.NET PROGRAMMING

SQL PROGRAMMING & DATA MODELLING

Chia Yuen Kwan

isscyk@nus.edu.sg

AGENDA

Day	Topics
Day 1	▪ RDBMS Introduction ▪ Development Tool ▪ SQL Select
Day 2	▪ SQL Select (cont'd)
Day 3	▪ Data Manipulation Language ▪ Data Definition Language
Day 4	▪ User View ▪ Stored Procedure

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

UNDERSTANDING RELATIONAL DATABASE DESIGN

Chia Yuen Kwan

isscyk@nus.edu.sg

Topics

- **Overview of Relational Database table Concepts**
- Primary Key
- Entity Relations Diagram
- Foreign Key
- Structured Query Language (SQL)
- Summary

Elements of a RDBMS

- A schema / database is a collection of logical structures of data or objects
- Some types of objects:
  - Tables
    - stores data / records
  - indexes
  - views
  - stored procedures

Relational Database Table

- All data is stored in tables
  - Each employee record is represented by a **row** in the table and their characteristics is represented by the **columns**

Emp_No	Emp_Name	IC_No	Dept_No
179	Chang	P28493	7
857	Robinson	S95843	4
342	Bill	T04842	7

Rows

Columns

- All relationship information is presented as data values in tables
  - No ordering

Data type

- Each column in a relational database has a datatype
- Common datatypes:

Data Type	
CHAR(size)	Fixed character data
VARCHAR(size)	Variable length character
INT	Whole numbers
DECIMAL(p,s)	Number with precision
DATE	Date field

Topics

- Overview of Relational Database table Concepts
- **Primary Key**
- Entity Relations Diagram
- Foreign Key
- Structured Query Language (SQL)
- Summary

Composite Primary Key

Order

OrderID [PK]	CustomerID	OrderDate
A1091	S009	21/7/2011
A1092	S010	12/1/2011
A1093	S010	21/8/2011

Single Primary Key

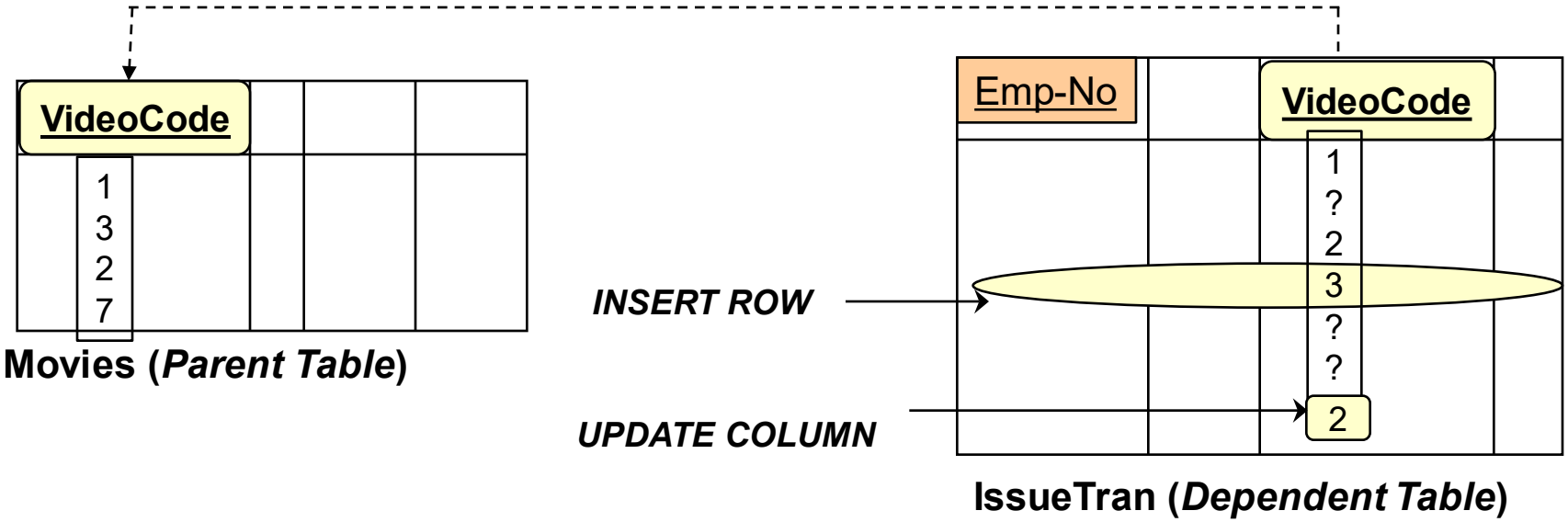
OrderDetails

OrderID [PK]	ProductID [PK]	Qty
A1091	Pen	100
A1091	Pencil	200
A1091	Eraser	250
A1092	Pen	10
A1092	Pencil	50
A1093	Pen	80
A1093	Eraser	90

Composite Primary Key : Key with more than 1 column

Foreign Key

- Foreign key constraints
  - Enforcing Referential Integritiy



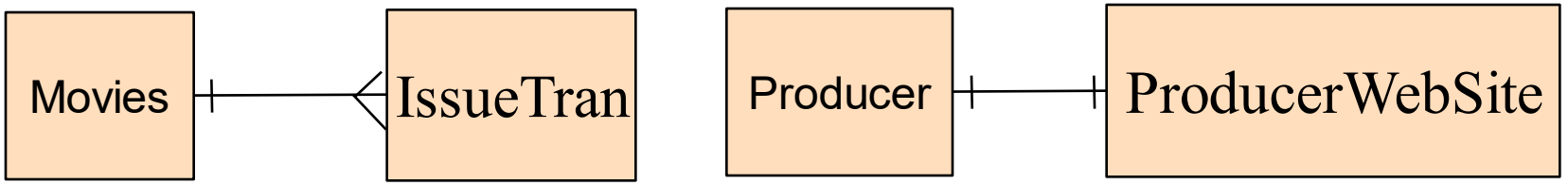
A row can be inserted or a column updated in the dependent table only if (1) there is a corresponding primary key value in the parent table, or (2) the foreign key value is set null.

Topics

- Overview of Relational Database table Concepts
- Primary Key
- **Entity Relations Diagram**
- Foreign Key
- Structured Query Language (SQL)
- Summary

Entity Relations

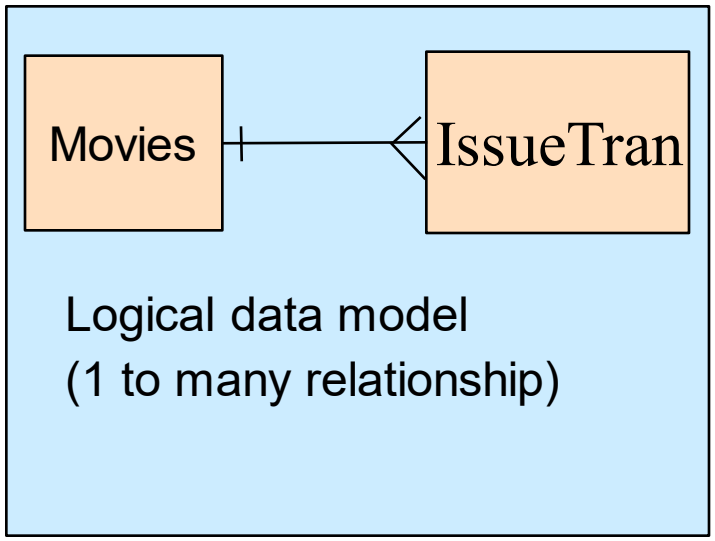
- Entities are another term for tables which hold data
- Entity Relations shows the relations of records in different tables
  - Typically
    - 1 to 1 relationship
    - 1 to many relationship

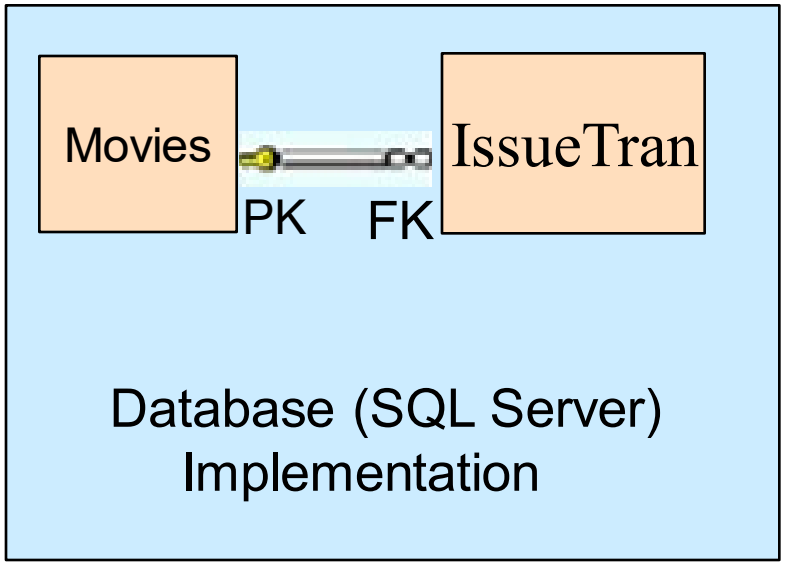


1 to many relationship

1 to 1 relationship

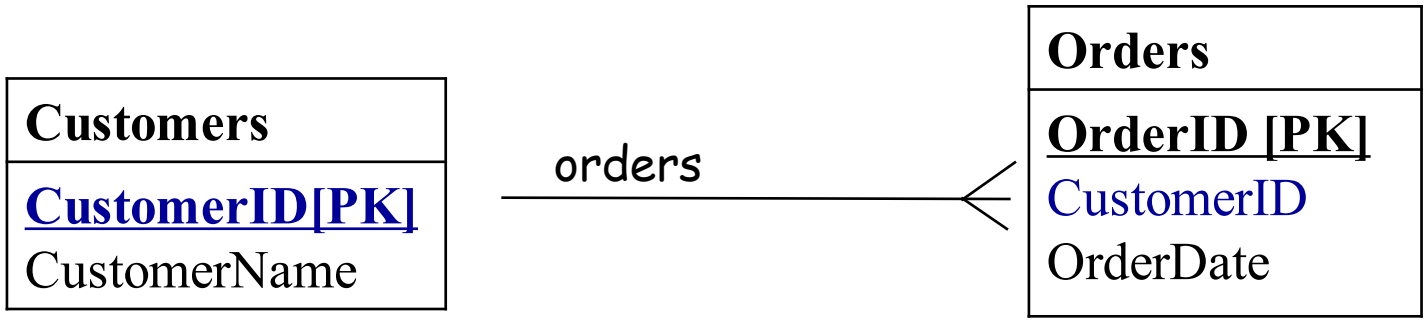
Database Design and Implementation





- Database does not enforce the creation of PK or FKs

Related Table - 1



Tables are related through common attribute(s)

CustomerID	Customer Name
S009	Lynn Wang
S010	Suzan Tan

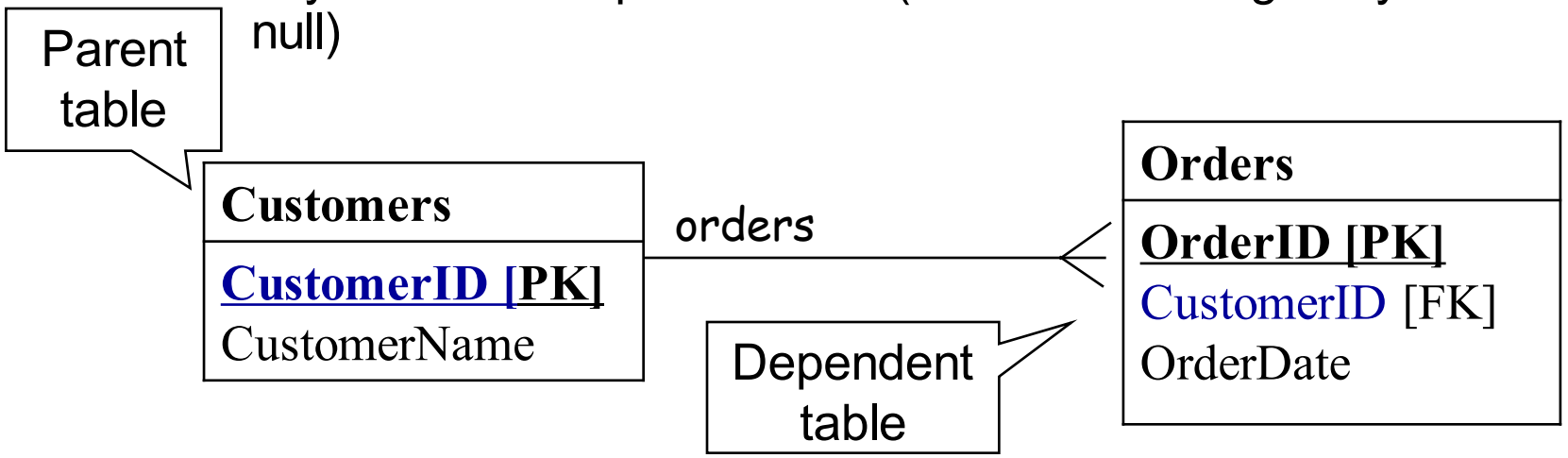
Order ID [PK]	CustomerID	Order Date
A1091	S009	21/7/2011
A1092	S010	12/1/2011
A1093	S010	21/8/2011

Topics

- Overview of Relational Database table Concepts
- Primary Key
- Entity Relations Diagram
- **Foreign Key**
- Structured Query Language (SQL)
- Summary

Foreign Key

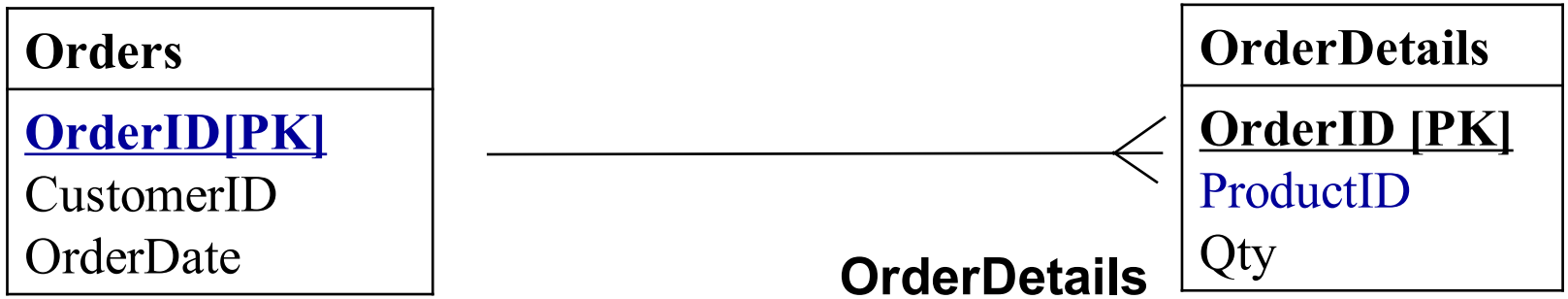
- Enforces referential integrity
  - foreign key attribute must correspond to an existing primary key value in the parent table (unless the foreign key value is null)



CustomerID	Customer Name
S009	Lynn Wang
S010	Suzan Tan

OrderID [PK]	CustomerID	OrderDate
A1091	S009	21/7/2011
A1092	S010	12/1/2011
A1093	S010	21/8/2011

Foreign Key (cont'd)



Order

OrderID [PK]	CustomerID	OrderDate
A1091	S009	21/7/2011
A1092	S010	12/1/2011
A1093	S010	21/8/2011

OrderDetails

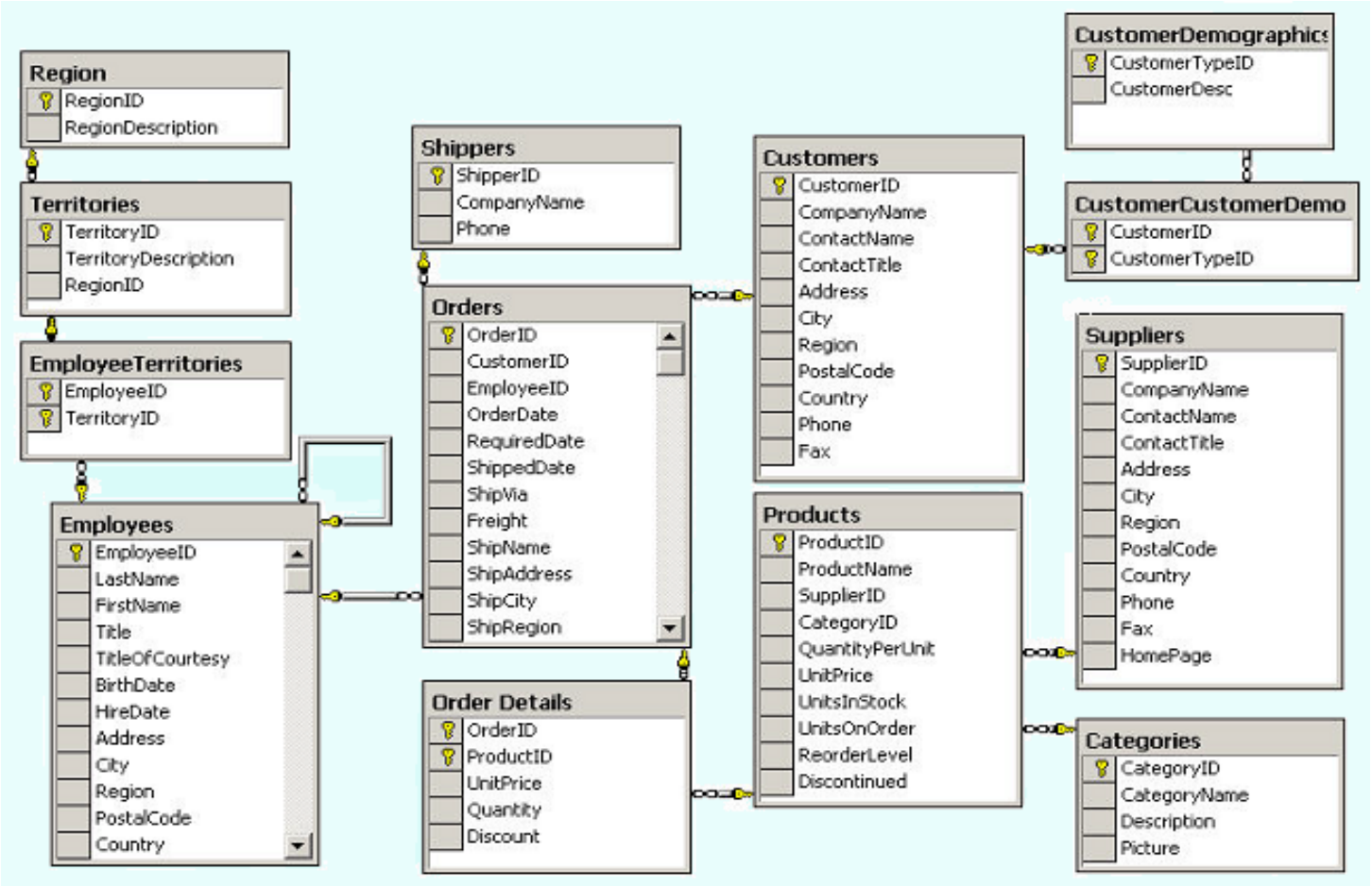
OrderID [PK]	ProductID [PK]	Qty
A1091	Pen	100
A1091	Pencil	200
A1091	Eraser	250
A1092	Pen	10
A1092	Pencil	50
A1093	Pen	80
A1093	Eraser	90

Database Diagram

- In RDBMS, databases are illustrated using an ERD (entity relationship diagram)
  - Example of a E-R Model is shown in the next slide.
  - This diagram illustrates Northwind Database provided along with MS SQL Server.
  - This system typically depicts a trading system consisting Sales and Purchases.
  - Each entity is represent by a table : eg. Customers table, Employees table, etc. Each row in the table is called the entity data.
  - The lines joining the tables are the relationships.
  - There are a number of relationships between entities in an ERD.
    - 1 to 1 relationship
    - 1 to many relationship
    - Also entity related to itself (Employees)

Database Diagram Diagram

Northwind Entity Relationship Diagram



Northwind ERD

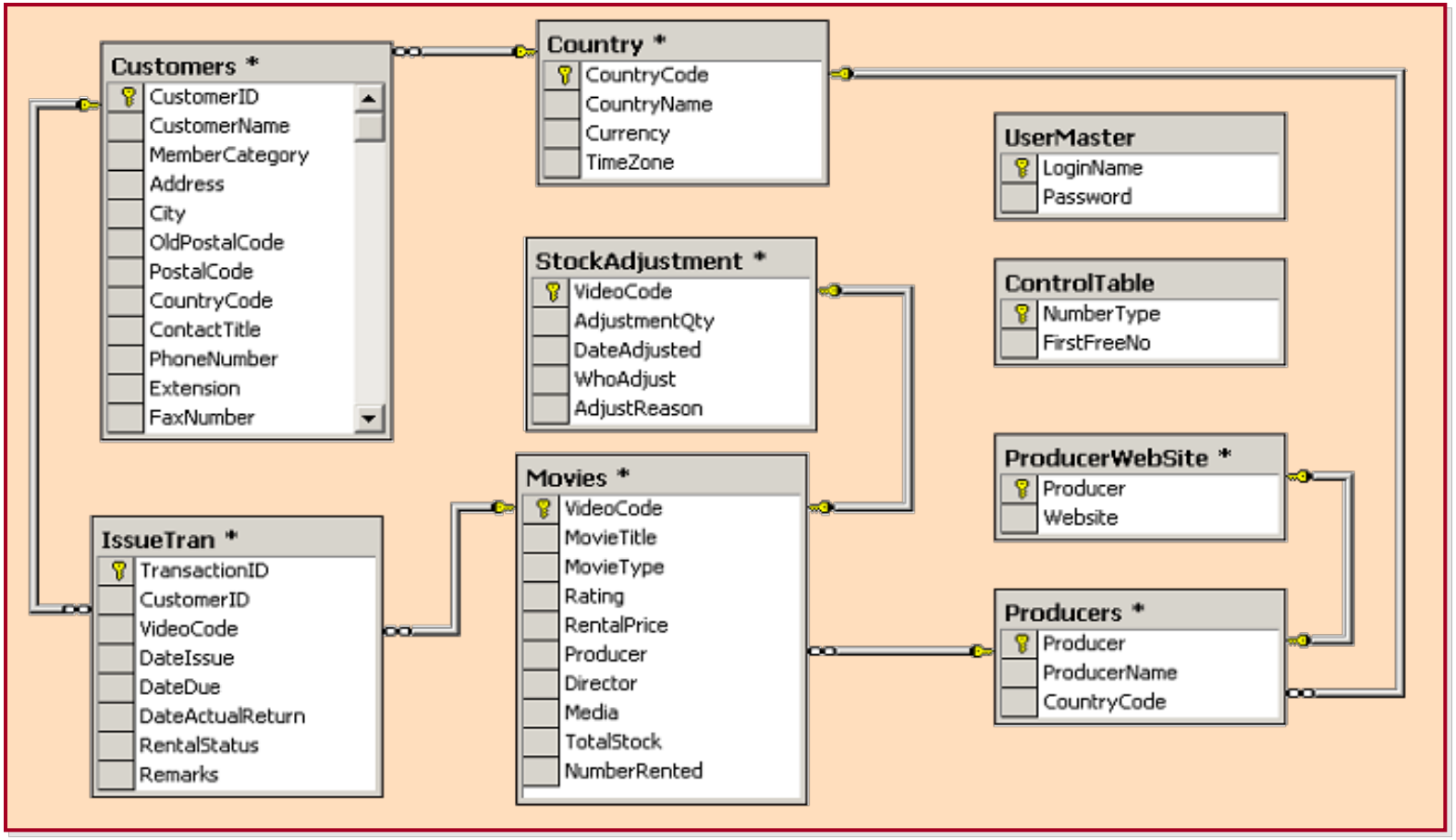
- Interpretation of the Northwind ERD is as follows :
  - Notice that there is an employee id field under the Orders and Employees table.
  - The employee id in Employees table is the primary key while the same id in the Orders table is the foreign key. In other words, for every row in the orders table, that row's employee id must be found in the Employee table.
  - The relationship between the Employee and Orders table is a 1 to many relationship. This means that for each employee id in the Employee table, there could be repetition of the same id in the Orders table.
  - Note that the (One to Many) relationship is determined by the a row of the Employee table with respect to many rows of the Orders table having the same EmployeeID

Northwind ERD (cont'd)

- Interpretation of the Northwind ERD (cont'd):
  - Employees Table has a self referencing relationship. The table has a "report to" column (apart from the employeeID) which requires the employee number to be insert
  - Each employee reports to another employee (I.e., the Boss). Hence each row in the employee table (ie each employee) has a column that requires another EmployeeID for "reporting to" relationship to be established.

employeeid	lastname	firstname	reportsto
1	Davolio	Nancy	2
2	Fuller	Andrew	NULL
3	Leverling	Janet	2
4	Peacock	Margaret	2
5	Buchanan	Steven	2
6	Suyama	Michael	5
7	King	Robert	5
8	Callahan	Laura	2
9	Dodsworth	Anne	5

Database Diagram Diagram



Topics

- Overview of Relational Database table Concepts
- Primary Key
- Entity Relations Diagram
- Foreign Key
- **Structured Query Language (SQL)**
- Summary

Structured Query Language

- A relational database language
  - It is not a programming language but a comprehensive database sub-language language for interacting with a database management system.

