Manipulating Table Data

CMPE 2400

Databases

Adding New Records

- Recall that a record is the same as a row of data
- Be mindful of primary and foreign key relationships when inserting a record into a table
- A primary key constraint will reject duplicate column values
- A foreign key constraint will prohibit entering values into the column that do not already exist in the primary key table

INSERT INTO x

- Adding an individual record to a table will be accomplished using the following:
  - insert into allows you to choose the table that will be affected, and to choose the columns for which you will enter data.
    - The chosen columns do not need to be listed in the same order as they are listed in the table.
    - If you are going to supply values for all columns, you may omit the column list entirely.
    - If a column has a default constraint, then the value for that column can be left out of the column list, and the default value will be used in its place.

Adding New Records

- Syntax 1- Including Column names

```
insert into Database.Schema.Table
  (Col1Name, Col2Name, Col3Name, etc)
values
  (Value1, Value2, Value3, etc)
```

Example:

```
insert into username_Referees.dbo.Refs
  (RefID, FirstName, LastName)
values
  (2, 'John', 'Mills')
```

Adding New Records (Without using column names)

- Adding an individual record to a table will be accomplished using the following:
  - values allows you to specify the data that you will enter into the chosen columns.
    - When excluding column names, the data *must* appear in the same order the columns occur in the table.
    - Columns in a table can be left without a value by making it null, if that is allowed by the constraints
    - As you might imagine, you *must* supply values for all non-null columns.

Adding New Records

- Syntax 2- Excluding Column names

```
insert into Database.Schema.Table
values  (Value1,Value2,Value3, etc)
```

Example:

```
insert into username_Referees.dbo.Refs
values  (1, 'Sylvia', 'Venezia', 'Master')
```

Create a sample database

- Create a database for holding some sample tables
  - Note: You are not responsible for database / table creation at this time.

```
use master
go

drop database if exists username_Referees
go

create database username_Referees
go

use username_Referees
go
```

```
drop table if exists Refs
go

create table Refs
(
    RefID           int  not null
      constraint PK_Refs primary key clustered,
    FirstName       varchar(20),
    LastName        varchar(20),
    CertLevel       char(6)
    constraint DF_Certs default 'Master'
)
go
```

Create a sample table

Add data

- Now enter the following data
- Note that one of the first names is currently unknown

First Name	Last Name	Cert Level
Sylvia	Venezia	Master
John	Mills	Master
?	Cooper	Master
Liz	Dun	Senior

Insert first record

- Add the first record, providing values for all columns, in the order of their creation.

```
insert into username_Referees.dbo.Refs
values
        (1, 'Sylvia', 'Venezia', 'Master')
```

Insert second record

- Add the second record
- use the default value for the CertLevel of 'Master'

```
insert into username_Referees.dbo.Refs
        (RefID, FirstName, LastName)
values
        (2, 'John', 'Mills')
```

Insert third record

- For the 3rd row, the first name must be null
  - This may also be accomplished by omitting the FirstName column from the column list, and leaving out the associated value in the values list.

```
insert into username_Referees.dbo.Refs
(RefID, FirstName, LastName)

values

        (3, null, 'Cooper')
```

Insert last record

- For the last record, override the default CertLevel value, and specify the columns out of order.

```
insert into username_Referees.dbo.Refs
        (FirstName, RefID, CertLevel, LastName)
values
        ('Liz', 4, 'Senior', 'Dun')
```

identity Property

- The identity property for a primary key column causes the column values to auto-increment whenever a new record is added.
- An initial value and a step value may be defined when creating the identity column.
  - Both of these values will default to 1 if not specified.
- This property may only be applied to an integer type column.
- The identity property may only be used on a single column in a table.

identity Property

- Modify the table creation script for the Refs table so that it appears as below, renaming the table being created as MoreRefs, and changing the names of the constraints.

```
drop table if exists MoreRefs
go

create table MoreRefs
(
  RefID             int   identity(10,10)  not null
        constraint PK_MoreRefs primary key clustered,
  FirstName               varchar(20),
  LastName                varchar(20),
  CertLevel               char(6)
        constraint DF_MoreCerts default 'Master'
)
go
```

identity Property Example

- Add the following record, and then look at the table to see that the identity column has been auto-populated.
  - Try running the query multiple times and see that it doesn't break.

```
insert into username_Referees.dbo.MoreRefs
    (FirstName, LastName, CertLevel)
values
    ('Harry', 'Potter', 'Wizard')
```

Obtaining the value of an identity property

- When we insert data into a table containing an identity column, the value generated for the identity column will also be copied into the variable **@@identity**.
- We can then use this value for insertion into other tables

Adding Multiple Records

- Multiple records may be added with a single insert into statement by using a select query **in place of** the values clause of the insert into structure
- The number of columns in the insert into statement and the select query must be the same
- The data types must match
- The columns in the select query may come from multiple tables through the use of a join(s)

