.   DENY CONTROL ON dbo.VENDOR TO operations;
11.   GRANT SELECT ON dbo.VENDOR TO operations;
12.   -- Read only on PRODUCT table
13.   DENY CONTROL ON dbo.PRODUCT TO operations;
14.   GRANT SELECT ON dbo.PRODUCT TO operations;
15.   -- Read only on SALE table
16.   DENY CONTROL ON dbo.SALE TO operations;
17.   GRANT SELECT ON dbo.SALE TO operations;
18.   -- Read only on PURCHASE_ORDER table
19.   DENY CONTROL ON dbo.PURCHASE_ORDER TO operations;
20.   GRANT SELECT ON dbo.PURCHASE_ORDER TO operations;
21.   -- No access on WAGE table
22.   DENY CONTROL ON dbo.WAGE TO operations;
```

Management Role

```
1.    -- Create management role and grant permissions
2.    CREATE ROLE management;
3.    -- Read/write on EMPLOYEE table
4.    GRANT CONTROL ON dbo.EMPLOYEE TO management;
5.    -- Read only on CUSTOMER table
6.    DENY CONTROL ON dbo.CUSTOMER TO management;
7.    GRANT SELECT ON dbo.CUSTOMER TO management;
8.    -- Read only on VENDOR table
9.    DENY CONTROL ON dbo.VENDOR TO management;
10.   GRANT SELECT ON dbo.VENDOR TO management;
11.   -- Read only on PRODUCT table
12.   DENY CONTROL ON dbo.PRODUCT TO management;
13.   GRANT SELECT ON dbo.PRODUCT TO management;
14.   -- Read only on SALE table
15.   DENY CONTROL ON dbo.SALE TO management;
16.   GRANT SELECT ON dbo.SALE TO management;
17.   -- Read only on PURCHASE_ORDER table
18.   DENY CONTROL ON dbo.PURCHASE_ORDER TO management;
19.   GRANT SELECT ON dbo.PURCHASE_ORDER TO management;
20.   -- Read/write on WAGE table
21.   GRANT CONTROL ON dbo.WAGE TO management;
```

Appendix: Miscellaneous Notes and Assumptions

- At one point I tried adding an ADDRESS table to reduce repetition. Later I realized it was added work for no benefit, so I re-integrated the address fields into the EMPLOYEE, CUSTOMER and VENDOR tables. I may have missed some artifacts of those changes in this document.
- For the Primary Key validation scripts, DELETE and INSERT statements are used in the WHILE loop instead of UPDATE in case duplicate records are present. They deliberately exclude checking values of columns that can be NULL. Also, the order of the validation steps matters. Instead of using a cursor to assign values to null IDs, they are assigned a value of 1 because they will be reassigned in a later step.
- I had originally included NOT NULL and IDENTITY constraints in my table creations scripts, but I had to delete them and put the NOT NULL constraints in a later section. The IDENTITY constraints cannot be added later using ALTER TABLE, so that functionality was lost. I may have missed some artifacts of those changes in this document.