- Commonly used SQL commands:
  - Data Query and Manipulation :
    - Select — Retrieves Data
    - Insert — Add new rows of Data
    - Delete — Removes row of Data
    - Update — Modifies existing Data
  - Data Definition
    - Create Table — Adds a new Table
    - Drop Table — Removes existing tables
    - Alter Table — Modifies structure of Tables

Topics

- Overview of Relational Database table Concepts
- Primary Key
- Entity Relations Diagram
- Foreign Key
- Structured Query Language (SQL)
- **Summary**

Summary

- Database Concepts
  - Collection of tables, objects
  - Table / Row / Column
- Important Keys of a Table
  - Primary Key
  - Foreign Key
- Entity Relations Diagram
  - Database Diagram
- Structured Query Language
  - Data Query and Manipulation Language
  - Data Definition Language

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

TOOLS FAMILIARIZATION

Chia Yuen Kwan

isscyk@nus.edu.sg

Topics

- **SQL Server Configuration Manager**
- SQL Management Studio
- Creation of Database and Tables using Tools
- Import Database file

Accessing SQL Server

- SQL Server Configuration Manager
  - Starts / Stop the SQL Server
    - In order to query, manipulate or control the databases in SQL Server, SQL Service services needs to be running

- SQL Management Studio
  - To perform queries, manipulation or control the databases in SQL Server

SQL Server Configuration Manager

- By default installation settings, SQL Server is started when Windows is started

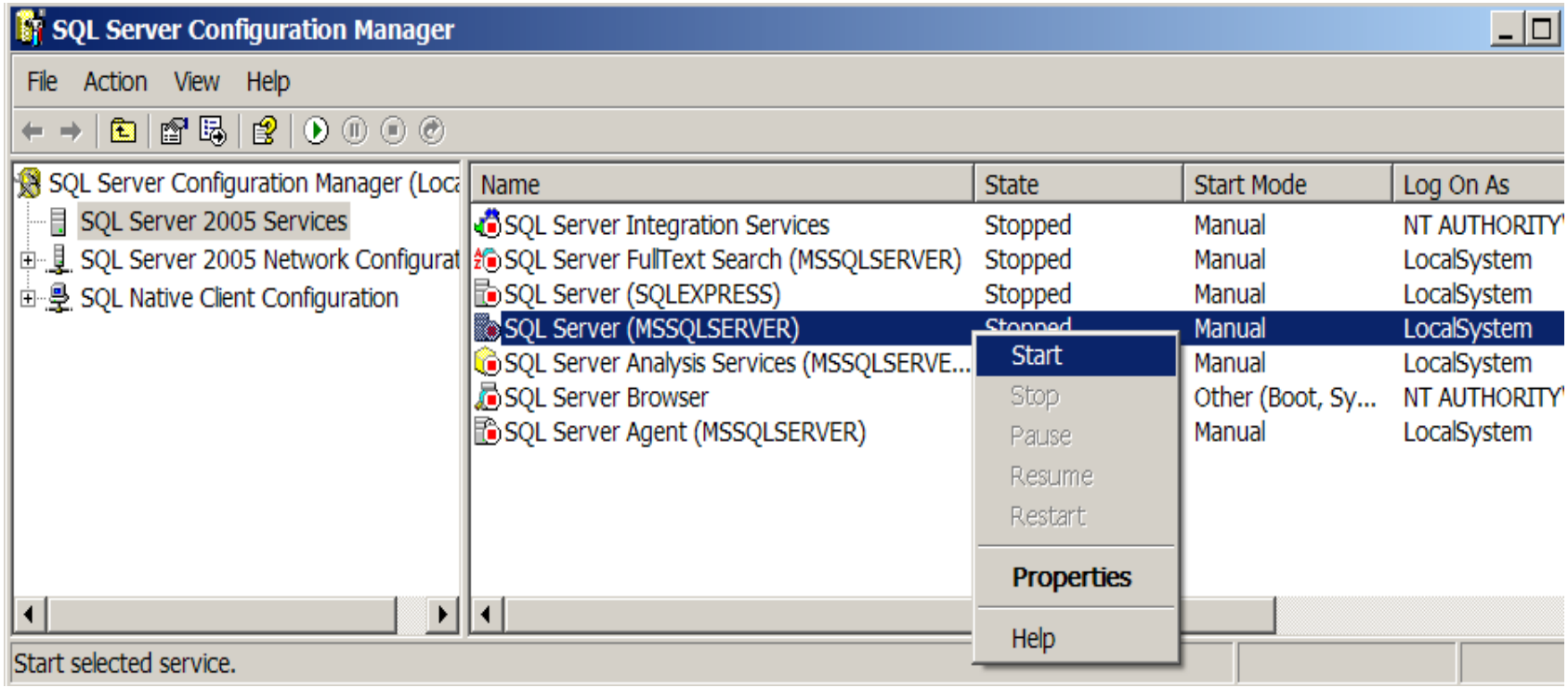
- To start or stop SQL Server, invoke **SQL Server Configuration Manager.**

  - to open Configuration Manager, access **SQLServerManager<version>.msc** file

SQL Server 2017	C:\Windows\SysWOW64\SQLServerManager14.msc
SQL Server 2016	C:\Windows\SysWOW64\SQLServerManager13.msc

SQL Server Configuration Manager

- In SQL Server Configuration Manager, start the SQL Server
  - Right click on the SQL Server (MSSQLServer), select Start

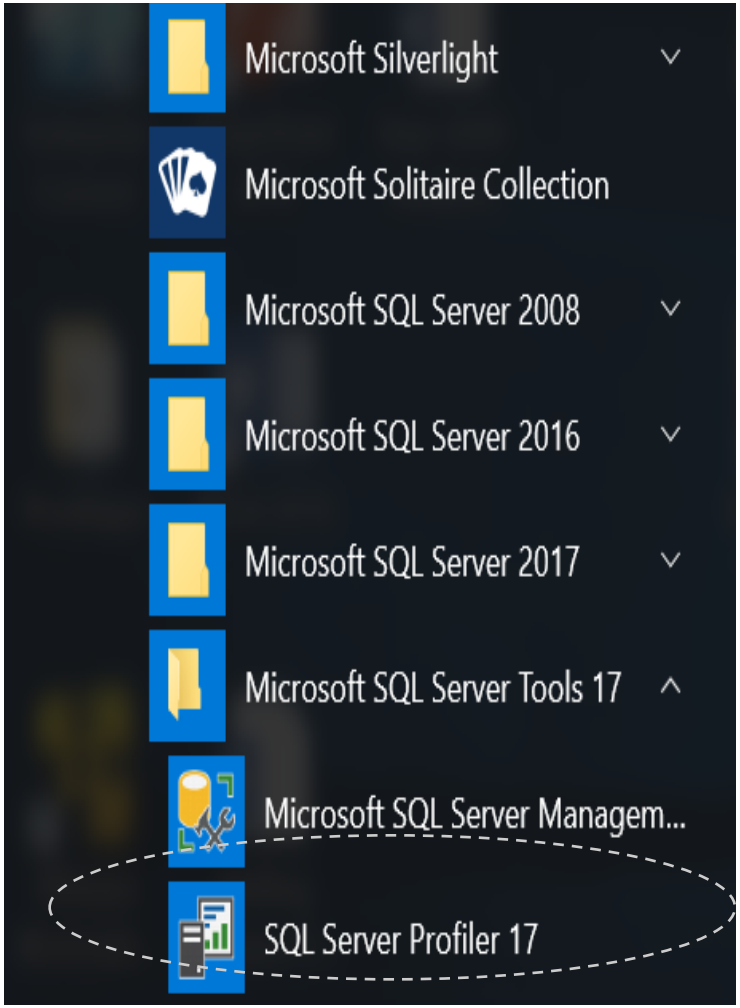


Topics

- SQL Server Configuration Manager
- **SQL Management Studio**
- Creation of Database and Tables using Tools
- Import Database file

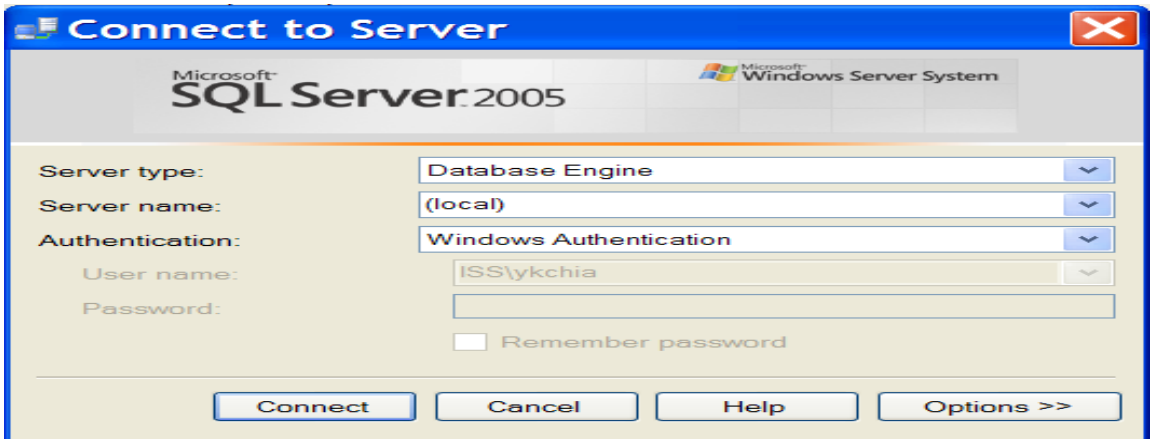
SQL Server Management Studio

- Invoke SQL Server Management Studio from Start Menu
- If SQL is not installed in your machine, download the installation file from Microsoft website
  - https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms



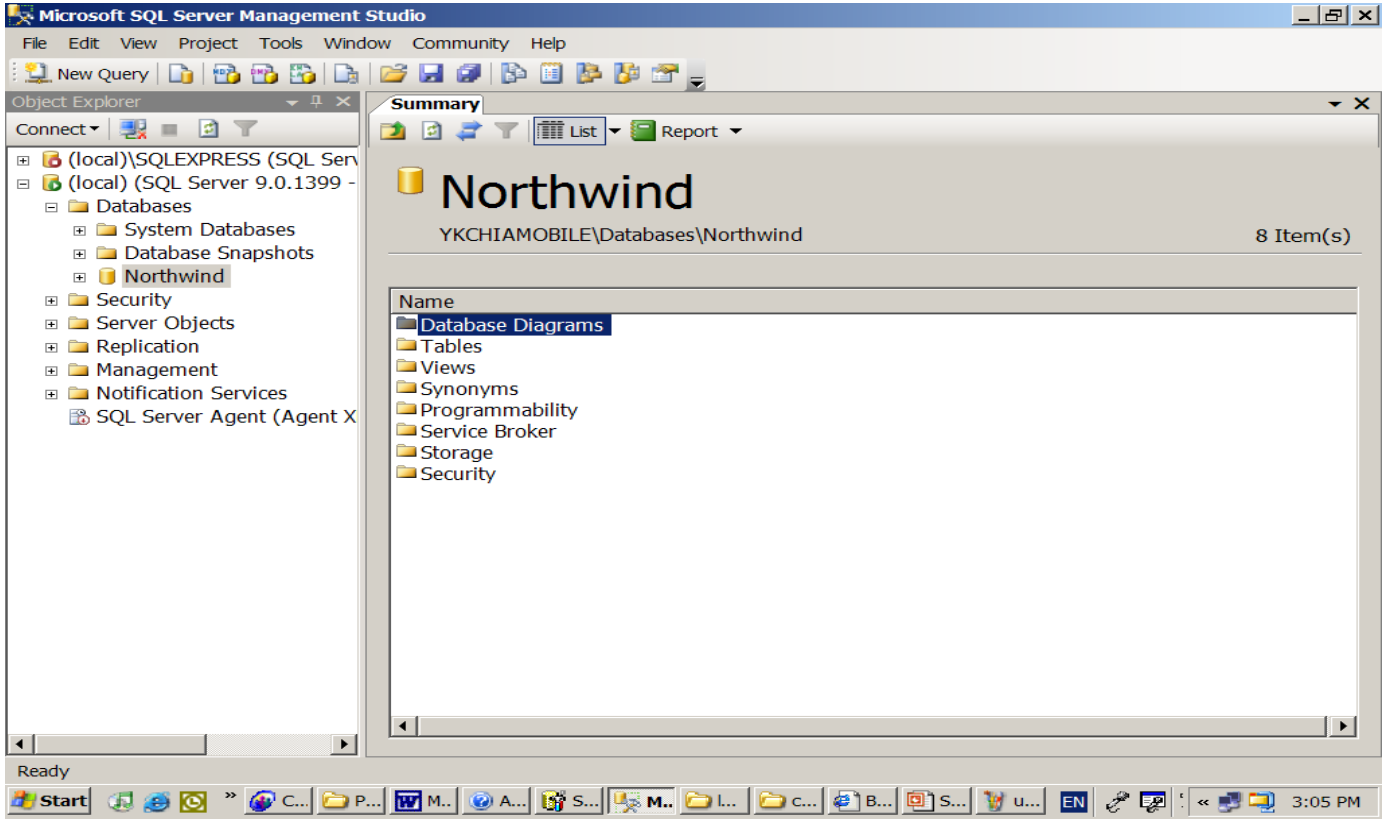
SQL Server Management Studio

- Make sure the Server type is “Database Engine”.
- Select the database server that you wish to connect to. *By default, the server name is your machine name.* You may also enter “**(local)”** for SQL Server installed on local machine.
  - For Windows Authentication, no user name or password is required. Whoever is authorised to access to the Windows environment, would be able to access SQL Server installed in the machine; as long as the SQL Server is configured to accept Windows Authentication.



SQL Server Management Studio

– Once you are in Management Studio, you can see the list of databases and objects in the object explorer (which might be empty if you have not created or import any database)



Topics

- SQL Server Configuration Manager
- SQL Management Studio
- **Creation of Database and Tables using Tools**
- Import Database file

Workshop – Create Tables With Tool

Step: Create a Database

- Click at Database to create a Database (“SAEx1”).
- Under the newly created database (“SAEx1”), go to the folder Database Diagram. Right click to create a Database Diagram.

Step: Create New Table

- In the empty Database Diagram, right click to select “New table” add tables as in the

Customers *

Column Name	Data Type	Allow Nulls
CustomerID	nchar(5)	☐
CompanyName	nvarchar(40)	☑
		☐

Orders *

Column Name	Data Type	Allow Nulls
OrderID	int	☐
CustomerID	nchar(5)	☐
		☐

Order Details *

Column Name	Data Type	Allow Nulls
OrderID	int	☐
ProductID	int	☐
Quantity	real	☑
		☐

Step: Set Primary Key

- Right click on the table to set the primary key as in the next diagram.

Customers *

	Column Name	Data Type	Allow Nulls
🔑	CustomerID	nchar(5)	☐
	CompanyName	nvarchar(40)	☑
			☐

Orders *

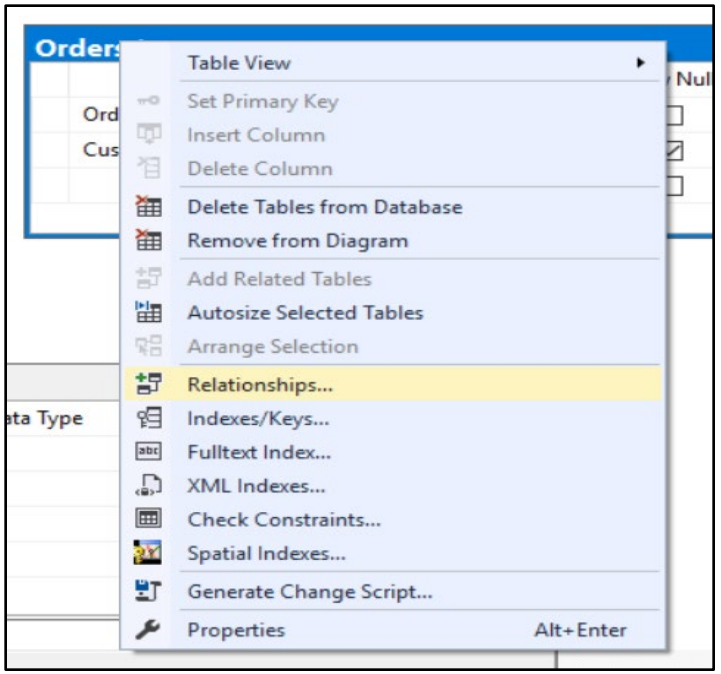
	Column Name	Data Type	Allow Nulls
🔑	OrderID	int	☐
	CustomerID	nchar(5)	☐
			☐

Order Details *

	Column Name	Data Type	Allow Nulls
🔑	OrderID	int	☐
🔑	ProductID	int	☐
	Quantity	real	☑
			☐

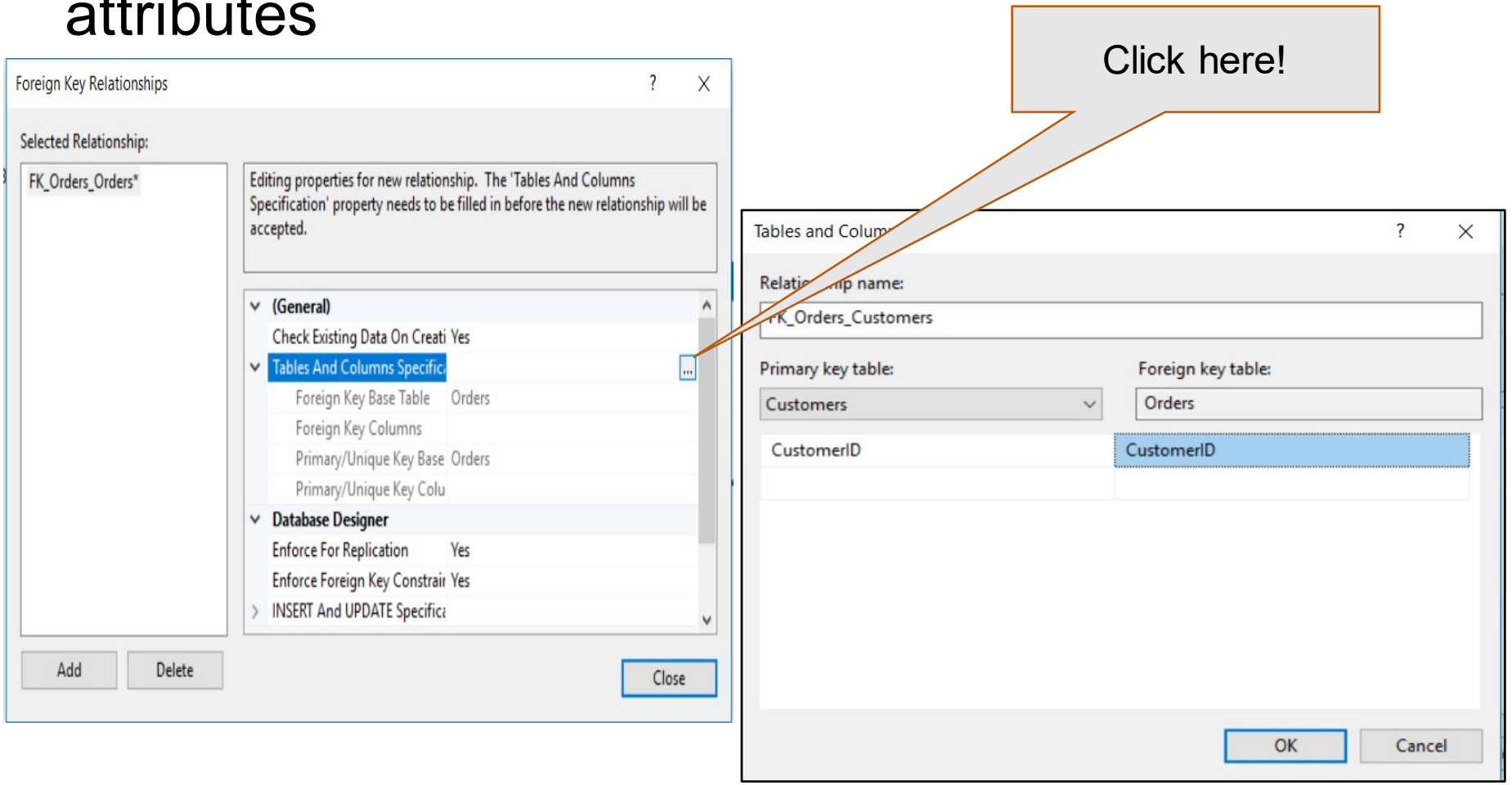
Step: Set Relations

- Set the relations for the tables
  - between Customers [Master] and Orders [Dependent]
  - between Orders [Master] and Order Details [Dependent]
- At Database Diagram, Right click on the (Dependent) table and select relationships



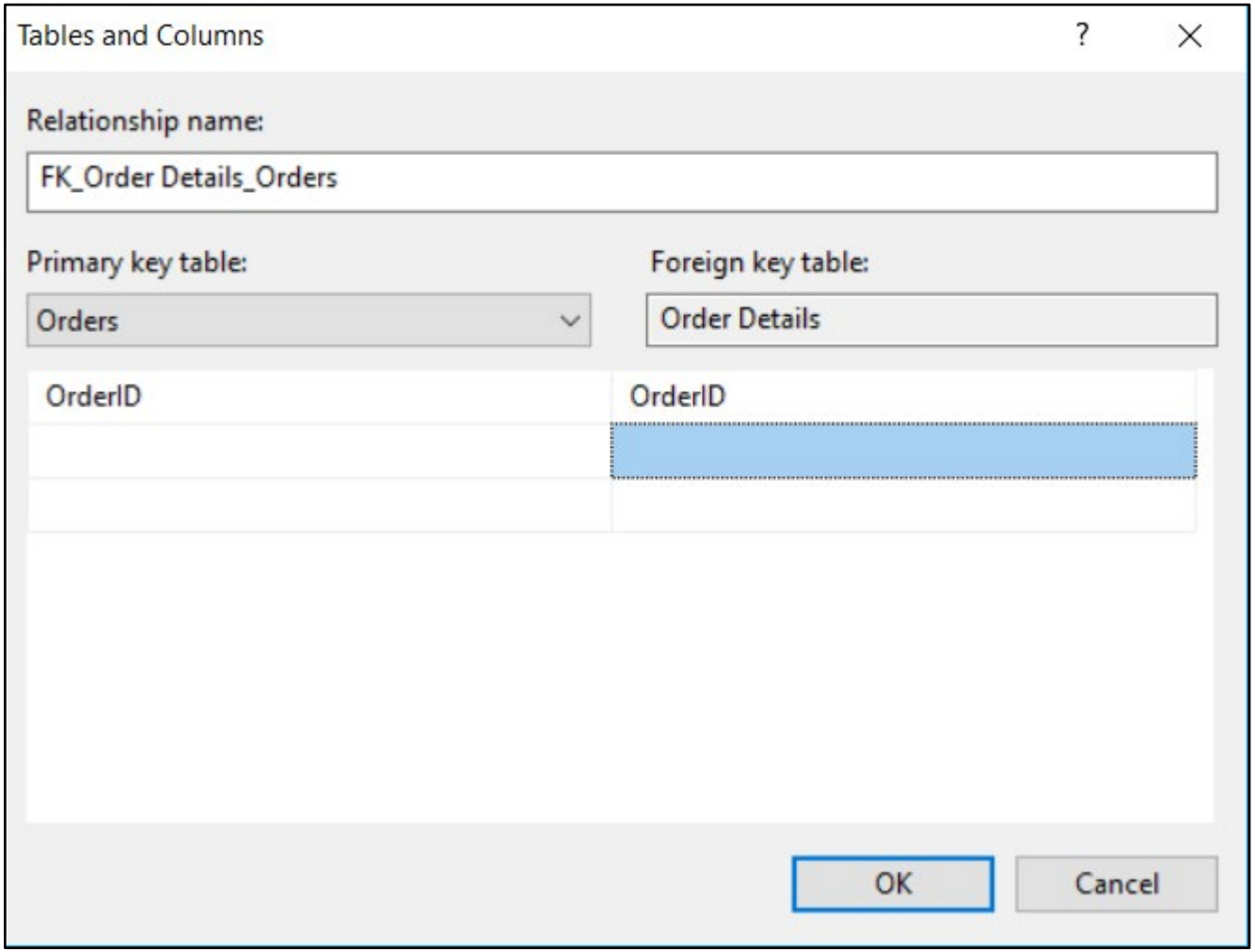
Step: Set Relations – Customers and Orders

- When defining the relations, specify the common attributes



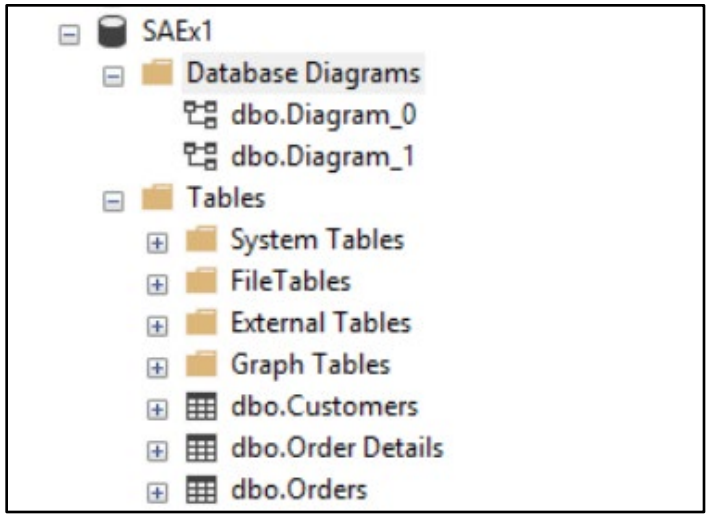
Step: Set Relations – Orders and Order Details

- Common attributes between Orders and Order Details



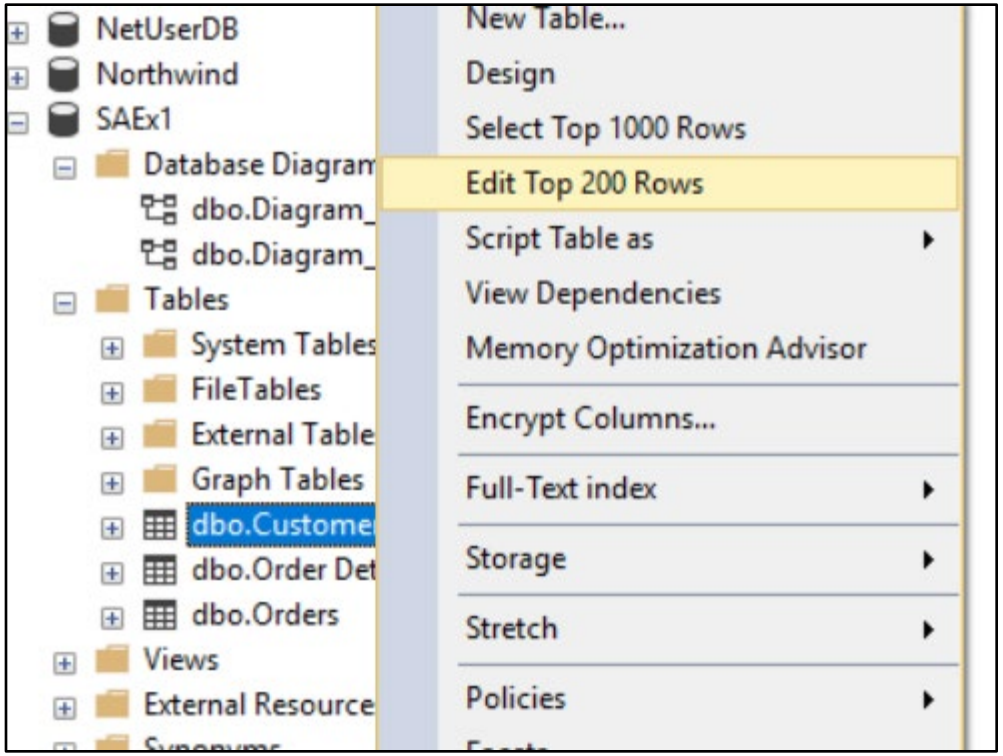
Step: Save the Creation

- Close the Database Diagram window to save the tables and diagram.
- Right click on “Database Diagrams” folder to refresh.
- Click on the “Tables folder” to check the table created.