Adding Multiple Records

- Syntax

```
insert into Database.Schema.NewTable
  (Col1, Col2, Col3, … )

    select    Col1, Col2, Col3, …
    from      OldTable(s)
    [where FilteringExpression]
```

Inserting Multiple Records Example

- Using the same create table script as for the MoreRefs table, change the table name to EvenMoreRefs, rename the constraints, and run the script.

```
if exists
(
  select       [name]
  from  sysobjects
  where        [name] = 'EvenMoreRefs'
)
drop table EvenMoreRefs
go

create table EvenMoreRefs
(
  RefID              int     identity(10,10)     not null
        constraint PK_EvenMoreRefs primary key clustered,
  FirstName          varchar(20),
  LastName           varchar(20),
  CertLevel          char(6)
        constraint DF_EvenMoreCerts default 'Master'
)
go
```

IDENTITY

- IDENTITY is used to have the DBMS automatically create a unique id for a field
- For special purposes such as data import, IDENTITY can be temporarily disabled:
- set identity_insert username_Referees.dbo.EvenMoreRefs on
  - <insert statements with explicit values for the ID column>
- set identity_insert username_Referees.dbo.EvenMoreRefs off

- Remember: only override the IDENTITY behaviour in appropriate circumstances

Inserting Multiple Records Example

- Next, using an insert into statement with a select query, copy all 'Master' level referees from the Refs table into the EvenMoreRefs table.
- Make the RefIDs match the Refs table

```
set identity_insert username_Referees.dbo.EvenMoreRefs on

insert into EvenMoreRefs (RefID, FirstName, LastName, CertLevel)
    select    RefID, FirstName, LastName, CertLevel
    from      Refs
    where     CertLevel = 'Master'

set identity_insert username_Referees.dbo.EvenMoreRefs off
```

Inserting Multiple Records Example

- Next, get all of the records from the More Refs table, but this time, let the identity column do its job.

```
insert into username_Referees.dbo.EvenMoreRefs
        (FirstName, LastName, CertLevel)
        select      FirstName, LastName, CertLevel
        from        username_Referees.dbo.MoreRefs
```

Creating and Filling Tables

- The select … into statement may be used to create an entire table of data, and populate the table using columns from another table(s).
- Because a new table is created as part of the select … into process, check to see whether the table exists before executing.
- You may use a where clause to filter rows out of the table from which you are selecting your column.
  - If you want only the table structure but no row data, then provide a filter that can never be true. (ie. ID = null)
- You may also use a join to source your data from more than one table.

Creating and Filling Tables

```
drop table if exists TargetTable
go

select Col1, Col2, Col3, ...
into        TargetDatabase.dbo.TargetTable
from        SourceDatabase.dbo.SourceTable
[where      Filter Expression(s)]
```

Trying it Out

- Using select ... into, create a replica of the EvenMoreRefs table called GettingTiredOfRefs.
  - You should see that the table columns you included in the select portion of the statement are all present.
    - Notice that the properties of the columns are all consistent. This would include the identity property if it had been set.
    - On the other hand, notice that the constraints are all missing.
  - If you were to open the table, then you would also see that all of the data is present.

Creating and Filling Tables Example

- Using select ... into, create a replica of the EvenMoreRefs table called GettingTiredOfRefs.

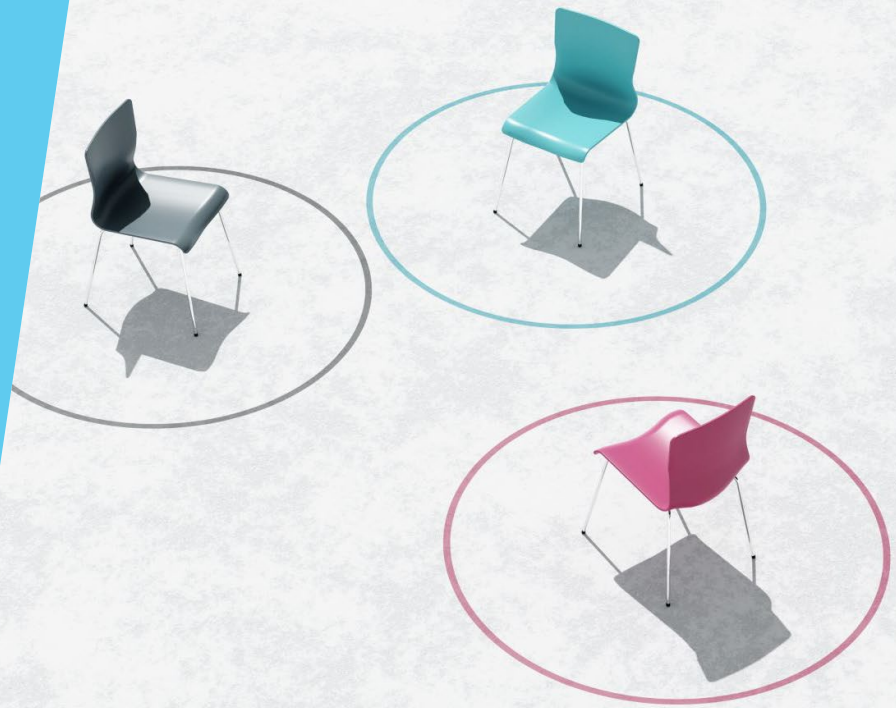
```
drop table if exists GettingTiredOfRefs
go

select      RefID, FirstName, LastName, CertLevel
into        username_Referees.dbo.GettingTiredOfRefs
from        username_Referees.dbo.EvenMoreRefs
where       CertLevel <> 'Wizard'
```