Step: Create Rows

- Right click on the Customers table and select “Edit Top 200 rows”



- Create a few rows of data. Once you are done, close the window to save the records.
- Repeat the same for Orders and Order Details tables

Question

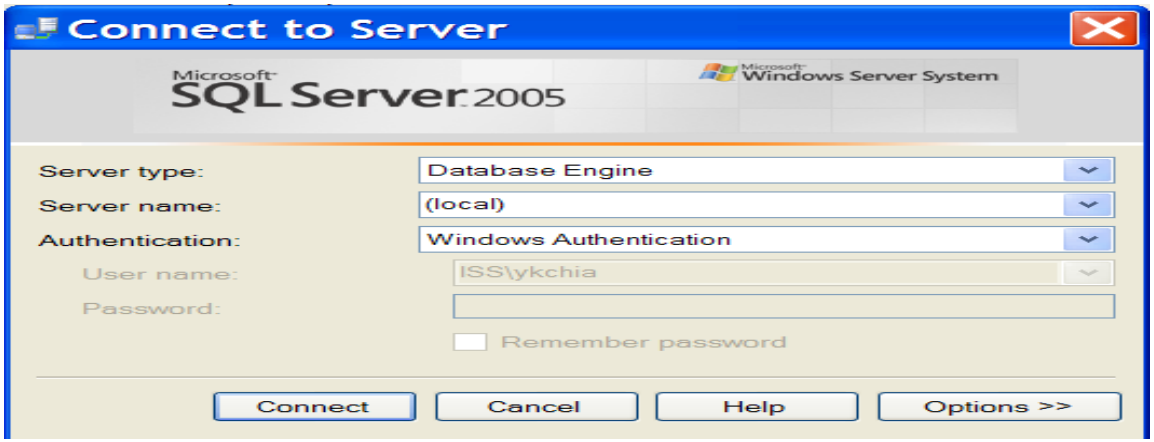
- During the record creation into the table, observe the followings.
  - Could you create rows with duplicate primary key column value?
  - Could you create row with no value in the column if the column is defined as follows?
    - “allow null”
    - Not “allow null”
  - Could you create rows into dependant table if there is no matching row in master table?

Topics

- SQL Server Configuration Manager
- SQL Management Studio
- Creation of Database and Tables using Tools
- **Import Database file**

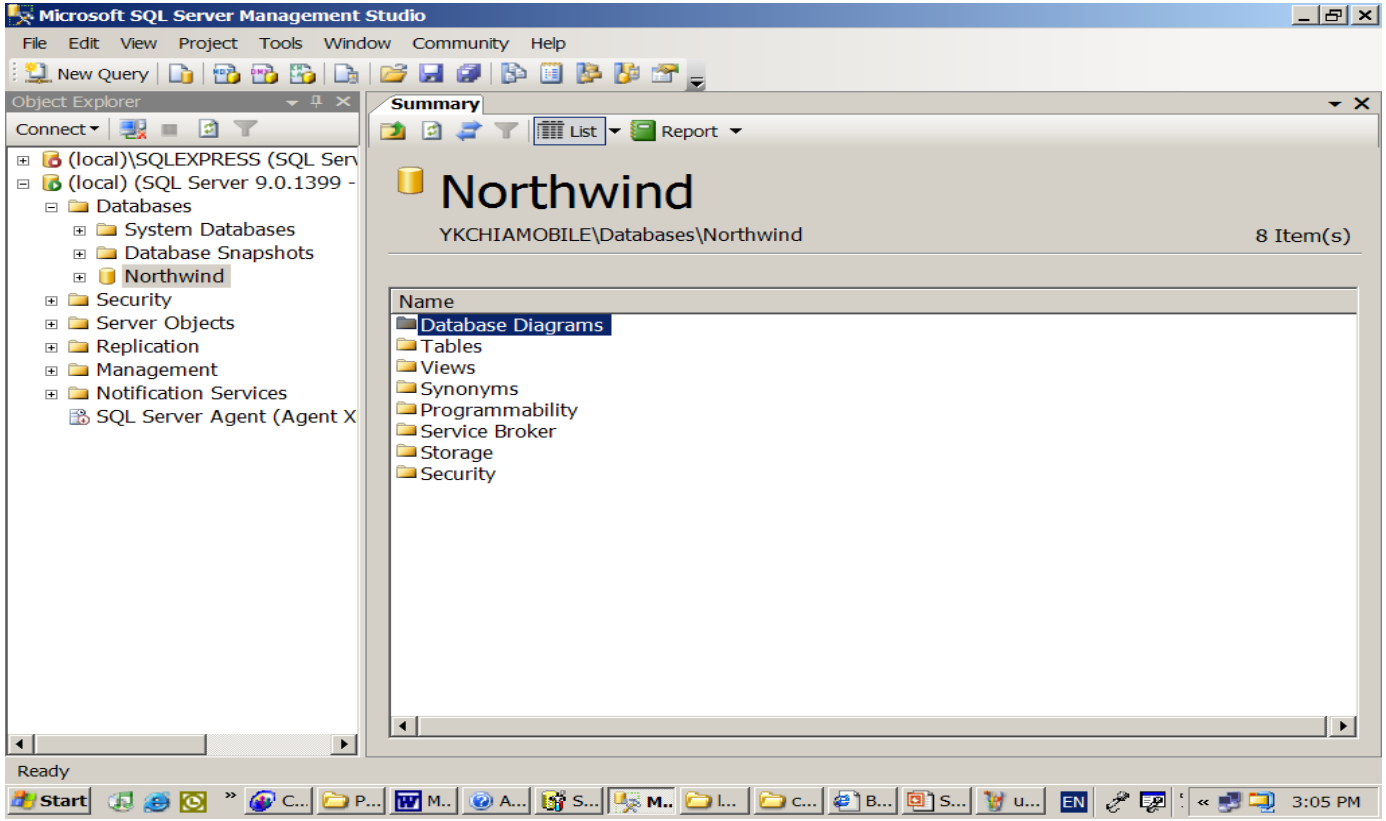
SQL Server Management Studio

- Make sure the Server type is “Database Engine”.
- Select the database server that you wish to connect to. By default, the server name is your machine name. You may also enter “**(local)”** for SQL Server installed on local machine.
  - For Windows Authentication, no user name or password is required. Whoever is authorised to access to the Windows environment, would be able to access SQL Server installed in the machine; as long as the SQL Server is configured to accept Windows Authentication.



SQL Server Management Studio

– Once you are in Management Studio, you can see the list of databases and objects in the object explorer (which might be empty if you have not created or import any database)



Exercise – Import Database

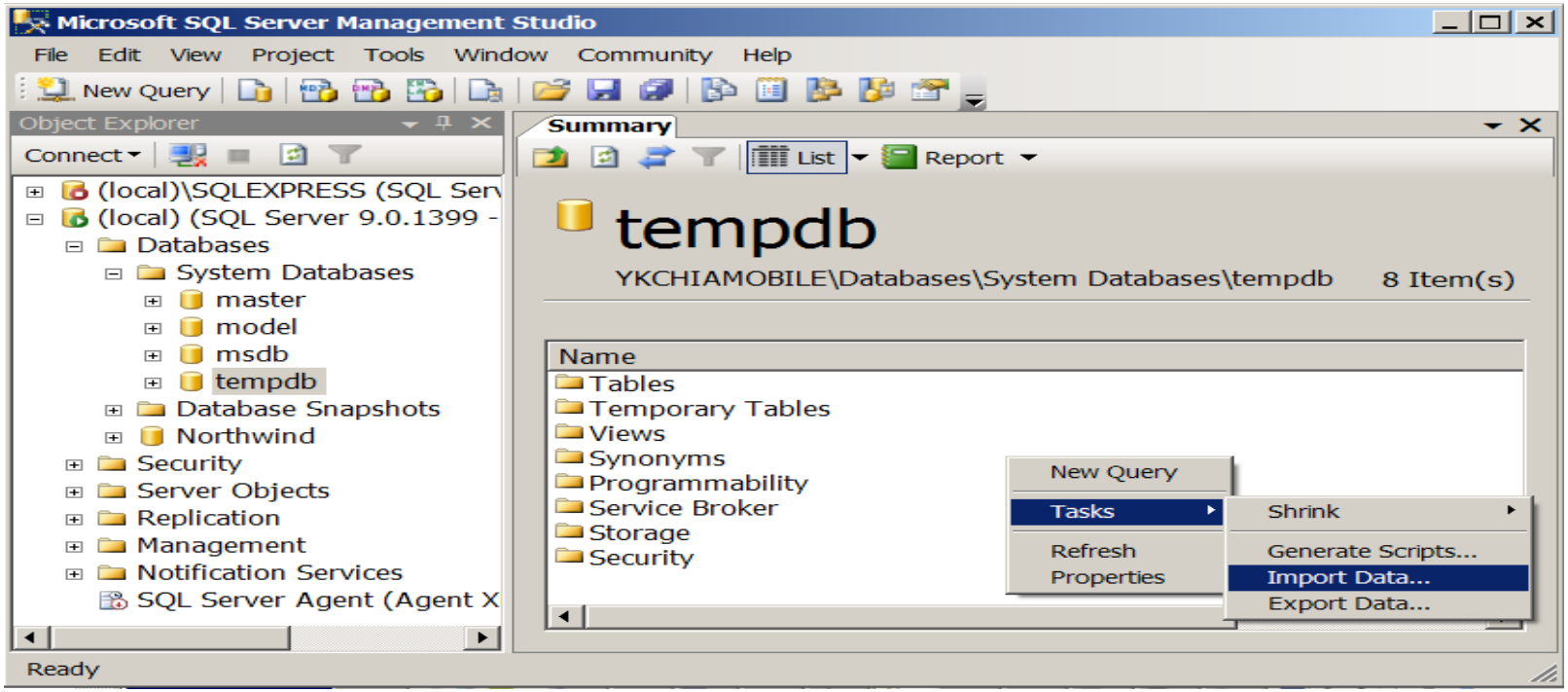
- Exercise
  - (Follow Slides 4 - 8) Start MS SQL Server (or verify SQL Server is started)
  - (Follow Slides 9 - 25) Using the MS Access Database provided (in ivle), migrate the **Dafesty** and **Northwind** Database to your local SQL Server. You need to perform the steps twice to import the two database.
    - DafestyVideo.mdb
    - Northwind.mdb

Importing Data

- To start Data transformation services:
  - Start SQLServer Management Studio

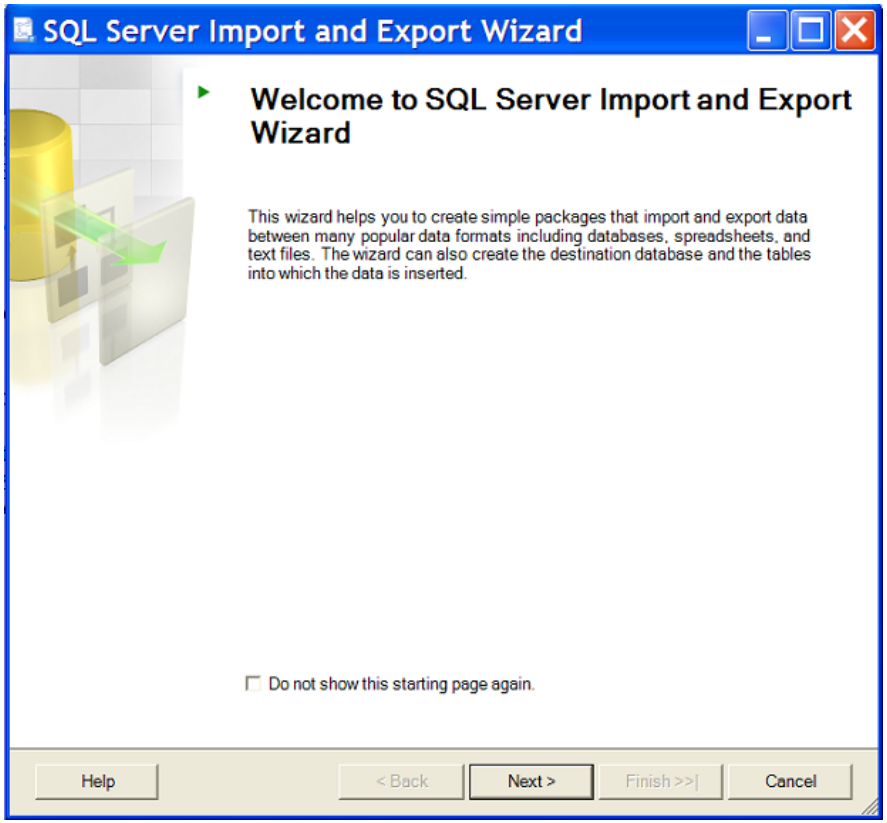
Importing Data

- Click on any database (e.g. tempdb under System Database), right click on the window in the left, select tasks -> Import Data



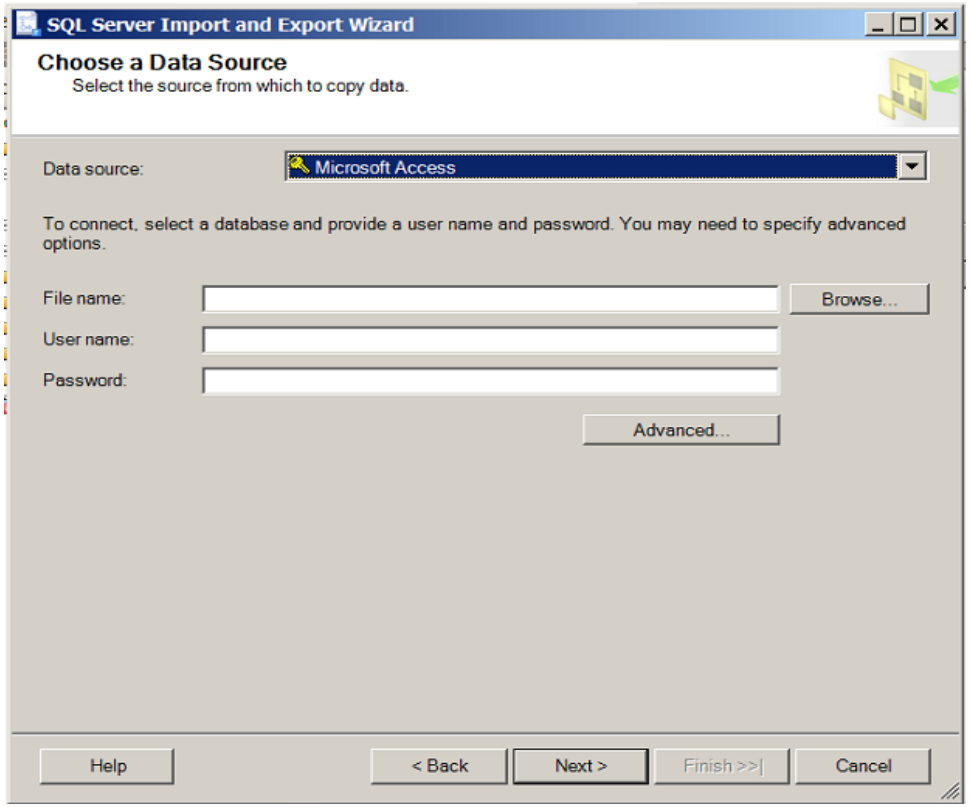
Importing Data

- SQL Server Import and Export Wizard is the first screen you see.



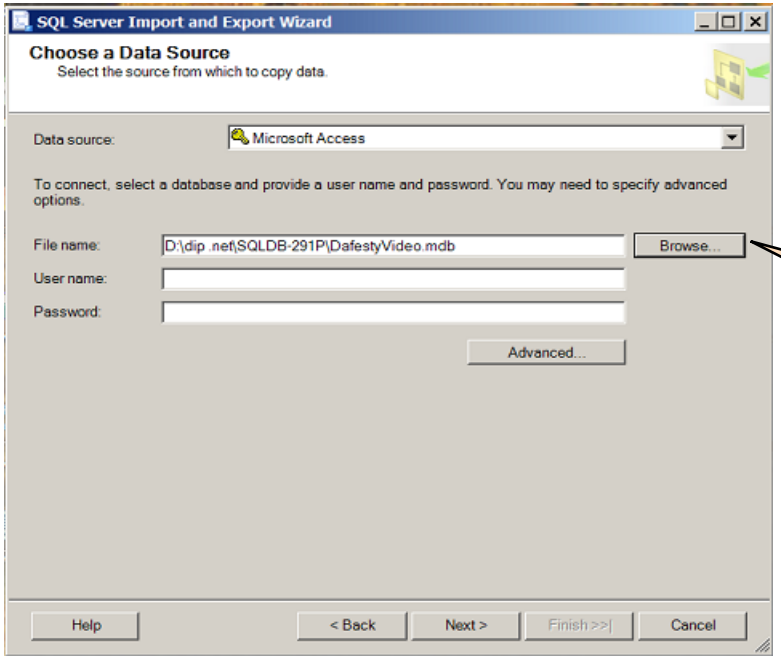
Importing Data

- To import Microsoft Access Database to the SQL Database
  - Select Microsoft Access under Data Source dropdown combo.



Importing Data

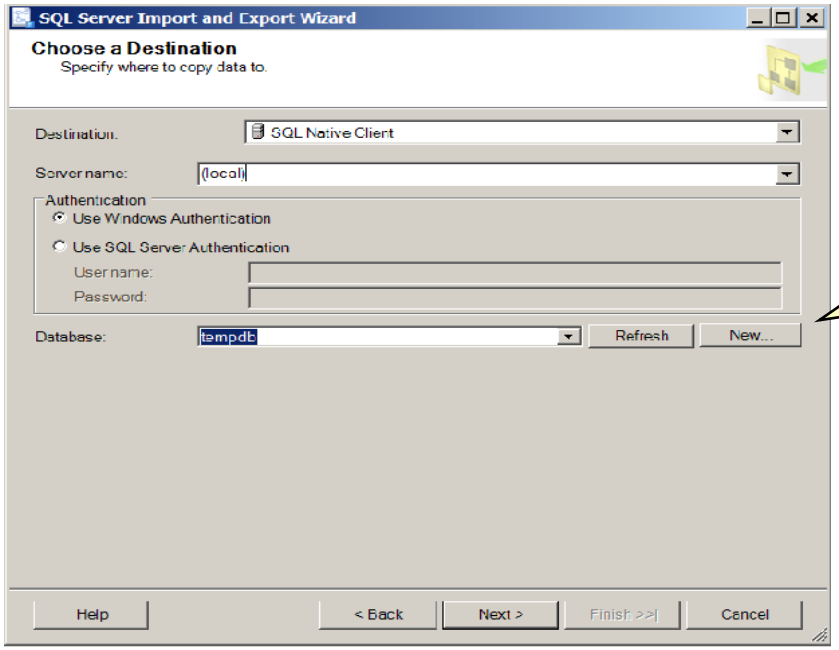
- Once the type of Database to be imported is determined, select the file name of the database to be imported :
  - Select the File name of the Database (Eg. DafestyVideo.mdb) under the File name textbox. If that access database requires a user name and password, insert them in the various textbox accordingly and click next button.



source files provided

Importing Data

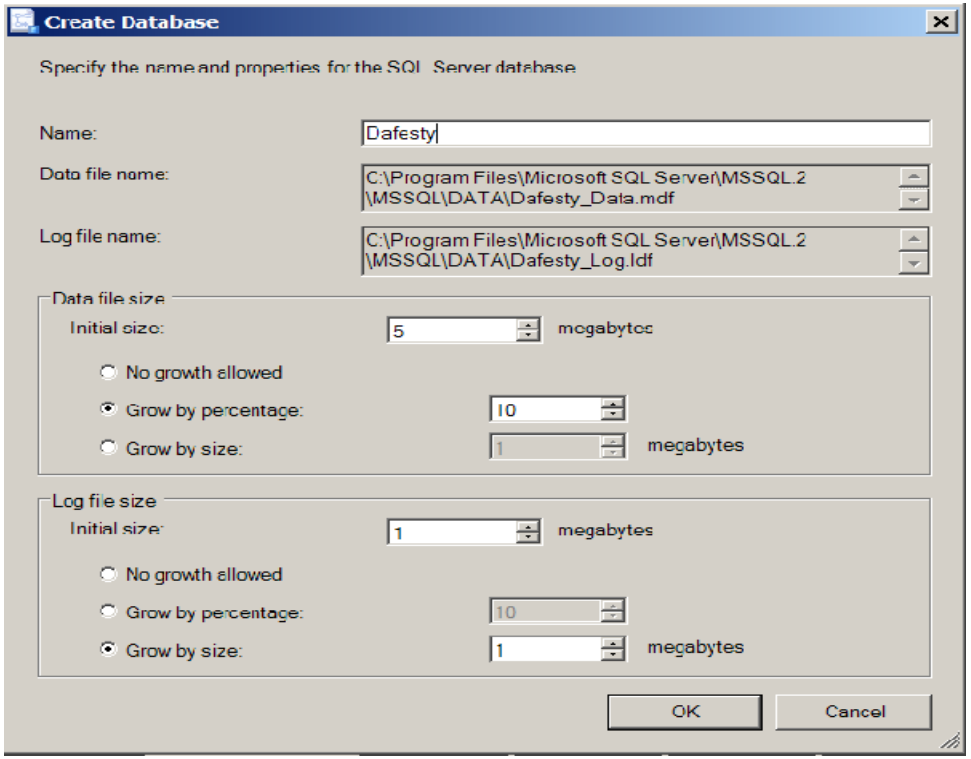
- Next, Select the Destination of the database to import to.
  - Under the Server name, enter (local) to install in local machine; or select the server under the server dropdown combo
  - Use SQL Server Authentication (Choose Windows Authentication)
  - For importing data into a new database, click the button “New”



Click **“New”** for importing data into a new database. Otherwise, data will be appended to existing database

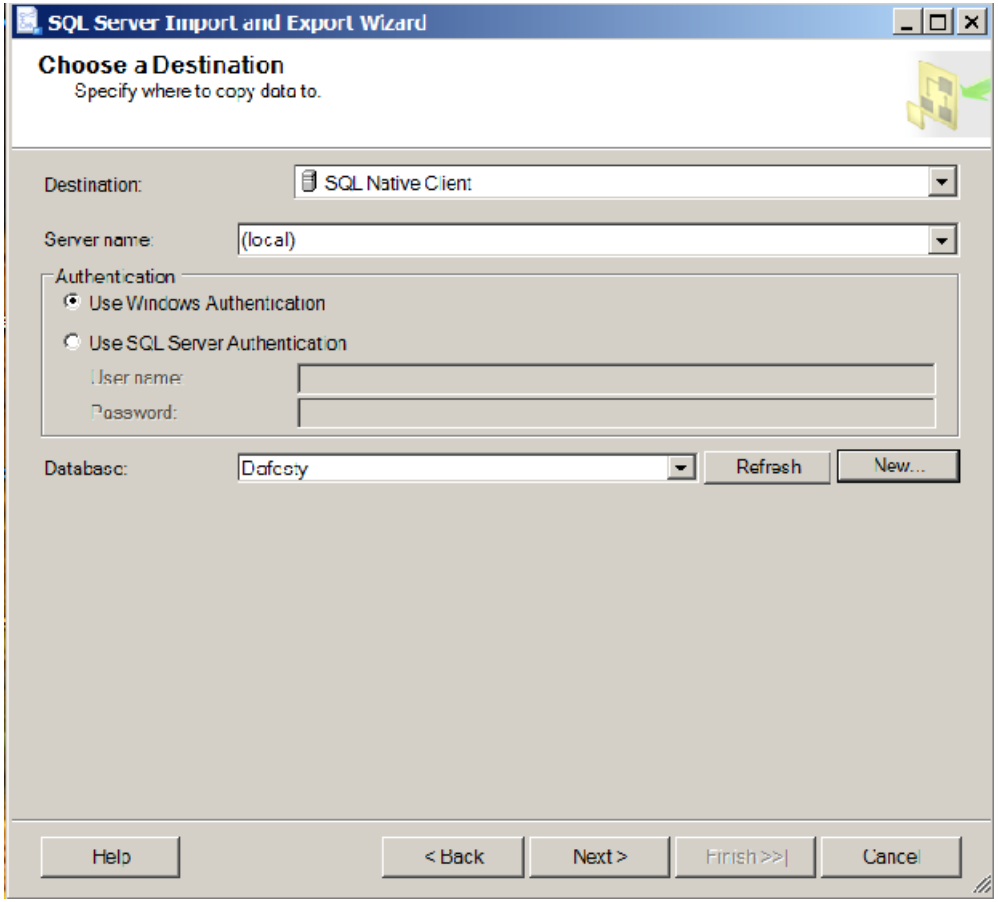
Importing Data

- Enter a database name (e.g. Dafesty or Dafesty*MyName*).
  - Accept other default values and click Ok.



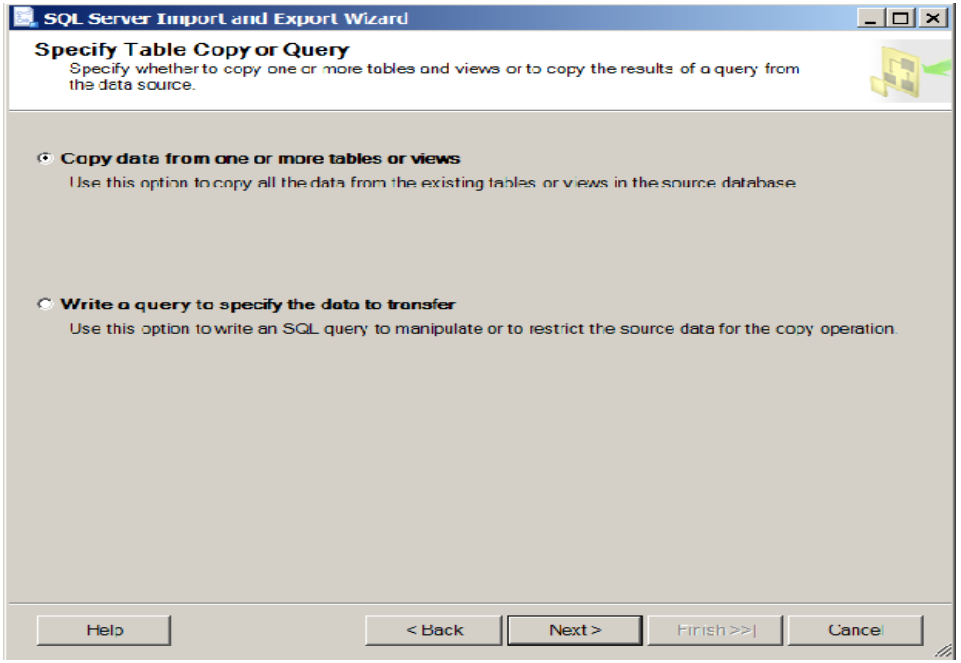
Importing Data

- It will bring you back to the screen to choose a destination for your datasource
  - Click Next



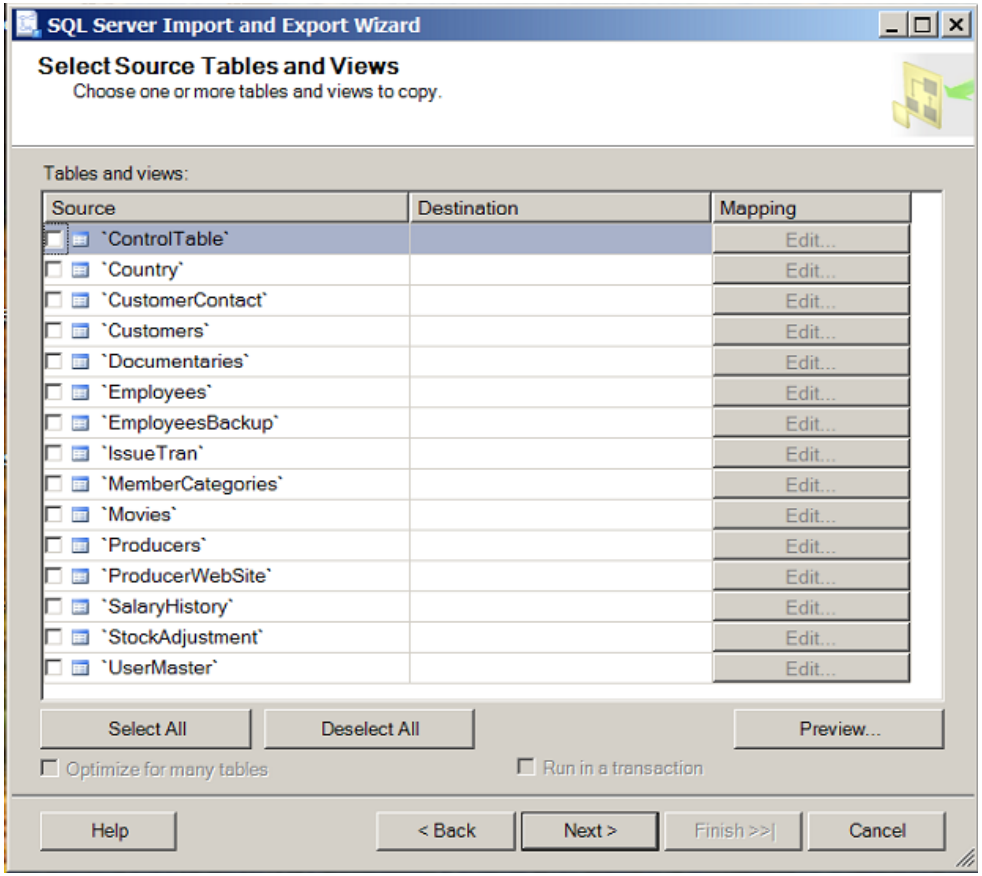
Importing Data

- To determine which portion of the database (ie some or all of the tables and their rows) are selected
  - Select copy table(s) and view(s) from the source database option and click next button. Using query to specify the data is more tedious but has more freedom in deciding which rows of a table to be ported to the new database



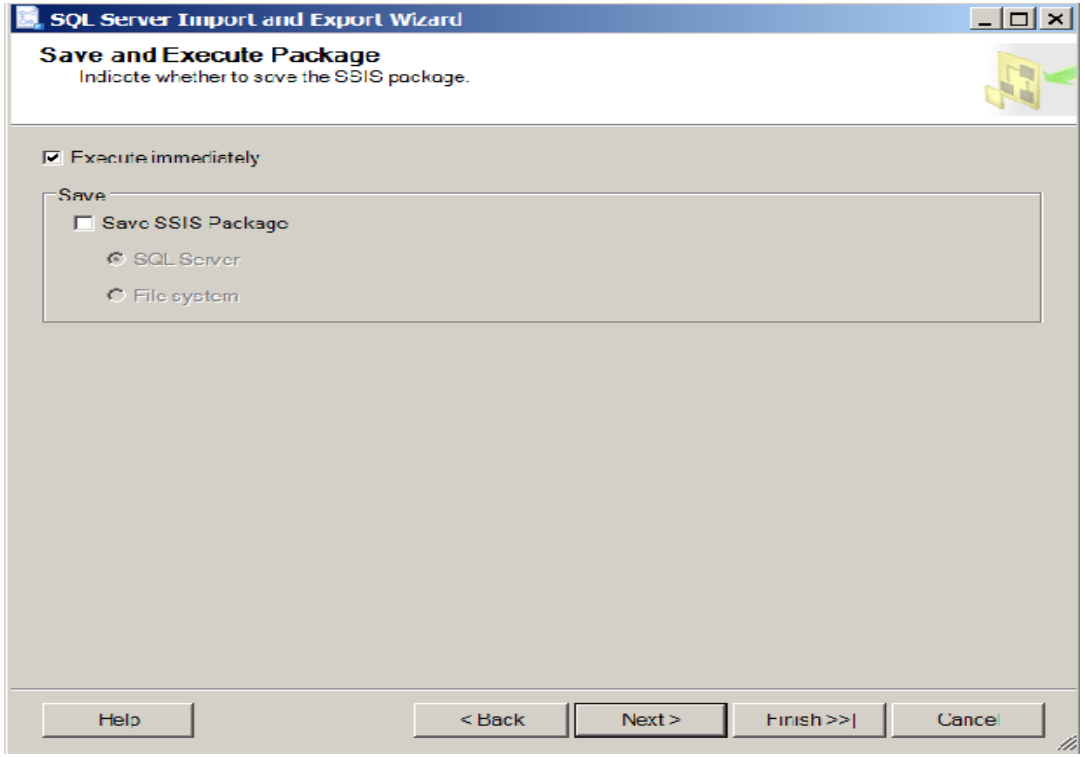
Importing Data

- Select the tables to be imported to SQL Server
  - You can select all the tables in the database with the Select All button



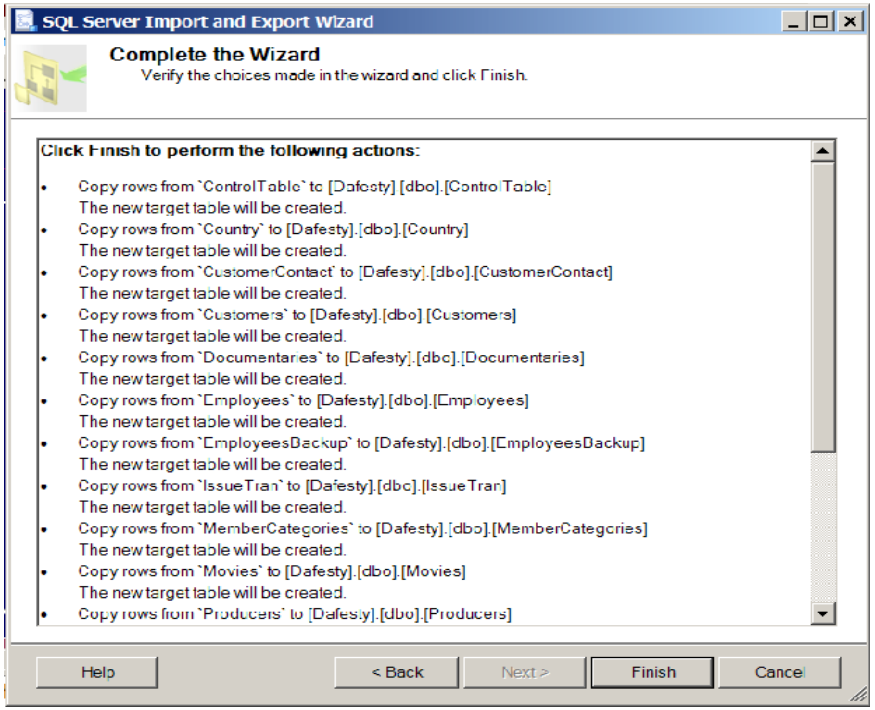
Importing Data

- After determining the database and data to be imported, you can select whether to execute the importing of the database at present or save it later.
  - Select Run immediately and click next button.



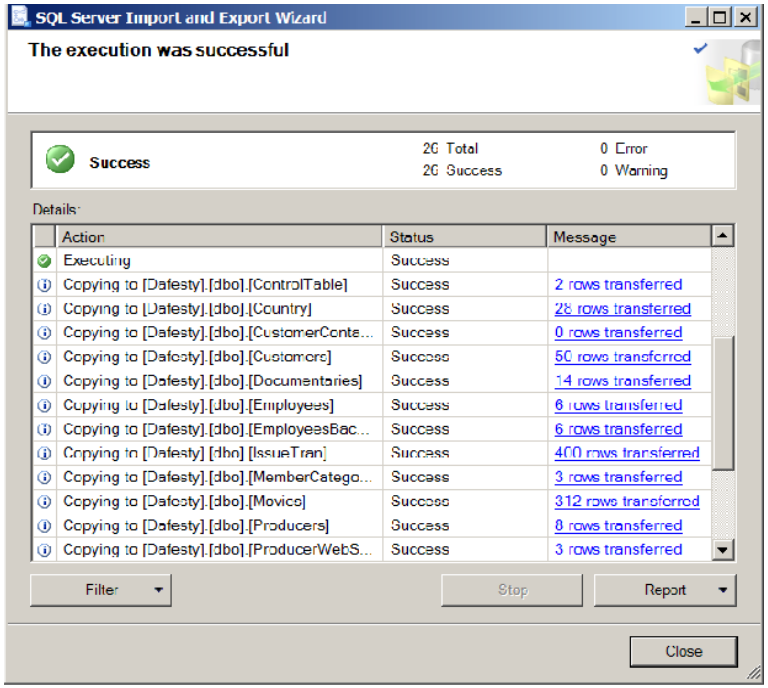
Importing Data

- After completing all the required parameters for importing the database, a summary of the parameters selected is shown as follows. To confirm the insertion, select Finish button.



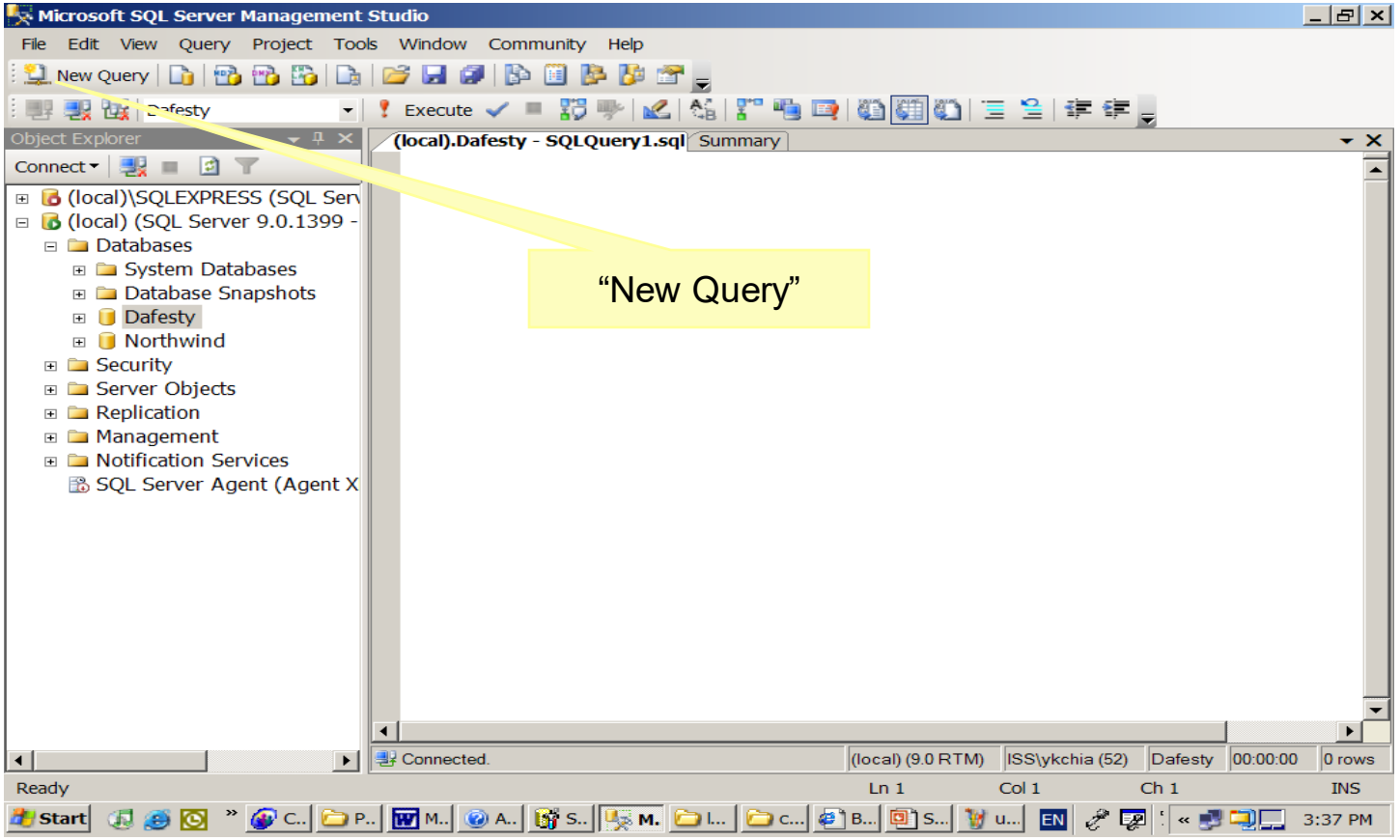
Importing Data

- The selected database will be replicated to SQL server as shown in the following diagram. Once the importation is finished, click Close button to end the process.

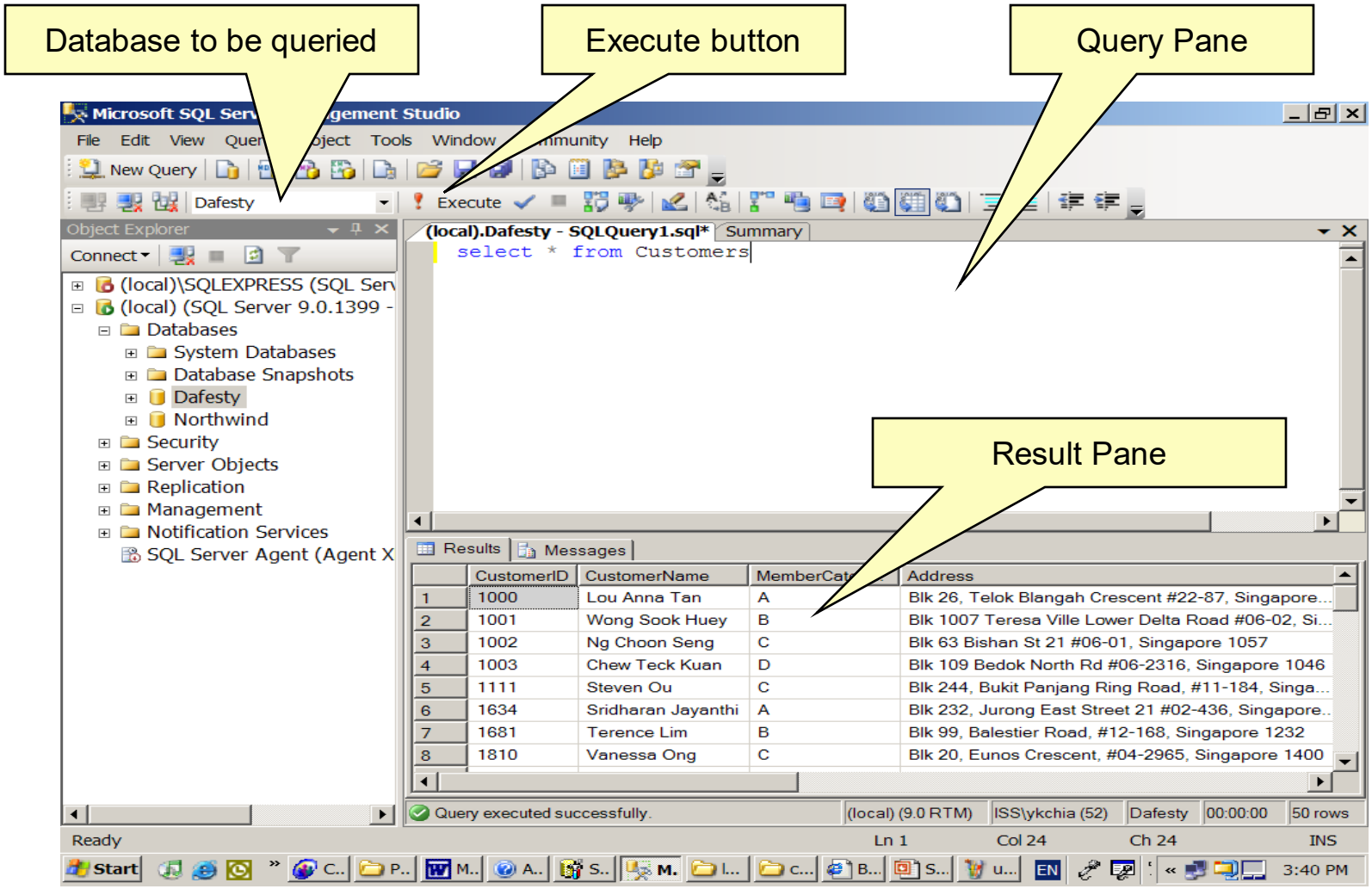


Querying Database

- To query a database, click “New query” in Management Studio. A query pane will be displayed, allowing SQL Statements to be input.



Querying Database



Querying Database

- Some Toolbar options explained:
  - Save File: Saves your SQL Statements in a .sql file.
  - Open File: Allows your .sql file to be loaded into the Query Pane.
  - Execute Query: Executes the SQL statement highlighted. If no SQL statement is highlighted, all the SQL statements will be parsed and executed.
  - Cancel Query Execution: Stops the execution if the SQL statements execution is taking too long.
  - Database Combobox: Allows the selection of the Database object (eg. Northwind database)

What you have tried…

- SQL Server Configuration Manager
- SQL Management Studio
- Creation of Database and Tables using Tools
- Import Database file

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

SQL SELECT STATEMENTS

Chia Yuen Kwan

isscyk@nus.edu.sg

Topics

- **Introduction**
- Basic Queries
- Basic Queries (Illustrations with Dafesty Video Rental Shop)
- Advanced Queries
- Advanced Queries (Illustrations with Dafesty Video Rental Shop)

Objectives

- Upon completion of this lesson, students should be able to use SQL for querying database Tables:
  - Selectively via columns and rows
  - Using Alias on columns and Tables
  - Using Joins
    - INNER JOINS, LEFT OUTER JOIN, RIGHT OUTER JOINS
    - FULL JOINS and CROSS JOINS
  - Involving Functions such as
    - Aggregate Functions
    - Mathematical, String and system Functions
  - Using Sub Queries
  - having conditions such as AND, OR, IN and BETWEEN conditions

SELECT Statement

- Use of the SELECT statement:
  - Retrieves data from one or more rows. Every SELECT statement produces a table of query results containing one or more columns and zero or more rows.
  - Note that the commands are NOT CASE SENSITIVE

SELECT Command

SELECT {[ALL, DISTINCT]} [(*select-column,…*) |*]
FROM (*table,….*)
{WHERE (*search condition*) {[AND|OR] (*search condition*)…}
{GROUP BY (*group-column,….*)}
{HAVING (*search condition*)}
{ORDER BY (*sort-column,….*)}

Topics

- Introduction
- **Basic Queries**
- Basic Queries (Illustrations with Dafesty Video Rental Shop)
- Advanced Queries
- Advanced Queries (Illustrations with Dafesty Video Rental Shop)

Select All

- SELECT * FROM Customers

The database table “Customers”

Id	Name	Age
DN001	John Chia	21
DN002	Tom	18
DN003	Jane	24

Select all rows and columns in a table

Result

Id	Name	Age
DN001	John Chia	21
DN002	Tom	18
DN003	Jane	24

Select Columns

- SELECT Id, Name FROM Customers

The database table “Customers”

Id	Name	Age
DN001	John Chia	21
DN002	Tom	18
DN003	Jane	24

Select particular columns of all rows in a table

Result

Id	Name
DN001	John Chia
DN002	Tom
DN003	Jane

Select Distinct

- SELECT DISTINCT Race FROM Customers

The database table “Customers”

Id	Name	Race
DN001	John Chia	CN
DN002	Tom	CN
DN003	Jane	IND
DN004	Shawn	ML

Select distinct values in a table

Result

Race
CN
IND
ML

Select - Where

- SELECT * FROM Customers WHERE Race ='CN'

The database table "Customers"

Id	Name	Race
DN001	John Chia	CN
DN002	Tom	CN
DN003	Jane	IND
DN004	Shawn	ML

To conditionally select data in a table

Result

Id	Name	Race
DN001	John Chia	CN
DN002	Tom	CN

Select - Not

- SELECT * FROM Customers WHERE NOT (Race ='CN')

The database table "Customers"

Id	Name	Race
DN001	John Chia	CN
DN002	Tom	CN
DN003	Jane	IND
DN004	Shawn	ML

NOT condition can be specified

Result

Id	Name	Race
DN003	Jane	IND
DN004	Shawn	ML

Select - Alias

- SELECT Id AS CustomerID, Name FROM Customers

The database table “Customers”

Id	Name	Age
DN001	John Chia	21
DN002	Tom	18
DN003	Jane	24

Alias

Alias redefines the result set column name

Result

CustomerID	Name
DN001	John Chia
DN002	Tom
DN003	Jane

Column name has been changed

Select - And

- SELECT * FROM Customers WHERE Race ='CN' AND Age >=21

The database table "Customers"

Id	Name	Age	Race
DN001	John Chia	21	CN
DN002	Tom	18	CN
DN003	Jane	24	IND
DN004	Shawn	20	ML

AND operator allowed.