Exercise 2

- Using select Into, create a table called AdditionalEmployees in the usename_DB, we created previously, and make a copy of the EmployeeId, LastName, FirstName and HireDate of all employees from the NorthwindTraders database into this new table.

Temporary Tables

- A temporary table may be created when you do not wish the table to persist after you are finished with your current operation.
  - The table will exist for the duration of your connection to the database.
    - A procedure is considered to form its own connection to the database. Thus once the procedure ends, any temporary tables created within are automatically destroyed. (We'll talk about stored procedures later in the course)
  - In order for a table to be viewed as temporary, just preface its name with a hash (#) when it is created.



Temporary Tables

- Temporary tables may be created using the standard table creation syntax, but at this time we won't be considering it.
- For the exercises you shall see in this section, only the select … into situation will benefit from use of temporary tables

Temporary Tables

- Consider the previous example using select .. into. In order to make the target table temporary, the query should have looked as follows.

```
select   RefID, FirstName, LastName, CertLevel
into     #GettingTiredOfRefs
from     username_Referees.dbo.EvenMoreRefs
where    CertLevel <> 'Wizard'
```

Deleting and Updating Records

- The delete keyword is used to remove entire records from a table.
- The update keyword is used to modify the contents of a column within an existing record.

Deleting Table Data

- The delete statement removes zero or more entire records from a table
  - You may not use delete to remove data from only one column within a table record.
  - You may not delete records from multiple tables.
- The following statement will delete **all records** from the table GettingTiredOfRefs
  - Note that when all records are deleted, that the structure of the table itself remains unchanged.

```
delete from username_Referees.dbo.GettingTiredOfRefs
```

Deleting Table Data

- Usually when the delete statement is used, it is focused on data that fits a certain set of parameters.
- Using the where clause we may set a filter that only deletes records that meet these parameters.

```
delete
from username_Referees.dbo.EvenMoreRefs
where CertLevel = ‘Master'
```

Updating Table Data

- The update statement allows you to change the contents of a record.
- Syntax:

```
update    Database.Schema.TableName
set       ColumnName1 = NewValue1,
          ColumnName2 = NewValue2
[where    FilterExpression]
```

Updating Table Data

- Use the where clause to set a filter that only updates records that meet the specified conditions.

```
update    username_Referees.dbo.Refs
set       FirstName = 'Old Man',
          LastName = 'Winter'
where     CertLevel = 'Senior'
```

Updating Table Data

- Note that you can even perform an update on column by using the previous value in the column and performing some modification of it. We'll try it in the next exercise

Exercise3

There are two parts to this exercise:

Using a **select into** statement, create a new table called **Refs2** in your Referees database, by copying all data from Refs into this table

Write an **update** statement that replaces all the cert level values in **Refs2** by their corresponding first 3 characters only

Referential Integrity

- As you will discover, if you have not already done so, delete may fail if you attempt to remove a record from a primary key column which is referenced as a foreign key in another table.
- The reason this will fail is that if the delete is allowed from the primary key field, all values referencing the field will be in violation of the associated foreign key constraint.
- Careful: if the “cascade delete” option was specified, the child records may be automatically deleted instead of receiving an error message

Referential Integrity

- The way to solve this problem is to make sure that you code a deletion of the record in the referencing table before you attempt to delete the primary key record.

Using Transactions

- Group a number of executions together into transactions to make a block that either all gets saved, or all gets discarded
- We indicate the beginning of a transaction by a **begin transaction** statement
- **commit** is used to permanently store the changes to the database
- **rollback** is used to undo the changes

Exercise 4

Use a transaction with 2 separate Insert statements to add 2 records to your Refs table. Commit the transaction and do a select * query to verify that the records have been added.

Use a new transaction to insert 2 new records in the Refs table. In the same transaction include a select * to verify that the data has been added to the table. Then use a Rollback statement at the end of the transaction to undo the changes.

Exercise 5

- Using **Select Into**, create a table in your username_DB database containing all the product name, unit price and category name of all products from the NorthwindTraders database.