Result

Id	Name	Age	Race
DN001	John Chia	21	CN

Select - Or

- SELECT * FROM Customers WHERE Race ='CN' OR Age >=21

The database table "Customers"

Id	Name	Age	Race
DN001	John Chia	21	CN
DN002	Tom	18	CN
DN003	Jane	24	IND
DN004	Shawn	20	ML

OR operator allowed

Result

Id	Name	Age	Race
DN001	John Chia	21	CN
DN002	Tom	18	CN
DN003	Jane	24	IND

Select - Between

- SELECT * FROM Customers WHERE Age BETWEEN 21 AND 24

The database table “Customers”

Id	Name	Age	Race
DN001	John Chia	21	CN
DN002	Tom	18	CN
DN003	Jane	24	IND
DN004	Shawn	20	ML

BETWEEN allows specification of a range of data

Result

Id	Name	Age	Race
DN001	John Chia	21	CN
DN003	Jane	24	IND

Select - Like

- SELECT * FROM Customers WHERE Name LIKE 'Chia%'

The database table "Customers"

Id	Name	Race
DN001	John Chia	CN
DN002	Chia Shoo	CN
DN003	Chia Sun	IND
DN004	Shawn	ML

% acts as a wildcard, to be used with LIKE

Result

Id	Name	Race
DN002	Chia Shoo	CN
DN003	Chia Sun	IND

Select - Like

- SELECT * FROM Customers WHERE Name LIKE '%Chia%'

The database table "Customers"

Id	Name	Race
DN001	John Chia	CN
DN002	Chia Shoo	CN
DN003	Chia Sun	IND
DN004	Shawn	ML

Result

Id	Name	Race
DN001	John Chia	CN
DN002	Chia Shoo	CN
DN003	Chia Sun	IND

Select - In

- SELECT * FROM Customers WHERE Race IN ('ML','IND')

The database table "Customers"

Id	Name	Race
DN001	John Chia	CN
DN002	Tom	CN
DN003	Jane	IND
DN004	Shawn	ML

IN specifies a set of valid values

Result

Id	Name	Race
DN003	Jane	IND
DN004	Shawn	ML

Select – Sub query

- SELECT * FROM Customers WHERE CustId IN **(SELECT CustId FROM Orders)**

The database table "Customers"

CustId	Phone No
C1	4831
C2	4832
C3	4833

Sub query

The database table "Orders"

OrderId	CustId	Order Date
T01	C1	1 May 06
T02	C1	2 May 06
T03	C2	14 May 06
T04	C4	16 May 06
T05	C5	18 May 06

Result

CustId	Phone No
C1	4831
C2	4832

Sub query is executed first, returning a set of row(s)

Exercise -Dafesty

Construct the SQL statement (Database – *Dafesty*) for the followings.

1) Retrieve the different *Rating* from the Movie table
2) Retrieve all *VideoCode* and *MovieTitle* from Movie table with *Rating* = 'PG' or 'R'
3) Retrieve all *VideoCode* and *Movie*Title from Movie table with *TotalStock* greater than 4
4) Retrieve *CustomerName* who has made a transaction (IssueTran)
   - Hint: use subquery for Customer and Issuetran tables
5) Retrieve *CustomerName* who has made a transaction (IssueTran) for the videocode = 27

Select - Null

- SELECT * FROM Customers WHERE Email IS NULL

The database table “Customers”

Id	Name	Email
DN001	John Chia	*NULL*
DN002	Chia Shoo	Shoo@gmail.com
DN003	Chia Sun	*NULL*
DN004	Shawn	shawn@gmail.com

IS NULL – retrieving rows whereby the value of the Column is null

Result

Id	Name	Email
DN001	John Chia	*NULL*
DN003	Chia Sun	*NULL*

Select – Order By

- SELECT * FROM Customers ORDER BY Age

The database table “Customers”

Id	Name	Age
DN001	John Chia	21
DN002	Tom	18
DN003	Jane	24

Order By specifies rows retrieved are to be sorted

Result

Id	Name	Age
DN002	Tom	18
DN001	John Chia	21
DN003	Jane	24

Select – Order By

- SELECT * FROM Customers ORDER BY Age DESC

The database table “Customers”

Id	Name	Age
DN001	John Chia	21
DN002	Tom	18
DN003	Jane	24

Displayed in descending order

Result

Id	Name	Age
DN003	Jane	24
DN001	John Chia	21
DN002	Tom	18

Select - Count

- SELECT Count(*) AS Total FROM Customers

The database table “Customers”

Id	Name	Age
DN001	John Chia	21
DN002	Tom	18
DN003	Jane	24

Count function returns the number of rows that satisfy the query command

Result

Total
3

Select - Average

- SELECT AVG(Age) AS AvgAge FROM Customers

The database table “Customers”

Id	Name	Age
DN001	John Chia	21
DN002	Tom	18
DN003	Jane	24

Avg function returns the average of the values found in all the rows of the specified column

Result

AvgAge
21

String Concatenation

- `SELECT 'Hello, participants!' + ' Good morning'`

Adds two or more strings together

	(No column name)
1	Hello, participants!Good morning

Working with Date

- `SELECT OrderID, OrderDate FROM Orders WHERE OrderDate='1996-07-26'`

Retrieve records based on certain date

	OrderID	OrderDate
1	10266	1996-07-26 00:00:00.000

Topics

- Introduction
- Basic Queries
- **Basic Queries (Illustrations with Dafesty Video Rental Shop)**
- Advanced Queries
- Advanced Queries (Illustrations with Dafesty Video Rental Shop)

Video Rental Shop ERD

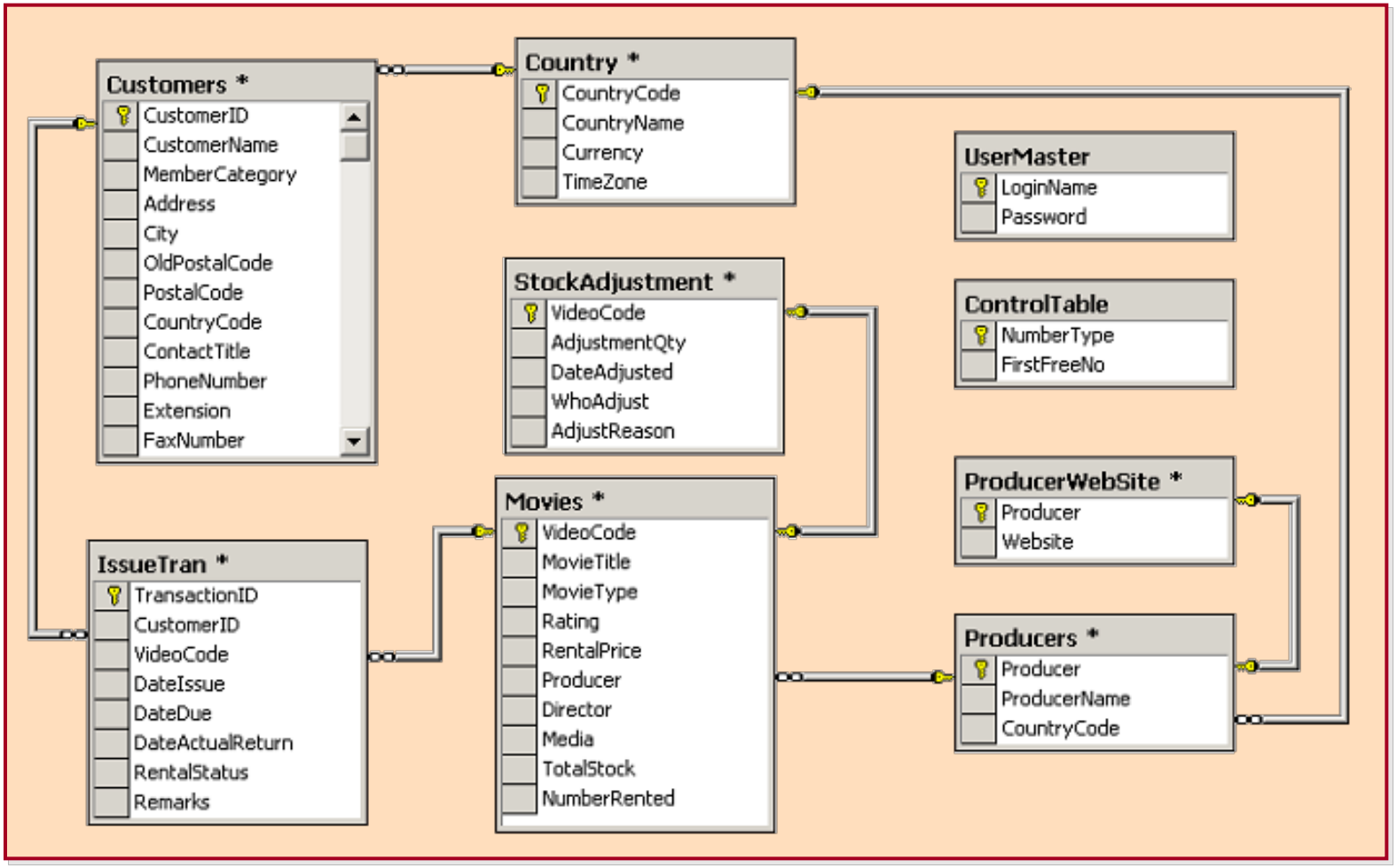


Table Meanings

- Movies Table
  - Stores all details pertaining to Movies like MovieTitle,MovieType,Rating,etc
- Customers Table
  - Stores all details pertaining to Customers like CustomerName, Address, etc
- IssueTran Table
  - Is a transaction table to record Video loan activities. A new record is created whenever a loan issue takes place. The same record is updated to a different status when the borrowed Video is returned.
  - Contains details like VideoCode,CustomerID, DateIssue, RentalStatus (in / out), ReturnDate, etc.
- Country Table
  - Stores a list of countries.
  - This is used for reference purposes for customer address.

Table Meanings

- StockAdjustment Table
  - This table is provided to capture adjustments in stock usually performed when annual stock verification takes place.
- Producers Table
  - Stores details regarding Movie Producers like ProducerName, CountryCode,etc.
- ProducerWebSite Table
  - Stores the website details for Movie Producers.
  - Only those Movie Producers who have websites are included in this table.
- ControlTable Table
  - Stores the first free number for generating serial numbers where required.
  - For instance, a new Transaction ID in the IssueTran Table may be created based on the first free number value that is kept in this table.
- User Table
  - Stores the **Application** Users' name and password.
  - This may be used for application level authentication to use the Video Rental System written in ESNET module.

Additional Dafesty Tables

- In addition to the tables shown on the ERD, the following tables are provided for exercises:
  - Documentaries Table
    - Stores Video details (almost identical to Movies Table)
    - This table is used for **Non-Movie** Video Tapes
  - Employees Table
    - Stores details of all Employees in Dafesty Video Rental Private Ltd.
    - Details include name, age, salary, etc.
  - SalaryHistory Table
    - Stores historical salary details of each employee.
    - A new record is created capturing the previous salary whenever a salary adjustment is made to an employee.

SQL Query

Select all rows and columns from a Table

```
SELECT * FROM Movies
```

- This statement retrieves all data from movies Table
  - SELECT indicates this is a data retrieval operation
  - The * stands for “all columns”
  - Movies is the name of the Table where the data resides
  - All rows in the Table will be retrieved since there are no conditions stipulated

SQL Query Output

Select all rows and columns from a Table

```
SELECT * FROM Movies
```

Output:

	VideoCode	MovieTitle	MovieType	Rating	RentalPri
1	1	Star Trek 3: Search f...	Sci-fi	PG	1.5
2	2	Star Trek 4: The Voya...	Sci-fi	PG	1.5
3	3	Star Trek 5: The Fina...	Sci-fi	PG	1.5
4	4	Demolition Man	Action	R	1.5
5	5	Nemesis	Action	R	1.5
6	6	Full Eclipse	Action	R	1.5
7	7	Marked for Death	Action	U	1.5

SQL Query

Project: Select a vertical subset of the Table

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies
```

- This statement retrieves five columns from the movies Table
  - This statement demonstrates the retrieval of selected columns (not all columns) from a given Table
  - The columns are VideoCode, MovieTitle, MovieType, Rating, TotalStock
  - The sequence in which the columns are displayed is the same as specified in the column list.
  - All rows in the Table is retrieved since there are no conditions stipulated

SQL Query Output

Project: Select a vertical subset of the Table

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies
```

Five Columns Displayed. Sequence of column matches sequence listed in Select Statement

Output:

	VideoCode	MovieTitle	MovieType	Rating	TotalStock
1	1	Star Trek 3: Search f...	Sci-fi	PG	7
2	2	Star Trek 4: The Voya...	Sci-fi	PG	0
3	3	Star Trek 5: The Fina...	Sci-fi	PG	3
4	4	Demolition Man	Action	R	3
5	5	Nemesis	Action	R	4
6	6	Full Eclipse	Action	R	6
7	7	Marked for Death	Action	U	3

Records Affected: 33

SQL Query

Project: Select a DISTINCT vertical subset of the Table

```
SELECT DISTINCT Rating
FROM Movies
```

- This statement retrieves a column from movies Table
  - This statement demonstrates the retrieval of selected column(s) from a given Table
  - DISTINCT requests that the data displayed have no duplicated row(s)

SQL Query Output

Project: Select a DISTINCT vertical subset of the Table

```
SELECT DISTINCT Rating
FROM Movies
```

Note: Only Ratings that are unique in the column will be displayed

Output:

	Rating
1	PG
2	R
3	U

Records Affected: 3

SQL Query

Selection: Select a horizontal subset from the Table

```
SELECT * FROM Movies
WHERE Rating = 'PG'
```

- This statement retrieves specific rows from movies Table
  - Unlike earlier commands, where we obtained all rows, we now want to display only certain rows.
  - A condition has to be specified in the WHERE clause
  - Only rows that satisfy the condition is retrieved
  - In our database, only 126 out of 312 Movies satisfy the condition.

SQL Query Output

Selection: Select a horizontal subset from the Table

```
SELECT * FROM Movies
WHERE Rating = 'PG'
```

Note: Only records that have 'PG' rating is displayed

Output:

	VideoCode	MovieTitle	MovieType	Rating	RentalPri
1	1	Star Trek 3: Search f...	Sci-fi	PG	1.5
2	2	Star Trek 4: The Voya...	Sci-fi	PG	1.5
3	3	Star Trek 5: The Fina...	Sci-fi	PG	1.5
4	12	The Last Starfighter	Sci-fi	PG	1.5
5	13	UHF	Comedy	PG	1.5
6	14	Firebirds	Action	PG	2.0
7	16	The Hunt for Red October	Action	PG	2.0

Records Affected: 126

SQL Query

Selection + Projection: Select a vertical and horizontal subset from the Table

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies
WHERE Rating = 'PG'
```

- This statement retrieves the specified columns and rows from Movies Table
  - This is a combination of earlier two queries that help us to obtain a vertical as well as horizontal subset.
  - The Table retrieved consists of 5 columns that satisfy the conditions in the WHERE clause

SQL Query Output

Selection + Projection: Select a vertical and horizontal subset from the Table

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies
WHERE Rating = 'PG'
```

Output:

	VideoCode	MovieTitle	MovieType	Rating	TotalStoc
1	1	Star Trek 3: Search f...	Sci-fi	PG	7
2	2	Star Trek 4: The Voya...	Sci-fi	PG	0
3	3	Star Trek 5: The Fina...	Sci-fi	PG	3
4	12	The Last Starfighter	Sci-fi	PG	7
5	13	UHF	Comedy	PG	5
6	14	Firebirds	Action	PG	2
7	16	The Hunt for Red October	Action	PG	2

Records Affected: 126

SQL Query

NOT Condition

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies
WHERE NOT(Rating = 'PG')
```

- This statement retrieves the specified columns and rows from Movies Table
  - Like the previous example, the Table retrieved consists of 5 columns that satisfy the conditions in the WHERE clause
  - The NOT clause modifies the condition used in the SQL statement.

SQL Query Output

NOT Condition

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies
WHERE NOT(Rating = 'PG')
```

The Records retrieved by this SQL query is mutually exclusive from the records retrieved by the previous SQL query. This SQL Query retrieve 186 records while the previous SQL query retrieved 126 records. Together, they are equivalent to the total number of records in the Movies Table

Output:

	VideoCode	MovieTitle	MovieType	Rating	TotalStock
1	4	Demolition Man	Action	R	3
2	5	Nemesis	Action	R	4
3	6	Full Eclipse	Action	R	6
4	7	Marked for Death	Action	U	3
5	8	Black Rain	Drama	R	4
6	9	Red Heat	Action	R	4
7	10	Die Hard	Action	R	2

Records Affected: 186

SQL Query

Using Alias: AS

```
SELECT MovieTitle AS Movie_Titles
FROM Movies
```

- This statement retrieves a column from movies Table
  - This statement demonstrates the retrieval of selected column(s) from a given Table
  - DISTINCT requests that the data displayed have no duplicated row(s)
  - AS redefines the name of the result set column.(ie Movie_Title). It also allows the assignment of names to result set columns that have no names. (eg. Derived results set columns have no names)

SQL Query Output

Using Alias: AS

```
SELECT MovieTitle AS Movie_Titles
FROM Movies
```

Output:

Columns' Name have changed to Movie_Titles

	Movie_Titles
1	Star Trek 3: Search for Spock
2	Star Trek 4: The Voyage Home
3	Star Trek 5: The Final Frontier
4	Demolition Man
5	Nemesis
6	Full Eclipse
7	Marked for Death

Records Affected: 312

SQL Query

AND condition

```
SELECT * FROM Movies
WHERE Rating = 'PG'
AND MovieType = 'Sci-fi'
```

- This statement retrieves all data from Movies Table that satisfy the conditions in the WHERE clause
  - AND allows more conditions to be placed in the SELECT statement thereby limiting the total number of rows returned

SQL Query Output

AND condition

```
SELECT * FROM Movies
WHERE Rating = 'PG'
AND MovieType = 'Sci-fi'
```

Output:

Notice that only Sci-fi Movie Type with 'PG' rating is selected

	VideoCode	MovieTitle	MovieType	Rating	RentalPri
1	1	Star Trek 3: Search f...	Sci-fi	PG	1.5
2	2	Star Trek 4: The Voya...	Sci-fi	PG	1.5
3	3	Star Trek 5: The Fina...	Sci-fi	PG	1.5
4	12	The Last Starfighter	Sci-fi	PG	1.5
5	88	Dune	Sci-fi	PG	2.0
6	97	My Science Project	Sci-fi	PG	2.0
7	117	Star Trek 6: The Undi...	Sci-fi	PG	2.0

Records Affected: 17

SQL Query

OR condition

```
SELECT * FROM Movies
WHERE Rating = 'U'
OR MovieType = 'Sci-fi'
```

- This statement retrieves all data from Movies Table that satisfy the conditions in the WHERE clause
  - OR allows either one of the conditions placed before and after it to be satisfied for any row in the Table to be retrieved

SQL Query Output

OR condition

```
SELECT * FROM Movies
WHERE Rating = 'U'
OR MovieType = 'Sci-fi'
```

Output:

Notice only Sci-fi Movies or Movies with U Rating are displayed

	VideoCode	MovieTitle	MovieType	Rating	RentalPrice	Produc
1	1	Star Trek 3: Search f...	Sci-fi	PG	1.5	Fu Sho
2	2	Star Trek 4: The Voya...	Sci-fi	PG	1.5	Kelvin
3	3	Star Trek 5: The Fina...	Sci-fi	PG	1.5	Neo Ke
4	12	The Last Starfighter	Sci-fi	PG	1.5	Abdul
5	51	Blood Ties	Drama	U	2.0	Consta
6	54	Freejack	Sci-fi	PG	2.0	Ang Ki
7	88	Dune	Sci-fi	PG	2.0	Koh Ti

Records Affected: 134

SQL Query

BETWEEN condition

```
SELECT * FROM IssueTran
WHERE DateIssue BETWEEN '20 Nov 2000' AND
'23 Nov 2000'
```

- This statement retrieves all data from IssueTran Table that satisfy the conditions in the WHERE clause
  - The BETWEEN keyword allows conditions, in the WHERE clause, to be defined as a range
  - SQL Server flexibly accepts different date formats
    - Refer to Books online for other formats

SQL Query Output

BETWEEN condition

```
SELECT * FROM IssueTran
WHERE DateIssue BETWEEN '20 Nov 2000' AND
'23 Nov 2000'
```

Output:

Note only 199 out of 400 rows are Retrieved

	TransactionID	CustomerID	Vid...	DateIssue	DateDue
1	2	4312	55	2000-11-20 00:00:00	2000-11
2	3	6598	94	2000-11-20 00:00:00	2000-11
3	4	1111	164	2000-11-20 00:00:00	2000-11
4	5	4312	291	2000-11-20 00:00:00	2000-11
5	6	8756	1	2000-11-20 00:00:00	2000-11
6	7	7856	254	2000-11-20 00:00:00	2000-11
7	8	8756	232	2000-11-20 00:00:00	2000-11
8	9	1000	299	2000-11-20 00:00:00	2000-11

Records Affected: 199

SQL Query

LIKE condition

```
SELECT MovieTitle
FROM Movies
WHERE MovieTitle LIKE 'Star%'
```

String Pattern

- This statement retrieves all data from Movies Table that satisfy the conditions in the WHERE clause
  - LIKE allow rows having values in the specified columns that matches the string pattern (ie Star%) to be retrieved
  - The "%" acts as a wildcard for other character(s) in the pattern. For example the wild card may be "s" (ie. Stars) or "dom" (ie Stardom)

SQL Query Output

LIKE condition

```
SELECT MovieTitle
FROM Movies
WHERE MovieTitle LIKE 'Star%'
```

Output:

	MovieTitle
1	Star Trek 3: Search for Spock
2	Star Trek 4: The Voyage Home
3	Star Trek 5: The Final Frontier
4	Star Wars: Making of a Saga
5	Star Trek 6: The Undiscovered Country
6	Star Trek 2: The Wrath of Khan
7	Star Trek 1: The Motion Picture
8	Star Trek: Generations

Records Affected: 11

SQL Query

LIKE condition

```
SELECT MovieTitle FROM Movies
WHERE MovieTitle LIKE '%Star%'
```

- This statement retrieves all data from Movies Table that satisfy the conditions in the WHERE clause
  - The wildcard (%) placed before and after the string pattern (ie Star) allows values in the specified columns (ie MovieTitle column) of the Movies Table to be retrieved as long as the column string has the character(s) having the same string pattern (ie. Star)
  - The string pattern is not Cap Sensitive

SQL Query Output

LIKE condition (with different sets of % sign)

```
SELECT MovieTitle FROM Movies
WHERE MovieTitle LIKE '%Star%'
```

Output:

	MovieTitle
1	Star Trek 3: Search for Spock
2	Star Trek 4: The Voyage Home
3	Star Trek 5: The Final Frontier
4	The Last Starfighter
5	Star Wars: Making of a Saga
6	Firestarter
7	Star Trek 6: The Undiscovered Country
8	Star Trek 2: The Wrath of Khan

Records Affected: 13

SQL Query

IN condition

```
SELECT * FROM Movies
WHERE Rating IN ('U','R')
```

- This statement retrieves all data from Movies Table that satisfy the conditions in the WHERE clause
  - The IN clause specifies a list of character(s) that values in the specified column (Rating column) should match.

SQL Query Output

IN condition

```
SELECT *
FROM Movies
WHERE Rating IN ('U','R')
```

Output:

	VideoCode	MovieTitle	MovieType	Rating	RentalPr
1	4	Demolition Man	Action	R	1.5
2	5	Nemesis	Action	R	1.5
3	6	Full Eclipse	Action	R	1.5
4	7	Marked for Death	Action	U	1.5
5	8	Black Rain	Drama	R	1.5
6	9	Red Heat	Action	R	1.5
7	10	Die Hard	Action	R	1.5
8	11	Die Hard 2	Action	R	1.5

Records Affected: 186

SQL Query

Using Sub Queries

```
SELECT * FROM Producers
WHERE Producer IN(SELECT Producer
                  FROM ProducerWebSite)
```

- This statement retrieves all data from Producers Table that satisfy the conditions in the WHERE clause
  - The IN clause specifies a list of values, generated by another SELECT statement, that values in the specified column (ie Producer column from Producers Table) should match.
  - The SELECT statement in brackets are called sub query.

SQL Query Output

Using Sub Queries

```
SELECT * FROM Producers
WHERE Producer IN(SELECT Producer
                  FROM ProducerWebSite)
```

Output:

	Producer	ProducerName	CountryCode
1	20th	20th Century Fox Prod...	UK
2	Columbia	Columbia Pictures Pro...	UK
3	Universal	Universal Studio Prod...	UK

Records Affected: 3

SQL Query

IS NULL condition

```
SELECT CustomerID,CustomerName,EmailAddress
FROM Customers
WHERE EmailAddress IS NULL
```

- This statement retrieves three columns from Customers Table that satisfy the conditions in the WHERE clause
  - IS NULL specifies that the rows in the Table whose value in the specified column's must be a null in order for that row to be retrieved

SQL Query Output

IS NULL condition

```
SELECT CustomerID,CustomerName,EmailAddress
FROM Customers
WHERE EmailAddress IS NULL
```

Output:

	CustomerID	CustomerName	EmailAddress
1	2131	Jon	NULL
2	2323	Richard Kwan	NULL
3	2345	Ng Teck Kie Anthony	NULL
4	2626	Steven Teo	NULL
5	5156	Lee Boon Kiat	NULL
6	6969	jason young	NULL
7	7856	Rajaram Venkatesh	NULL
8	8888	Kelvin Koh	NULL

Records Affected: 8

SQL Query

ORDER BY clause

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies ORDER BY MovieTitle
```

- This statement retrieves five columns from movies Table that satisfy the conditions in the WHERE clause
  - ORDER BY specifies that rows retrieved are to be sorted in ascending order in the specified column (ie. MovieTitle)
  - ORDER BY, by default, sorts rows in ascending order. To sort by descending order, use "…. ORDER BY MovieTitle DESC"
  - ORDER BY allows sorting to be done in two or more columns. However, the first column that appears in the statement will have priority over the others. Eg. "ORDER BY MovieTitle, VideoCode"

SQL Query Output

ORDER BY clause

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies ORDER BY MovieTitle
```

Output:

Note: Sorted in Ascending order of Movie Title

	VideoCode	MovieTitle	MovieType	Rating	TotalStoc
1	154	A Few Good Men	Drama	R	1
2	141	A League of Their Own	Drama	PG	5
3	49	Above the Law	Action	R	4
4	224	Absolute Power	Drama	R	1
5	115	Addams Family	Comedy	PG	4
6	27	Air America	Comedy	R	4
7	189	Airheads	Comedy	R	6
8	238	Aladdin	Animation	U	3

Records Affected: 312

SQL Query

ORDER BY clause (Descending order)

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies ORDER BY MovieTitle DESC
```

- This statement retrieves five columns from movies Table that satisfy the conditions in the WHERE clause
  - ORDER BY specifies that rows retrieved are to be sorted in descending order in the specified column (ie. MovieTitle)
  - ORDER BY allows sorting to be done in two or more columns. However, the first column that appears in the statement will have priority over the others. Eg. “ORDER BY MovieTitle, VideoCode”

SQL Query Output

ORDER BY clause (Descending order)

```
SELECT VideoCode,MovieTitle,MovieType,Rating,TotalStock
FROM Movies ORDER BY MovieTitle DESC
```

Output:

Note: Sorted in Descending order of Movie Title

	VideoCode	MovieTitle	MovieType	Rating	TotalStock
1	282	X-Files (Wetwired)	Drama	PG	4
2	275	X-Files (War of the C...	Drama	PG	4
3	264	X-Files (Tooms)	Drama	PG	6
4	258	X-Files (The Host)	Drama	PG	3
5	260	X-Files (The Erlenmey...	Drama	PG	1
6	272	X-Files (The Blessing Way	Drama	PG	6
7	283	X-Files (Talitha Cumi)	Drama	PG	6

Records Affected: 312

SQL Query

COUNT function

```
SELECT COUNT(*) AS TotalNoOfMovies FROM Movies
```

- This statement retrieves all data from Movies Table
  - The COUNT function returns the “number of rows” that satisfy the query command : “SELECT * FROM Movies”
  - Using alias allows the naming of the columns that are derived.
  - COUNT and other functions are usually used in conjunction with GROUP BY clauses.

SQL Query Output

COUNT function

```
SELECT COUNT(*) AS TotalNoOfMovies FROM Movies
```

Output:

	TotalNoOfMOvies
1	312

Records Affected: 312

SQL Query

COUNT function

```
SELECT COUNT(Director) AS DirectorsInMovies
FROM Movies
```

- This statement retrieves all data from Movies Table
  - The COUNT function returns the “number of rows” that satisfy the query command : “SELECT * FROM Movies”
  - Using alias allows the naming of the columns that are derived.
  - COUNT and other functions are usually used in conjunction with GROUP BY clauses.

SQL Query Output

COUNT function

```
SELECT COUNT(Director) AS DirectorsInMovies
FROM Movies
```

Output:

	DirectorsInMovies
1	311

Records Affected: 311

SQL Query

SUM function

```
SELECT SUM(TotalStock) FROM Movies
```

- This statement retrieves one columns from Movies Table
  - This column returns the sum of the values found in all the rows of the specified column (ie TotalStock column) of the Movies Table that satisfy the query: “SELECT * FROM Movies”
  - The specified column (ie TotalStock column) must have a data type that allows arithmetic operations
  - SUM and other functions are usually used in conjunction with GROUP BY clauses.

SQL Query Output

SUM function

```
SELECT SUM(TotalStock) FROM Movies
```

Output:

	(No column name)
1	1098

Records Affected: 1

SQL Query

AVG function

```
SELECT AVG(RentalPrice) FROM Movies
```

- This statement retrieves one columns from Movies Table
  - This column returns the Average of the values found in all the rows of the specified column (ie RentalPrice column) of the Movies Table that satisfy the query: ““SELECT * FROM Movies”
  - The specified column (ie RentalPrice column) must have a data type that allows arithmetic operations
  - AVG and other functions are usually used in conjunction with GROUP BY clauses.

SQL Query Output

AVG function

```
SELECT AVG(RentalPrice) FROM Movies
```

Output:

	(No column name)
1	1.8621794871794872

Records Affected: 1

Topics

- Introduction
- Basic Queries
- Basic Queries (Illustrations with Dafesty Video Rental Shop)
- **Advanced Queries**
- Advanced Queries (Illustrations with Dafesty Video Rental Shop)

Inner Join (1)

- How do we retrieve name, id of the customers and video code they have rented?

table "Customers"

Id	Name
100	Chia
101	Derek
102	Esther
103	Venkat

table "IssueTrans"

TransId	CustomerId	VideoCode
T01	100	V01
T02	100	V02
T03	101	V11

Need to access values from two tables

Inner Join (2)

- SELECT C.Id, C.Name, T.VideoCode FROM Customers C, IssueTrans T WHERE C.Id = T.CustomerId

table "Customers"

Id	Name
100	Chia
101	Derek
102	Esther
103	Venkat

table "IssueTrans"

TransId	CustomerId	VideoCode
T01	100	V01
T02	100	V02
T03	101	V11

Id	Name	VideoCode
100	Chia	V01
100	Chia	V02
101	Derek	V11

Result

Inner join retrieve rows whose values exists in both tables

Left Outer Join (1)

- How do I list details of all customers, and include video code for those who have made orders?

```
SELECT C.Id, C.Name, T.VideoCode
FROM Customers C LEFT OUTER JOIN IssueTrans T
ON C.Id = T.CustomerId
```

Left table

Right table

Left Outer Join retrieved all rows from the left table and insert values from the right table (insert null if not found in the right table).

Left Outer Join (2)

- SELECT C.Id, C.Name, T.VideoCode FROM Customers C LEFT OUTER JOIN IssueTrans T ON C.Id = T.CustomerId

table "Customers"

Id	Name
100	Chia
101	Derek
102	Esther
103	Venkat

table "IssueTrans"

TransId	CustomerId	VideoCode
T01	100	V01
T02	100	V02
T03	101	V11

Id	Name	VideoCode
100	Chia	V01
100	Chia	V02
101	Derek	V11
102	Esther	*null*
103	Venkat	*null*

Result

Self Join (1)

- How do I list the employees name with their supervisor name?

The database table “Employees”

Id	Name	ReportsTo
01	Lim	*Null*
02	MK Leong	01
03	Venkat	02
04	Derek	02

Self Join (2)

- SELECT staff.name, supervisor.name FROM Employees staff, Employees supervisor WHERE staff.reportsto = supervisor.id

The database table “Employees”

Id	Name	ReportsTo
01	Lim	*Null*
02	MK Leong	01
03	Venkat	02
04	Derek	02
05	Esther	02

self join:
the table joins with itself

Result

Name	Name
MK Leong	Lim
Venkat	MK Leong
Derek	MK Leong
Esther	MK Leong

Self Join (3)

- SELECT staff.name, supervisor.name FROM Employees staff, Employees supervisor WHERE staff.reportsto = supervisor.id

The database table "Employees"

Id	Name	ReportsTo
01	Lim	*Null*
02	MK Leong	01
03	Venkat	02
04	Derek	02
05	Esther	02

The staff "Lim" is not selected under staff column. How to reconstruct the query to include "Lim" in the staff list?

Result

supervisor

staff

Name	Name
MK Leong	Lim
Venkat	MK Leong
Derek	MK Leong
Esther	MK Leong

Group By

- SELECT CountryCode FROM Customers GROUP BY CountryCode

The database table "Customers"

Name	CountryCode	Category
Chia	SIN	A
Derek	SIN	B
Esther	SIN	B
Venkat	IND	A

Result

CountryCode
SIN
IND

list the different country codes

Group By with Count

- SELECT CountryCode, Count(*) AS Num FROM Customers GROUP BY CountryCode

The database table "Customers"

Name	CountryCode	Category
Chia	SIN	A
Derek	SIN	B
Esther	SIN	B
Venkat	IND	A

List the country codes with the number of customers

Result

CountryCode	Num
SIN	3
IND	1

Group By and Having

- SELECT CountryCode, Count(*) AS Num FROM Customers GROUP BY CountryCode HAVING COUNT(*) > 1

The database table "Customers"

Name	CountryCode	Category
Chia	SIN	A
Derek	SIN	B
Esther	SIN	B
Venkat	IND	A

list the country codes whereby there are more than 1 customers from the country

Result

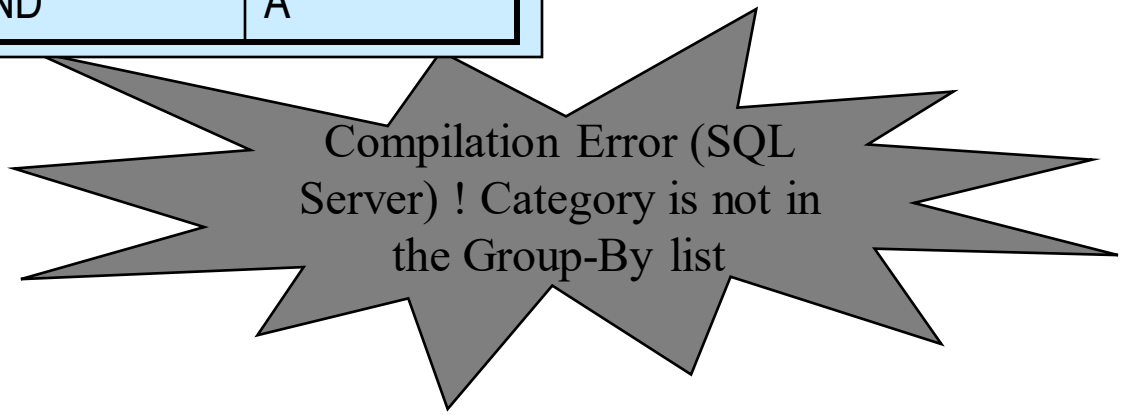
CountryCode	Num
SIN	3

Group By

- SELECT CountryCode, Category FROM Customers GROUP BY CountryCode

The database table "Customers"

Name	CountryCode	Category
Chia	SIN	A
Derek	SIN	B
Esther	SIN	B
Venkat	IND	A



Group By and Count

- How do we retrieve the country code with more customers than “IND”?

The database table “Customers”

Name	CountryCode	Category
Chia	SIN	A
Derek	SIN	B
Esther	SIN	B
Venkat	IND	A

```
SELECT CountryCode FROM Customers
GROUP BY CountryCode
HAVING Count(*) > (SELECT Count(*) FROM Customers
                   WHERE CountryCode = 'IND')
```

Join And Group By

- SELECT C.Id, Count(*) As Num FROM Customers C, IssueTrans T WHERE C.Id = T.CustomerId GROUP BY C.Id

table "Customers"

Id	Name
100	Chia
101	Derek
102	Esther
103	Venkat

table "IssueTrans"

TransId	CustomerId	VideoCode
T01	100	V01
T02	100	V02
T03	101	V11

Id	Num
100	2
101	1

Result

Union

- SELECT * FROM USA-Customers UNION SELECT * FROM Europe-Customers

The database table “USA-Customers”

CustId	Phone No
U1	4831
U2	4832

The database table “Europe-Customers”

CustId	Phone No
E1	305656
E2	456677

Result

CustId	Phone No
U1	4831
U2	4832
E1	305656
E2	456677

Union return a merger of multiple tables columns with compatible data type

Top

- SELECT TOP 2 CustomerID FROM Customers

The database table “Customers”

CustId	Phone No
U1	4831
U2	4832
U3	4678
U4	4568

TOP *n* returns the first nth rows in the results

Result

CustId
U1
U2

Topics

- Introduction
- Basic Queries
- Basic Queries (Illustrations with Dafesty Video Rental Shop)
- Advanced Queries
- **Advanced Queries (Illustrations with Dafesty Video Rental Shop)**

SQL Query

GROUP BY clause

```
SELECT MovieType
FROM Movies
GROUP BY MovieType
```

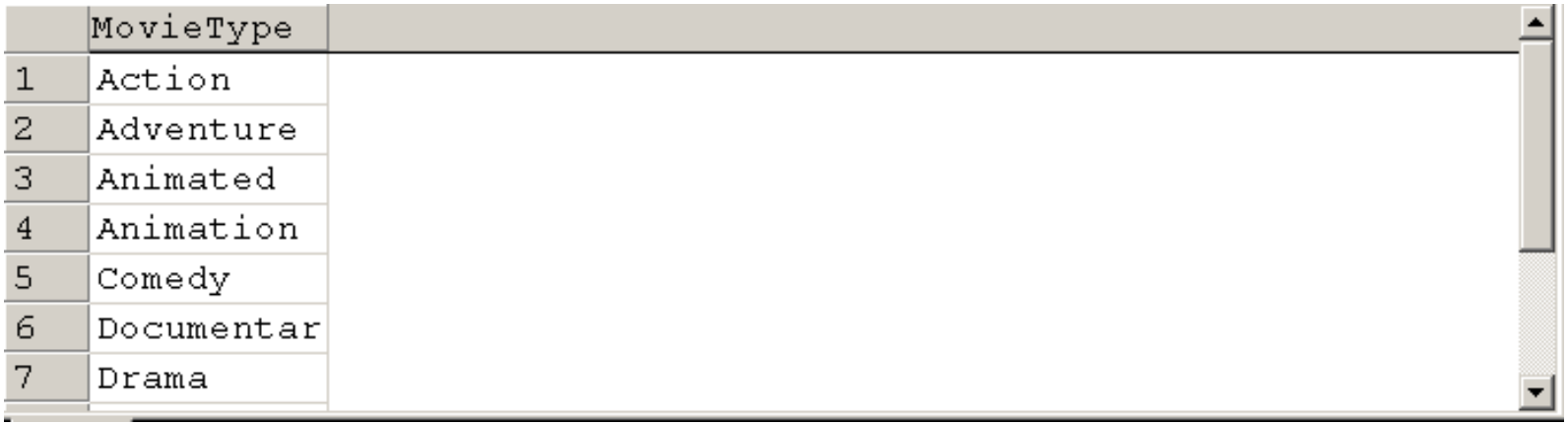
- This statement retrieves a column from Movies Table
  - GROUP BY aggregates the returned rows that have the same value in the column (MovieType column) specified in the GROUP BY clause
  - GROUP BY is constantly used with functions such as COUNT, SUM and AVG

SQL Query Output

GROUP BY clause

```
SELECT MovieType
FROM Movies
GROUP BY MovieType
```

Output:



	MovieType
1	Action
2	Adventure
3	Animated
4	Animation
5	Comedy
6	Documentar
7	Drama

Records Affected: 33

SQL Query

GROUP BY clause with COUNT function

```
SELECT COUNT(MovieType) AS NumberOfMovies, MovieType
FROM Movies
GROUP BY MovieType
```

- This statement retrieves a column from Movies table
  - GROUP BY aggregates the returned rows that have the same value in the column (MovieType column) specified in the GROUP BY clause
  - COUNT(MovieType) in this case returns the “number of rows” aggregated for each returned value in MovieType column

SQL Query Output

GROUP BY clause with COUNT function

```
SELECT COUNT(MovieType) AS NumberOfMovies,MovieType
FROM Movies
GROUP BY MovieType
```

Output:

	NumberOfMovies	MovieType
1	83	Action
2	10	Adventure
3	4	Animated
4	8	Animation
5	51	Comedy
6	1	Documentar
7	121	Drama
8	4	Horror

Records Affected: 33

SQL Query

HAVING clause

```
SELECT COUNT(MovieType) AS NumberOfMovies,MovieType
FROM Movies
GROUP BY MovieType
HAVING COUNT(MovieType) > 50
```

- This statement retrieves a column from Movies Table
  - COUNT(…) aggregates the returned rows based on each row having the same values as other rows with respect to a specified column(s) (MovieType column)
  - HAVING specifies a search condition that will be factored in the rows returned after the GROUP BY clause has taken effect

SQL Query Output

HAVING clause

```
SELECT COUNT(MovieType) AS NumberOfMovies,MovieType
FROM Movies
GROUP BY MovieType
HAVING COUNT(MovieType) > 50
```

Output:

	NumberOfMovies	MovieType
1	83	Action
2	51	Comedy
3	121	Drama

Records Affected: 3

SQL Query

SUM function with GROUP BY

```
SELECT SUM(TotalStock) AS TotalNumberOfMovies,Rating
FROM Movies
GROUP BY Rating
```

- This statement retrieves two columns from Movies Table
  - The two columns are TotalStock and Rating
  - The SUM function adds up all the values in the TotalStock column where the same Rating are grouped together
  - TotalStock column must have a data type that allows arithmetic operations

SQL Query Output

SUM function with GROUP BY

```
SELECT SUM(TotalStock) AS TotalNumberOfMovies,Rating
FROM Movies
GROUP BY Rating
```

Output:

	TotalNumberOfMovies	Rating
1	486	PG
2	549	R
3	63	U

Records Affected: 3

SQL Query

AVG function with GROUP BY

```
SELECT AVG(RentalPrice) AS AveragePriceOfMovie,Rating
FROM Movies
GROUP BY Rating
```

- This statement retrieves two columns from Movies Table
  - The two columns are RentalPrice and Rating
  - The AVG function adds up all the values in the RentalPrice column, where the same Rating are grouped together, and then divides them by the number of rows used for each Rating
  - TotalStock column must have a data type that allows arithmetic operations

SQL Query Output

AVG function with GROUP BY

```
SELECT AVG(RentalPrice) AS AveragePriceOfMovie,Rating
FROM Movies
GROUP BY Rating
```

Output:

	AveragePriceOfMovie	Rating
1	1.8928571428571428	PG
2	1.8353293413173652	R
3	1.8947368421052631	U

Records Affected: 33

SQL Query

Joining Tables: INNER JOIN

```
SELECT Producers.Producer,ProducerName,WebSite
FROM Producers
INNER JOIN ProducerWebSite
ON ProducerWebSite.Producer = Producers.Producer


--ALTERNATIVE SYNTAX OF INNER JOIN

SELECT Producers.Producer,ProducerName,WebSite
FROM Producers,ProducerWebSite
WHERE ProducerWebSite.Producer = Producers.Producer
```

SQL Query

- This statement retrieves three columns from ProducerWebSite and Producers Table.
  - When retrieving rows from two or more Tables that have the same column name, qualify (ie differentiate) the column to be retrieved by giving the Table name followed by the column name (Producers.Producer)
  - Inner join is an inclusive join. It retrieved rows whose value exists in *both* Tables. In other words, if either one of the Table have a rows whose values (ie Joining Condition) is not found in the other Table, that row will not be retrieved.
  - Syntactically, INNER JOIN can be represent with just JOIN because the default JOIN is the INNER JOIN.
  - The WHERE condition can be used in lieu of the INNER JOIN. Refer to the alternative syntax.
  - The WHERE condition is not a filter as in previous cases but acts as a condition for joining. It is often referred as JOIN condition.

SQL Query Output

Joining Tables: INNER JOIN

```
SELECT Producers.Producer,ProducerName,WebSite
FROM Producers
INNER JOIN ProducerWebSite
ON ProducerWebSite.Producer = Producers.Producer
```

Output:

	Producer	ProducerName	WebSite
1	20th	20th Century Fox Prod...	www.century.com
2	Columbia	Columbia Pictures Pro...	www.columbia.com
3	Universal	Universal Studio Prod...	www.universal.com

Records Affected: 3

SQL Query

Joining Tables: LEFT OUTER JOIN

```
SELECT Producers.Producer,ProducerName,WebSite
FROM Producers
LEFT OUTER JOIN ProducerWebSite
ON Producers.Producer = ProducerWebSite.Producer
```

Left Table

- This statement retrieves three columns from ProducerWebSite and Producers Table.
  - OUTER JOIN consists of the LEFT and RIGHT outer joins.
  - The LEFT OUTER JOIN retrieved all the rows from the Table on the left side of the join clause (ie. Producers Table) and attempts to insert values from the Table (ie ProducerWebSite Table) on the right side of the clause.
  - OUTER JOIN is an exclusive join. In other words, if the column values based on the ioin condition cannot be found, NULL is displayed.

SQL Query Output

Joining Tables: LEFT OUTER JOIN

```
SELECT Producers.Producer,ProducerName,WebSite
FROM Producers
LEFT OUTER JOIN ProducerWebSite
ON Producers.Producer = ProducerWebSite.Producer
```

Output:

	Producer	ProducerName	WebSite
1	20th	20th Century Fox Prod...	www.century.com
2	Columbia	Columbia Pictures Pro...	www.columbia.com
3	George	George Lucas Production	NULL
4	Pixar	Pixar Entertainment	NULL
5	Raintree	RainTree Pictures	NULL
6	Universal	Universal Studio Prod...	www.universal.com
7	Walt	Walt Disney Studio	NULL
8	Warner	Warner Brothers Produ...	NULL

Records Affected: 8

SQL Query

Joining Tables: SELF JOIN

```
SELECT D1.VideoTitle AS DocumentaryTitle,
       D2.VideoTitle AS DocumentaryPrequel
FROM Documentaries D1
INNER JOIN Documentaries D2
ON D2.VideoCode = D1.PreviousEpisode

--ALTERNATIVE SYNTAX
SELECT D1.VideoTitle AS DocumentaryTitle,
       D2.VideoTitle AS DocumentaryPrequel
FROM Documentaries D1,Documentaries D2
WHERE D2.VideoCode = D1.PreviousEpisode
```

- This statement retrieves two columns from Documentaries Table.
  - A self join is required when some conditions require that the Table to join with itself
  - An Alias is required when performing SELF JOINS so that the same Table can be recognised as different Tables.

SQL Query Output

Joining Tables: SELF JOIN

```
SELECT D1.VideoTitle AS DocumentaryTitle,
       D2.VideoTitle AS DocumentaryPrequel
FROM Documentaries D1
INNER JOIN Documentaries D2
ON D2.VideoCode = D1.PreviousEpisode
```

Output:

	DocumentaryTitle	DocumentaryPrequel
1	Poverty in Russia 2	Poverty in Russia 1
2	Lonely Planet: Paris 2	Lonely Planet: Paris 1

Records Affected: 2

SQL Query

Joining Columns: UNION

```
SELECT MovieTitle FROM Movies
UNION
SELECT VideoTitle FROM Documentaries
```

- This statement retrieves a column from Customers and Producers Table.
  - UNION allows two or more Table columns to be merged into one as long as the datatype for the columns are compatible.
  - The values in the columns are by default DISTINCT rather than ALL. In other words, the retrieved values in the specified columns will not be repeated unless stated otherwise

SQL Query Output

Joining Columns: UNION

```
SELECT MovieTitle FROM Movies
UNION
SELECT VideoTitle FROM Documentaries
```

Output:

	MovieTitle
1	A Few Good Men
2	A League of Their Own
3	Above the Law
4	Absolute Power
5	Addams Family
6	African Elephants
7	Air America

Records Affected: 58

SQL Query

Limiting Result Set: TOP n

```
SELECT TOP 5 CustomerID,COUNT(CustomerID) AS NoOfTransMade
FROM IssueTran
GROUP BY CustomerID
ORDER BY NoOfTransMade DESC
```

- This statement retrieves two columns from IssueTran Table
  - COUNT(…) sums the number of rows in the Specified Column (ie CustomerID Column) when the GROUP BY condition is placed on the SQL query.
  - ORDER BY arranges the results in Descending order.
  - Finally, TOP n (ie 5 in the above query) displays the first nth rows in the results of the SQL query.

SQL Query Output

Limiting Result Set: TOP n

```
SELECT TOP 5 CustomerID,COUNT(CustomerID) AS NoOfTransMade
FROM IssueTran
GROUP BY CustomerID
ORDER BY NoOfTransMade DESC
```

Only Top 5 Rows of the Result Set is displayed.

Output:

	CustomerID	NoOfTransactionsMade
1	6542	14
2	8756	14
3	1003	13
4	2345	13
5	2270	11

Records Affected: 5

Sub Query vs Join

- Sub query vs Joins
  - There are a lot of situations when joins can be used in lieu of sub queries to obtain the desired result set.
  - For example, in the situation where it is required to List the Movie Titles and Ratings of all Movies borrowed
  - Using a Sub query, we have

```
SELECT Movietitle FROM Movies, Issuetran
WHERE Movies.Videocode = Issuetran.Videocode
and CustomerID = 9999
```

  - Using a Join, we have

```
SELECT MovieTitle FROM Movies
WHERE videocode IN
(SELECT Videocode FROM Issuetran WHERE Customerid=9999 )
```

Sub Query vs Join Output

Either way, the result set returned is the same.

```
SELECT Movietitle FROM Movies, Issuetran
WHERE Movies.Videocode = Issuetran.Videocode
and CustomerID = 9999
```

Output:

	MovieTitle
1	Star Wars: Making of ...
2	I Come in Peace
3	Blood Ties
4	Pacific Heights
5	Internal Affairs
6	The Substitute
7	Aladdin
8	Liar Liar

Records Affected: 8

Sub Query vs Join

- Sub query vs Joins
  - However, there are circumstances when a Join cannot be used in lieu of Sub queries.
  - One such circumstance is when an aggregation of rows is used as a condition for the query. For example:

```
SELECT MovieTitle,RentalPrice
FROM Movies
WHERE RentalPrice > (SELECT AVG(RentalPrice) FROM Movies)
```

  - Note that the result set using the above query cannot be repeated using a query with Joins.

Sub Query vs Join

- Some considerations in deciding to use Sub query vs Joins:
  - Use Join when data is drawn from two tables
  - Use sub query when comparison is made to an aggregation of second table
  - When either sub query or joins can be used:
    - Choose the one that seems natural to an English statement or to you.
    - Sub query is simplier to develop since sub query can be independently tested
    - However, the speed of retrieval using joins (on index fields) could be faster

Summary

- Revision of Key Concepts
- Discussion on Workshop
- Questions and Answer
- Assignments and Home Work

Appendix A: Advance Query

- Apart from the Left Outer Join, we can also use a Right Outer Join on tables/views for retrieving records.

Appendix A

Joining Tables: RIGHT OUTER JOIN

```
SELECT Producers.Producer,ProducerName,WebSite
FROM ProducerWebSite
RIGHT OUTER JOIN Producers
ON Producers.Producer = ProducerWebSite.Producer
```

Right Table

- This statement retrieves three columns from ProducerWebSite and Producers Table.
  - The RIGHT OUTER JOIN is similar to the LEFT OUTER JOIN except that the Table on the right of the join clause (Producers Table) will display all the rows in it irregardless of whether there is row with similar value in the other table.

Appendix A

Joining Tables: RIGHT OUTER JOIN

```
SELECT Producers.Producer,ProducerName,WebSite
FROM ProducerWebSite
RIGHT OUTER JOIN Producers
ON Producers.Producer = ProducerWebSite.Producer
```

Output:

Notice that Producer records in Producers Table that are not found in the ProducersWebSite Table will be displayed as null in the Result set

	Producer	Produce...	WebSite
1	20th	20th Century Fo...	www.century.com
2	Columbia	Columbia Pictures Pro...	www.columbia.com
3	George	George Lucas Production	NULL
4	Pixar	Pixar Entertainment	NULL
5	Raintree	RainTree Pictures	NULL
6	Universal	Universal Studio Prod...	www.universal.com
7	Walt	Walt Disney Studio	NULL
8	Warner	Warner Brothers Produ...	NULL

Records Affected: 8

Appendix B: Field Names

- If there are any field names with spaces, MS SQL Server have problems reading the field name.
  - Example of a field names with spaces are Order Details table in Northwind database.
  - In order to query the table, we need to place the field name in a pair of square brackets. (the square brackets acts as an escape sequence)
  - For example, we do not: SELECT * FROM Order Details
  - Instead we : SELECT * FROM [Order Details]

Appendix C: Functions

- This appendix serves to provide information regarding functions that are available in SQL Server.
- Functions allows users to simplify many of their operations.
  - For example users can use COUNT function to return the number of rows returned by the query.
- The Functions that will be discussed are:
  - Aggregate Functions:
    - Functions which operate on a set of records to return a single but summarizing value.
  - Scalar Functions:
    - Functions which operate on a single set of record and return a single value.
    - Some Scalar functions include Mathematical Functions, String Functions and System Functions.
  - A brief description for these functions are provided here. Refer to Help Online for more details.

Appendix C

- Aggregate Functions:
  - Commonly used Aggregate functions are as follows:

Aggregate Function	Function Description
AVG	Returns the average of the values in a group. Null values are ignored.
COUNT	Returns the number of items in a group.
MAX	Returns the maximum value in the expression.
MIN	Returns the minimum value in the expression.
SUM	Returns the sum of all the values, or only the DISTINCT values, in the expression. SUM can be used with numeric columns only. Null values are ignored.

Appendix C

- Mathematical Functions:
  - Performs a calculation based on input values provided as parameters to the function, and returns a numeric value.
  - Commonly used Mathematical functions are as

Mathematical Function	Function Description
CEILING	Returns the smallest integer greater than, or equal to, the given numeric expression.
FLOOR	Returns the largest integer less than or equal to the given numeric expression.
RAND	Returns a random **float** value from 0 through 1.
ROUND	Returns a numeric expression, rounded to the specified length or precision.

Appendix C

- Date and Time Functions:
  - These scalar functions perform an operation on a date and time input value and return a string, numeric, or date and time value.
  - Commonly used Date and Time functions are as

Date and Time Function	Function Description
DATEADD	Returns a new **datetime** value based on adding an interval to the specified date.
DATEDIFF	Returns the number of date and time boundaries crossed between two specified dates.
GETDATE	Returns the current system date and time in the Microsoft® SQL Server™ standard internal format for **datetime** values.
DAY / MONTH / YEAR	Returns an integer that represents the day / month / year part of a specified date respectively

Appendix C

- String Functions:
  - These scalar functions perform an operation on a string input value and return a string or numeric value.
  - Commonly used String functions are as follows:

String Function	Function Description
ASCII	Returns the ASCII code value of the leftmost character of a character expression.
LEFT	Returns the part of a character string starting at a specified number of characters from the left.
LEN	Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks.
LOWER	Returns a character expression after converting uppercase character data to lowercase.

Appendix C

- String Functions (continued):

String Function	Function Description
LTRIM	Returns a character expression after removing leading blanks.
RIGHT	Returns the part of a character string starting a specified number of *integer_expression* characters from the right.
RTRIM	Returns a character string after truncating all trailing blanks.
SUBSTRING	Returns part of a character, binary, text, or image expression.
STR	Returns character data converted from numeric data.
UPPER	Returns a character expression with lowercase character data converted to uppercase.

Appendix C

- System Functions:
  - These scalar functions perform operations on and return information about values, objects, and settings in Microsoft® SQL Server™.
  - Commonly used System functions are as follows:

System Function	Function Description
CAST & CONVERT	Explicitly converts an expression of one data type to another. CAST and CONVERT provide similar functionality.
ISDATE	Determines whether an input expression is a valid date.
ISNULL	Replaces NULL with the specified replacement value.
ISNUMERIC	Determines whether an expression is a valid numeric type.

Appendix C

- System Functions (continued):
  - Cast and Convert are some of the common functions to handle different data types. (Example, from smalldatetime to String or double to integer.)
  - Changing data types allow operations that was previously impossible.
    - For example, smalldatetime type cannot be concatenated with a string datatype. Hence Casting is used to allow smalldatetime data type to be cast/converted to a string before it is concatenated with another string.

Appendix C

- System Functions (continued):
  - Example of using CAST:

```
SELECT DISTINCT DateIssue, CAST(Dateissue AS nvarchar(20)) AS
[Casted DateIssue]
FROM Issuetran
WHERE CUSTOMERID = 9999
```

  - Output:

	DateIssue	Casted DateIssue
1	2000-11-23 00:00:00	Nov 23 2000 12:00AM
2	2000-11-24 00:00:00	Nov 24 2000 12:00AM
3	2000-11-25 00:00:00	Nov 25 2000 12:00AM
4	2000-11-26 00:00:00	Nov 26 2000 12:00AM

Appendix C

- System Functions (continued):
  - Example of using CONVERT:

```
SELECT DISTINCT DateIssue,
CONVERT(nvarchar(20),Dateissue,101) AS [Converted DateIssue]
FROM Issuetran
WHERE CUSTOMERID = 9999
```

  - Output:

	DateIssue	Converted DateIssue
1	2000-11-23 00:00:00	11/23/2000
2	2000-11-24 00:00:00	11/24/2000
3	2000-11-25 00:00:00	11/25/2000
4	2000-11-26 00:00:00	11/26/2000

Appendix D: MS SQL Server Data Types

- Data types that are used by MS SQL Server:
  - A brief description for these data types are provided here. Refer to Help Online for more details.

Data Types	Data Type Description
bigint	Integer (whole number) data from -2^63 (-9223372036854775808) through 2^63-1 (9223372036854775807).
int	Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647).
smallint	Integer data from 2^15 (-32,768) through 2^15 - 1 (32,767).
tinyint	Integer data from 0 through 255.
bit	Integer data with either a 1 or 0 value.

Appendix D

- Data types that are used by MS SQL Server (continued):

Data Types	Data Type Description
decimal	Fixed precision and scale numeric data from -10^38 +1 through 10^38 –1.
numeric	Functionally equivalent to **decimal.**
money	Monetary data values from -2^63 (-922,337,203,685,477.5808) through 2^63 - 1 (+922,337,203,685,477.5807), with accuracy to a ten-thousandth of a monetary unit.
float	Floating precision number data from -1.79E + 308 through 1.79E + 308.
real	Floating precision number data from -3.40E + 38 through 3.40E + 38.

Appendix D

- Data types that are used by MS SQL Server(continued):

Data Types	Data Type Description
datetime	Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds.
smalldatetime	Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.
char	Fixed-length non-Unicode character data with a maximum length of 8,000 characters.
varchar	Variable-length non-Unicode data with a maximum of 8,000 characters.
nchar	Fixed-length Unicode data with a maximum length of 4,000 characters.

Appendix D

- Data types that are used by MS SQL Server(continued):

Data Types	Data Type Description
nvarchar	Variable-length Unicode data with a maximum length of 4,000 characters. **sysname** is a system-supplied user-defined data type that is functionally equivalent to **nvarchar(128)** and is used to reference database object names.
binary	Fixed-length binary data with a maximum length of 8,000 bytes.
varbinary	Variable-length binary data with a maximum length of 8,000 bytes.
Image	Variable-length binary data with a maximum length of 2^31 - 1 (2,147,483,647) bytes.
timestamp	A database-wide unique number that gets updated every time a row gets updated.

End of Appendix

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

SQL QUERY EXERCISE – LIST THE RESULTS

Chia Yuen Kwan

isscyk@nus.edu.sg

Q1

- SELECT o.custId FROM orders o GROUP BY custId

table "Orders"

OrderId	CustId	OrderDate
x1	c1	1 May 05
x2	c2	2 May 05
x3	c1	14 May 05
x4	c4	16 May 05

Result : ???

Q2

- SELECT o.custId, count(*), count(o.orderId) FROM orders o GROUP BY custId

table "Orders"

OrderId	CustId	OrderDate
x1	c1	1 May 05
x2	c2	2 May 05
x3	c1	14 May 05
x4	c4	16 May 05

Result : ???

Q3

- SELECT o.custId, o.orderId FROM orders o GROUP BY custId

table "Orders"

OrderId	CustId	OrderDate
x1	c1	1 May 05
x2	c2	2 May 05
x3	c1	14 May 05
x4	c4	16 May 05

Result : ???

Q4

- SELECT o.custid, o.orderid FROM orders o GROUP BY custId, orderId

table "Orders"

OrderId	CustId	OrderDate
x1	c1	1 May 05
x2	c2	2 May 05
x3	c1	14 May 05
x4	c4	16 May 05

Result : ???

Q5

- SELECT c.custid, o.orderid FROM customers c, orders o WHERE c.custId = o.custId

table "Customers"

CustId	PhoneNo
c1	4831
c2	4832
c3	4833

table "Orders"

OrderId	CustId
x1	c1
x2	c1
x3	c2
x4	c4
x5	c5

table "Order Details"

OrderId	Product Id
x1	p1
x1	p2
x2	p3
x5	p4

Result : ???

Q6

- SELECT c.custid, c.phoneNo, o.orderid FROM customers c, orders o WHERE c.custId = o.custId

table "Customers"

CustId	PhoneNo
c1	4831
c2	4832
c3	4833

table "Orders"

OrderId	CustId
x1	c1
x2	c1
x3	c2
x4	c4
x5	c5

table "Order Details"

OrderId	ProductId
x1	p1
x1	p2
x2	p3
x5	p4

Result : ???

Q7

- SELECT c.custId, o.orderId, od.productId FROM customers c, orders o, [order details] od WHERE c.custId = o.custId AND o.orderId = od.orderId

table "Customers"

CustId	PhoneNo
c1	4831
c2	4832
c3	4833

table "Orders"

OrderId	CustId
x1	c1
x2	c1
x3	c2
x4	c4
x5	c5

table "Order Details"

OrderId	ProductId
x1	p1
x1	p2
x2	p3
x5	p4

Result :
???

Q8

- SELECT c.custid, count(*) FROM customers c, orders o WHERE c.custId = o.custId GROUP BY c.custId

table "Customers"

CustId	PhoneNo
c1	4831
c2	4832
c3	4833

table "Orders"

OrderId	CustId
x1	c1
x2	c1
x3	c2
x4	c4
x5	c5

table "Order Details"

OrderId	ProductId
x1	p1
x1	p2
x2	p3
x5	p4

Result : ???

Q9

- SELECT c.custid, count(*) FROM customers c, orders o WHERE c.custId = o.custId GROUP BY c.custId having count(*) > 1

table "Customers"

CustId	PhoneNo
c1	4831
c2	4832
c3	4833

table "Orders"

OrderId	CustId
x1	c1
x2	c1
x3	c2
x4	c4
x5	c5

table "Order Details"

OrderId	ProductId
x1	p1
x1	p2
x2	p3
x5	p4

Result : ???

Q10

- SELECT c.custid, o.orderId, count(*) FROM customers c, orders o WHERE c.custId = o.custId GROUP BY c.custId, o.orderId

table "Customers"

CustId	PhoneNo
c1	4831
c2	4832
c3	4833

table "Orders"

OrderId	CustId
x1	c1
x2	c1
x3	c2
x4	c4
x5	c5

table "Order Details"

OrderId	ProductId
x1	p1
x1	p2
x2	p3
x5	p4

Result :
???

Q11

- SELECT c.custid, o.orderId, count(*) FROM customers c, orders o WHERE c.custId = o.custId having count(*) > 1

table "Customers"

CustId	PhoneNo
c1	4831
c2	4832
c3	4833

table "Orders"

OrderId	CustId
x1	c1
x2	c1
x3	c2
x4	c4
x5	c5

table "Order Details"

OrderId	ProductId
x1	p1
x1	p2
x2	p3
x5	p4

Result : ???

Q12

- SELECT c.custid, o.orderId FROM customers c LEFT OUTER JOIN orders o ON c.custId = o.custId

table "Customers"

CustId	PhoneNo
c1	4831
c2	4832
c3	4833

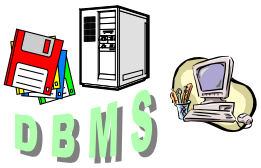
table "Orders"

OrderId	CustId
x1	c1
x2	c1
x3	c2
x4	c4
x5	c5

table "Order Details"

OrderId	ProductId
x1	p1
x1	p2
x2	p3
x5	p4

Result :
???



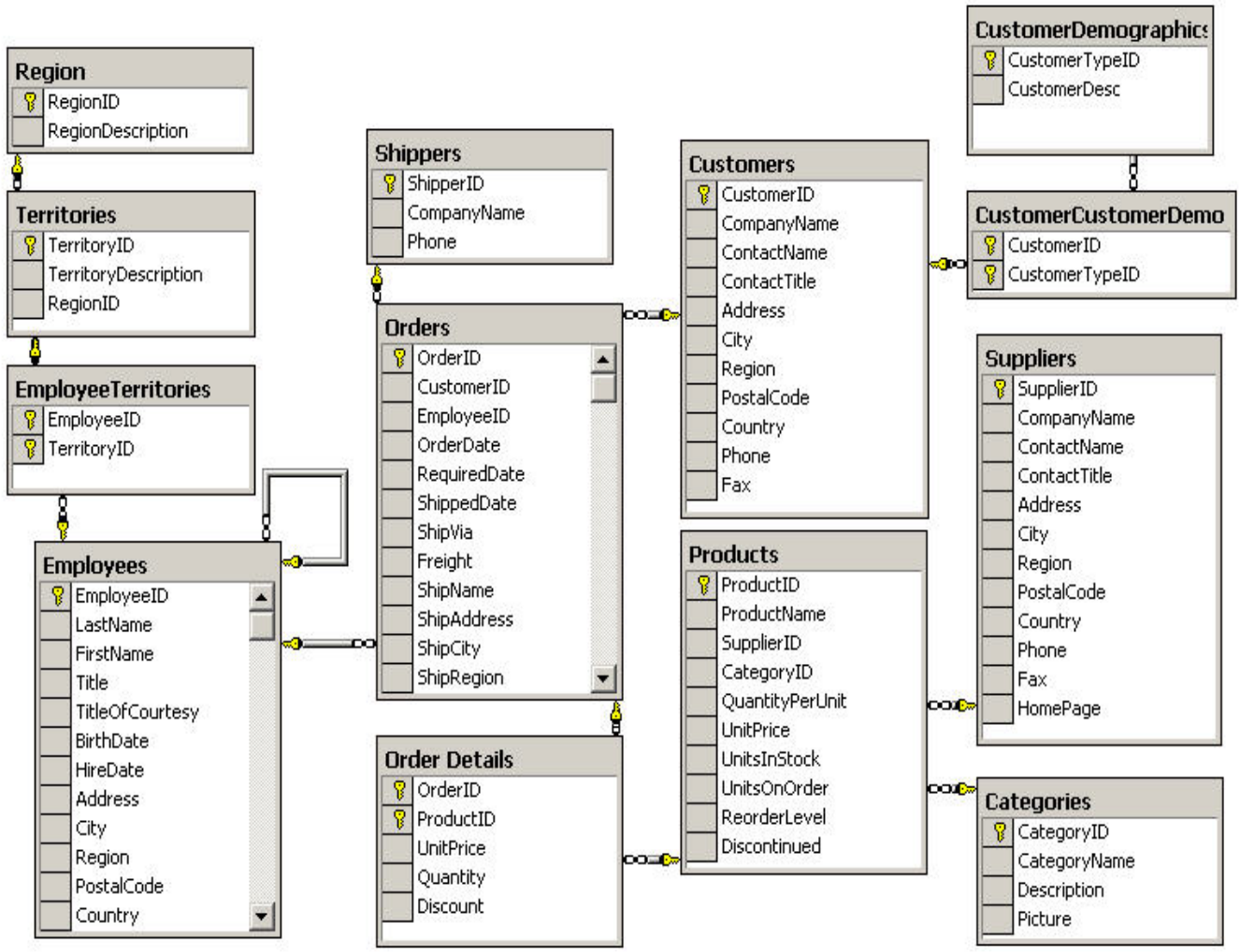
WORKSHOP
ON
SQL Data Query

Institute of Systems Science
National University of Singapore

EXERCISE

SQL FOR DATA QUERY

All Questions in this workshop involve the Northwind Database. The ERD of the Northwind database is given in the following figure.



Using Northwind Database provided by SQL Server, write SQL statements for the following data retrieval operations.

Note: You may need to refer to the schema for the exact field names while framing the queries to the following questions. While framing the question, the words 'code', 'number' or 'ID' may have been synonymously used. For instance 'Customer Number' when used may refer to a field called CustomerID. Similarly, wherever the word 'name' is used, appropriate interpretation may be needed based on the schema – for instance if 'customer name' is required to be printed, you may need to retrieve CompanyName field from the Customers Table; likewise when 'employee name' is required (without any further qualification, you may retrieve the lastname field of Employees table.

1)
   a. List all details of all Shippers that the company is dealing with.
   b. List all details of Shippers with the output presented in ascending order of shipper names.

2)
   a. List all employees - you need to display only the details of their First Name, Last Name, Title, Date of birth and their city of residence.
   b. Based on the designations (Titles) available in the employees table, can you extract the designation list?

3) Retrieve the details of all orders made on 19 May 1997

4) Retrieve details of all customers that are located in the cities of London or Madrid.

5) List all customers (Customer number and names) who are located in UK. The list should be produced in alphabetical order of customer names.

6) Provide a list of Orders (Order IDs and order dates) made by customer whose ID is 'Hanar'.

7) List the Fully Qualified Names of All Northwind Employees as a single column. Fully qualified Names should look like this: Dr. Venkat Raman OR Ms Esther Tan, where Ms is the Title of courtesy Esther is first name and Tan is last name.
   Hint: You may need to use string concatenation.

   Is it possible that this is listed in alphabetical order of the last names?

8) List all Orders (Order number and date) of the orders made by the Customer "Maison Dewey" (column: company name). Note: Maison Dewey is the name of the customer.

9) Retrieve the details of all Products' having the word "lager" in the product name.

10) Retrieve the Customer IDs and contact names of all customers who have yet to order any products.

11) Display the average product price.

12) Prepare a list of cities where customers reside in. The list should not contain any duplicate cities.

13) Retrieve the number of customers who has made orders.

14) Retrieve the company name and phone number of customers that do not have any fax number captured.

15) Retrieve the total sales made. Assume sales is equal to unit price * quantity.

16) List order ids made by customer 'Alan Out' and 'Blone Coy'

Note: Attempt the rest of the questions after the topic on Advance Queries has been covered.

17) Prepare a list of customer ids and the number of orders made by the customers.

18) Retrieve company name for the customer id 'BONAP', and also order ids made by him.

19)
   a. Retrieve the number of orders made, IDs and company names of all customers that have made more than 10 orders. The retrieved list should be display in the descending order of 'number of orders made'.

   b. Retrieve the number of orders made, IDs and company names for the customer with customer id 'BONAP'.

   c. Retrieve the number of orders made, IDs and company names of all customers that have more orders than customer with customer id 'BONAP'.

20)
   a. Prepare a Product list belonging to Beverages and Condiments categories (you may use in your SQL statement Categories Codes 1 and 2). The list should be sorted on Product code and Product Name.

   b. How would the above query change if you are not provided category codes but only the names "Beverages", "Condiments" for retrieval. Would this require a join or subquery?

21)
   a. How many employees do we have in our organization?

   b. How many employees do we have in USA?

22) Retrieve all details of Orders administered by persons who hold the designation Sales Representative and shipped by United Package.

23) Retrieve the names of all employee. For each employee list the name of his/her manager in adjacent columns.

24) Retrieve the five highest ranking discounted product. "Discounted Product" indicates products with the total largest discount (in dollars) given to customers.

25) Retrieve a list of Northwind's Customers (names) who are located in cities where there are no suppliers.

26) List all those cities that have both Northwind's Supplier and Customers.

27)
   a. Northwind proposes to create a mailing list of its business associates. The mailing list would consist of all Suppliers and Customers collectively called Business Associates here. The details that need to be captured are the Business Associates' Names, Address and Phone.

   b. Is it possible for you to add on to the same list Northwind's Shippers also. Since we do not have address of shippers, it is sufficient only phone is included leaving the address column blank.

28) Retrieve the manager's name of the employee who has handled the order 10248.

29) List the product name and product id with unit price greater than average unit product price.

To understand the answers to the following questions a sample data is provided and a sample output is provided for each question. This is only a rough sample - the northwind database should be used for framing SQL queries to obtain the answers.

Customer Table:

CustomerID	Name	Phone
C001	ABC	11223
C002	PQR	34232
C003	XYZ	78223

C004	RST	73874

Order Table:

OrderID	CustomerID	OrderDate
100	C001	06 May 97
101	C003	07 July 97
102	C001	09 July 97
103	C003	04 Sep 97
104	C004	12 Oct 97
105	C001	30 Dec 97
106	C001	05 Jan 98

Order Details Table:

OrderID	ProductID	UnitPrice	Quantity
100	P1	5.80	1200
100	P2	2.50	5000
100	P4	3.25	3000
100	P7	1.25	2000
101	P2	2.75	230
101	P6	3.40	2200
101	P2	2.50	500
102	P3	1.80	2000
102	P8	7.00	500
102	P9	4.00	750
103	P1	5.80	1000
103	P4	3.25	2030
104	P2	2.50	200
104	P4	3.25	600
104	P5	10.00	250
105	P4	3.25	3200
105	P5	10.50	490
105	P6	3.40	1500
106	P3	1.80	2000
106	P8	7.00	1000

Not part of table -These are provided for reference only

Amount	CustID
6960	C001
12500	C001
9750	C001
2500	C001
632.5	C003
7480	C003
1250	C003
3600	C001
3500	C001
3000	C001
5800	C003
6597.5	C003
500	C004
1950	C004
2500	C004
10400	C001
5145	C001
5100	C001
3600	C001
7000	C001

30) List all the orders (order number and amount) that exceed $10000 value per order. Amount means Quantity*Price.

For the above data: 101 (9362.50) and 104 ($4950) do not qualify.

31) List all the orders that exceed $10000 value per order. Your list should include order number and customer id.

For the above data: 101 and 104 do not qualify.

Since CustomerID is not part of the table, you should extract CustomerID by Joining the Order and OrderDetails table.

32) List all the orders that exceed $10000 value per order. Your list should include order number and customer id and customer name.

For the above data: 101 and 104 do not qualify.
Since CustomerID is not part of the table, you should extract CustomerID by Joining the Order and OrderDetails table. Since Customer name is not part of either of these tables, this is obtained by further joining Customer Table also.

33) List the total orders made by each customer. Your list should have customer id and Amount (Quantity * Price) for each customer.

Since CustomerID is not part of the table, you should extract CustomerID by Joining the Order and OrderDetails table. For the above data:

CustomerID	Amount
C001	73055
C003	21760
C004	4950

34) Retrieve the Average Amount of business that a northwind customer provides. The Average Business is total amount for each customer divided by the number of customer.

For the above data, the Average Amount is: (73055 + 23460 + 4950) / 3 = **33821.67**

35) List all customers (Customer id, Customer name) who have placed orders more than the average business that a northwind customer provides.

For the above data only customer C001 stands above the average of 33828.67.

36) List the total orders made by each customer. Your list should have customer id and Amount (Quantity * Price) for each customer in the year 1997. *(Use year(orderdate) to retrieve the year of the column orderdate)*

Since Order Number 106 was in Jan 98, it is not included

CustomerID	Amount
C001	62455
C003	23460
C004	4950

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

DATA MANIPULATION LANGUAGE

Chia Yuen Kwan

isscyk@nus.edu.sg

Objectives

- Upon completion of this lesson, students should be able to use SQL command syntax for:
  - Inserting, updating or deleting rows of data from the table.
  - Selectively or throughout the whole table.

INSERT

- Function of an INSERT Command:
  - Insert row(s) into a table

INSERT Command

```
INSERT INTO  table-name   (column-name,),
             | VALUES     ([constant, NULL],)
      or
             | SELECT retrieval condition
```

INSERT Statement

- **Single-Row Insert (Without Column Names)**

```
INSERT INTO ProducerWebSite
VALUES (‘Pixar', 'www.Pixar.com')
```

  - a record to be inserted into the table without specifying the columns to be inserted.
  - The values to be added represents of all the columns in that Table.

- **Single-Row Insert (With Column Names)**

```
SELECT * INTO GoodCustomers
FROM Customers
```

- Create a table GoodCustomers with records copied from Customers table

INSERT Statement

- **Single-Row Insert (With Column Names)**

```
INSERT INTO GoodCustomers
(CustomerID,CustomerName,Address)
VALUES (9000,'Grace Leong','15 Bukit Purmei
Road, Singapore 0904')
```

  - a single row of record (with 3 columns) to be inserted into the GoodCustomers Table

INSERT Statement

- **Insert using a Query (INSERT INTO… SELECT… FROM…)**

```
INSERT INTO GoodCustomers
  (CustomerID,CustomerName,Address,PhoneNumber,
   MemberCategory)
  SELECT
    CustomerID,CustomerName,Address,PhoneNumber,
    MemberCategory
  FROM Customers
  WHERE MemberCategory in ('A','B')
```

  - Allows the insertion of records from one table to another table.
  - The data types for both columns must be similar.

UPDATE

- Function of an Update Command:
  - *Changes data in one or more rows of a table*

UPDATE Command

```
UPDATE  table-name
   SET  (column-name = expression,),
   WHERE search-condition
```

UPDATE Statement

- Example:
  - **Selective Update**

```
UPDATE GoodCustomers
SET PhoneNumber = 7775588
WHERE CustomerName = 'Grace Leong'
```

  - **Update All Rows**

```
UPDATE GoodCustomers
 SET PhoneNumber = 7775588
```

  - **Update with Subquery**

```
UPDATE GoodCustomers
 SET PhoneNumber = 7775588
WHERE CustomerID in (SELECT CustomerID FROM
Customers
                        WHERE MemberCategory = 'B')
```

DELETE

- Function of a Delete command:
  - *Removes one or more rows from a table*
  - Note that DELETE is a dangerous command. Once the record (ie Row) is deleted, it cannot be undeleted.

DELETE Command

```
DELETE FROM   table-name
    {WHERE   search-condition}
```

DELETE Statement

- Example:
  - **Delete Selected Rows (From the Table Good Customers that fits the condition)**

```
DELETE FROM GoodCustomers
WHERE MemberCategory = 'B'
```

  - **Delete All Rows (from the Table)**

```
DELETE FROM GoodCustomers
```

  - **Delete Rows that satisfy certain conditions (with Subquery)**

```
DELETE FROM GoodCustomers
WHERE CustomerID in
        (SELECT CustomerID FROM Customers
                WHERE MemberCategory = 'A')
```

Summary

- DML (Data Manipulation Language)
  - Insert, Update, Delete Command

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

DATABASE DEFINITION LANGUAGE

Chia Yuen Kwan

isscyk@nus.edu.sg

Objectives

- By the end of this lesson, students should be able to use SQL command for
  - Data Definition Language to:
    - Create Tables with
      - Primary Keys
      - Foreign Keys
    - Create Indexes on Tables
    - Alter Table
    - Drop Table and Indexes
  - Define Constraints
    - Required Data Constraint
    - Validity Constraint
    - Entity Integrity
    - Referential Integrity

Topics

- **DDL (Data Definition Language)**
  - **Create / Alter / Drop**

- Defining Constraints

Data Definition Language

- DDL (Data definition language) portion of SQL
  - define and manage the structure and organization of the stored data (ie objects) and relationships among the stored data items.
  - Objects includes Tables, Views, indexes, etc

- Commands for DDL include:
  - CREATE : Create a new object in the Server system.
  - DROP : Remove the object from the system.
  - ALTER : Change the characteristic of the objects in the Server that has been present in the system.

CREATE Statement

- Function of a CREATE Statement:
  - Define new objects (database, table, index, views, etc)
    - CREATE TABLE
    - CREATE INDEX
    - CREATE VIEW

CREATE TABLE

- Example:

records created must have value in the column

```
CREATE TABLE GoodCustomers
(CustomerID          nvarchar(4)          not null,
 CustomerName        nvarchar(50)         not null,
 Address             nvarchar(65)         not null,
 PhoneNumber         nvarchar(9),
 MemberCategory      nvarchar(2)          not null,
 PRIMARY KEY (CustomerID,MemberCategory))
```

composite primary key

- Interpretation:
  - Creates a table
    - named GoodCustomers with five columns: CustomerID, CustomerName, Address, PhoneNumber and MemberCategory, with PhoneNumber can have Null values
    - having a Composite Primary Key consisting of CustomerID and MemberCategory.

CREATE TABLE

- Example (with foreign key definition) :

```
CREATE TABLE ProducerWebSite
(Producer                    varchar(50)    not null,
 WebSite                     varchar(200)   not null,

 PRIMARY KEY(Producer),
 FOREIGN KEY (Producer)      REFERENCES
                             Producers (Producer))
```

table-name (column-name)

- Interpretation:
  - Creates a table
    - named ProducerWebSite with two columns: Producer and Website,
    - having a Producer Column as the Primary Key
    - and having Producer Column of Producers Table as the Foreign Key

CREATE TABLE

- Alternative syntax (indicating constraint name) :

```
CREATE TABLE ProducerWebSite
(Producer                    varchar(50)    not null,
 WebSite                     varchar(200)   not null,

 PRIMARY KEY(Producer),
 CONSTRAINT ProducerWS_FK_1 FOREIGN KEY (Producer)
                            REFERENCES Producers
                            (Producer))
```

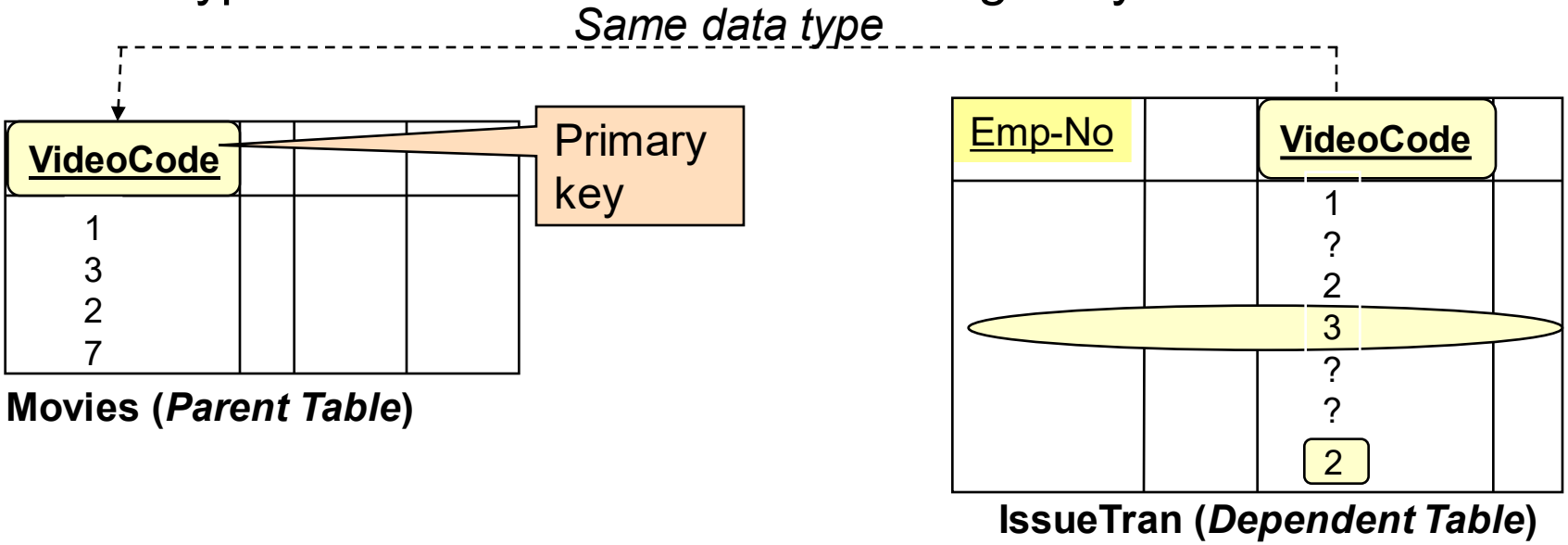
foreign key column

Constraint name

- Interpretation:
  - Creates a table
    - named ProducerWebSite with two columns: Producer and Website,
    - having a Producer Column as the Primary Key
    - and having Producer Column of Producers Table as the Foreign Key

Create Table – Foreign Key

- Associated column in the parent table
  - must be a primary key (or unique index)
  - Data type must be identical to the foreign key



- Rows can be inserted or foreign key column can be updated in the dependent table only if
  - (1) there is a corresponding primary key value in the parent table, or
  - (2) the foreign key value is set null.

CREATE INDEX

```
CREATE [UNIQUE] INDEX index-name
       ON table-name (column-name [,.....] [ASC|DESC] )
```

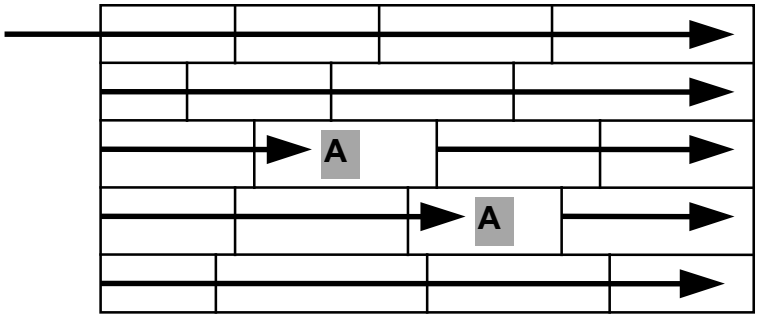
index-name:	Unique name that identifies the index
table-name:	Name of table being indexed
column-name:	Name of column(s) on which index is created. A limit is often imposed on the number of columns that can be used in a compound index

{ } / [ ] denotes optional items

Searching without Index Key

- Searching a record without index key (or primary key) involves scanning the whole table
- *E.g. Select * from Customers*

  where membercategory = 'A'



Records in Customer table are stored in random order

(if the column membercategory is not defined as primary or index key, the above query will result in table scan by the DBMS)

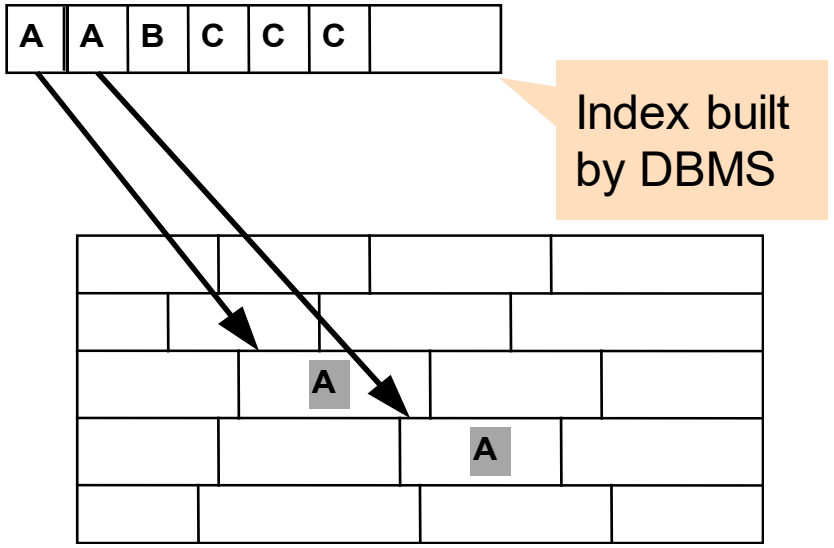
Index Key

- Index Key
  - optimise searching

An index is a listing of keys stored in order, accompanied by location to the record.

Searching a record using an index key can be speed up.

Table Read using matching index.



*Select * from Customers where membercategory = 'A' (membercategory is defined as an index key)*

CREATE INDEX

- Example:

index name

```
CREATE UNIQUE INDEX gdCust_idx ON
GoodCustomers(PhoneNumber)


CREATE UNIQUE INDEX gdCust_idx ON
       GoodCustomers(CustomerID,CustomerName)


CREATE INDEX Cust_idx ON Customers(Address)
```

Table-name (column-name)

Index Key

- An index key can be unique or non-unique
- A primary key is actually a primary unique index

Disadvantages of indexes

- Every index increases the storage space in the database

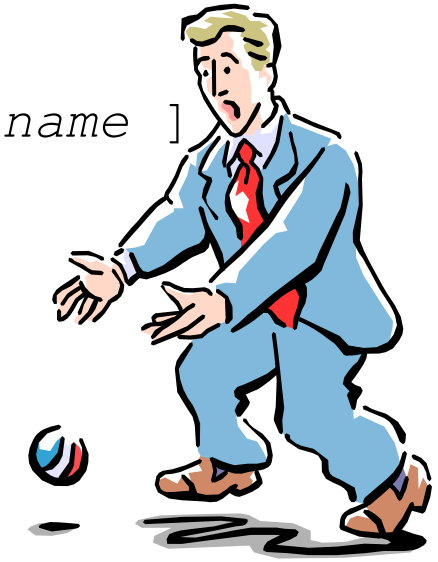
- When data are inserted, updated or deleted, the index must be updated. An index saves time in retrieval of data, but it costs time in insert, update or delete operation

DROP Statement

- Function of a DROP statement:
  - *Remove (erase) an existing object that is no longer needed*

DROP Command

```
DROP [table-name | index-name | view-name ]
```

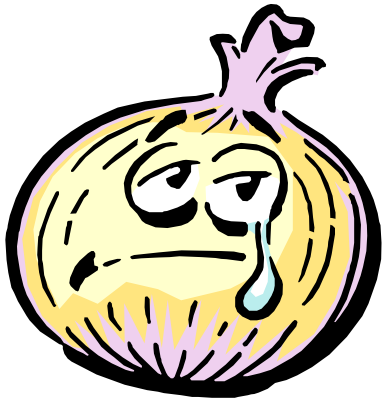


DROP Statement

- Example:

```
DROP TABLE GoodCustomers

DROP INDEX Cust_idx ON GoodCustomers
```



ALTER Statement

- Function:
  - *Change the definition of an existing table.*

```
ALTER TABLE table-name { option(s) }
    {ADD column-name  data-type  {NOT NULL} {WITH DEFAULT},
    |DROP column-name [ , ....]
    |ALTER COLUMN column-name column-type
    |ADD UNIQUE (column-list)
    |ADD PRIMARY KEY key-name (column-list)
    |ADD FOREIGN KEY (column-list) REFERENCES table-name (column-
        name)
       [ON DELETE {CASCADE | NO ACTION} ]
    |DROP PRIMARY KEY
    |DROP FOREIGN KEY constraint-name ]
    |DROP CHECK }
```

| denotes one and only one of the command is to be selected and not all

ALTER Statement

- Example:
  - Adding a Column

```
ALTER TABLE GoodCustomers
ADD CustomerPassword nvarchar(25)
```

  - Dropping a Column

```
ALTER TABLE GoodCustomers
DROP COLUMN CustomerPassword
```

  - Dropping a Primary Key

```
ALTER TABLE GoodCustomers
DROP PK_GoodCustomers
```

Primary key constraint name

ALTER Statement

- Example:
  - Adding Primary Key

```
ALTER TABLE Country ADD PRIMARY KEY (CountryCode)
```

  - Adding Unique Key

```
ALTER TABLE IssueTran ADD UNIQUE (TransactionID)
```

  - Adding Foreign Key

```
ALTER TABLE IssueTran
ADD FOREIGN KEY (CustomerId) REFERENCES Customers (CustomerId)
```

If you encounter problem in the process of adding foreign key, check whether existing data fulfills referential integrity constraints

Topics

- DDL (Data Definition Language)
  - Create / Alter / Drop

- **Defining Constraints**

SQL for Data Integrity

- Data integrity can be lost in many ways:
  - Invalid data added to data base
  - Existing data modified to a incorrect value

- SQL can be used to enforce date integrity by inserting the following types of constraints when the object is created (using DDL):
  - Required Data
  - Validity Checking
  - Entity Integrity
  - Referential Integrity

Required Data Constraints

- Required Data
  - Fields (or columns) cannot accept null value
  - Usually handled by **Not Null**
  - Eg.:

records does not accept record with no value in Producer column

```
CREATE TABLE ProducerWebSite
(Producer                    nvarchar(50)   NOT NULL,
 WebSite                     nvarchar(200)  NOT NULL
 PRIMARY KEY(Producer)
 FOREIGN KEY(Producer)       REFERENCES Producers)
```

Validity Checking Constraint

- Validity Checking
  - Columns having a particular range or format

```
CREATE TABLE StockAdjustment
(VideoCode              SmallInt                not null,
AdjustmentQty           Int,
DateAdjusted            DateTime,
WhoAdjust               nvarchar(20),
AdjustReason            nvarchar(50),
CONSTRAINT Con_VideoCode CHECK(VideoCode BETWEEN 0 AND 99999))
```

The table can only accept video code that falls within a certain range of values

Entity Integrity Constraint

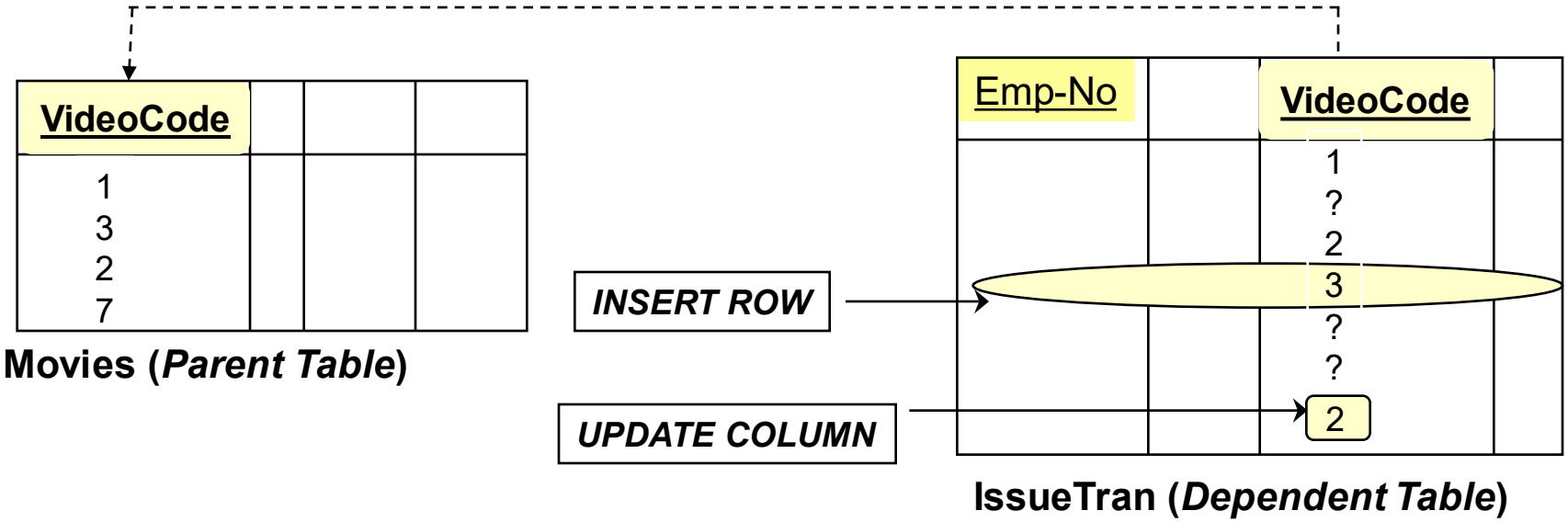
- Entity Integrity (or Entity Constraint)
  - Each row in the table to have a unique value for a particular column(s)
  - Usually implemented using a **UNIQUE** constraint or a **PRIMARY KEY** constraint
  - Eg.:

defines a unique index key

```
CREATE TABLE Producers
(Producer            nvarchar(50)   not null,
 ProducerName        nvarchar(50)   not null      UNIQUE,
 CountryCode         nvarchar(3)    not null,
 PRIMARY KEY(Producer,ProducerName),
 FOREIGN KEY(CountryCode)REFERENCES Country(CountryCode)
                                          ON DELETE CASCADE)
```

Referential Integrity Constraint

- Enforced through Foreign Key:
  - *Every non-null value in a foreign key must have a corresponding value in the primary key which it references.*



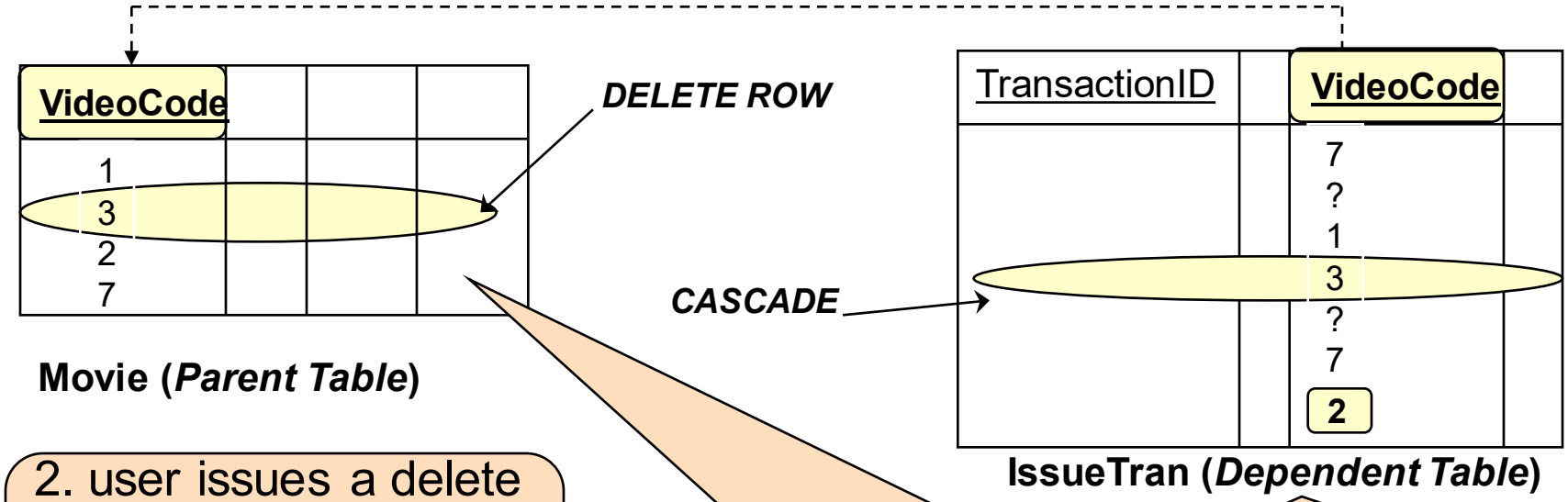
A row can be inserted or a column updated in the dependent table only if (1) there is a corresponding primary key value in the parent table, or (2) the foreign key value is set null.

Referential Integrity – Effects for Deletes Operation (1)

Database designers may explicitly declare the effect (e.g. CASCADE) if a row is deleted from the **parent** table on the dependent table.

- CASCADE deletes associated dependent rows if parent table's row is deleted

1. CASCADE effect is specified in foreign key definition



2. user issues a delete command to delete a row from the parent table

3. RDBMS delete the row in parent table and the associated row in the dependent table

SQL for Referential Integrity

- SQL data definition for defining referential integrity constraints:

  Parent table:

```
CREATE TABLE Movies
(VideoCode              smallint              not null,
… other column definitions
PRIMARY KEY (VideoCode) )
```

- Dependent table:

```
CREATE TABLE IssueTran
(TransactionID          smallint              not null,
 VideoCode              smallint              not null,
 … other column definitions
 PRIMARY KEY(TransactionID),
 FOREIGN KEY(VideoCode)  REFERENCES Movies(VideoCode)
                                 ON DELETE CASCADE)
```

SQL for Referential Integrity

- Interpretation of the commands :
  - The above example basically creates a Movie table with VideoCode being the primary key
  - The IssueTran table is created next with a foreign key, VideoCode.
  - Note that the IssueTran table references to the Movies table with a delete cascade referential integrity
  - In other words, if a row in Movies table is deleted (ie. the video code no longer exists), every transaction row in the IssueTran table having the VideoCode will be deleted.
- Defining referential integrity rules using SQL DDL is known as Declarative Referential Integrity.
- Enables enforcement at the database server level, eliminating the possibility of application errors.

Referential Integrity – Effects for Deletes Operation (2)

SQLServer:

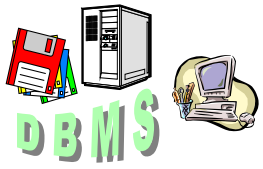
NO ACTION (default): raise error (operation on parent row not allowed) if there exists at lease one row in the referencing table

CASCADE: deletes associated dependent rows if parent table's row is deleted

The above effects can be defined for Update and Delete operations.

Summary

- Data Definition Language:
  - Create Tables with
    - Primary Keys
    - Foreign Keys
  - Create Indexes on Tables
  - Alter Table
  - Drop Table and Indexes
- Define Constraints
  - Required Data Constraint
  - Validity Constraint
  - Entity Integrity
  - Referential Integrity



SQL Programming and DBMS

WORKSHOP

ON

SQL Data Definition Language and Data Modification using DML

Institute of Systems Science
National University of Singapore

EXERCISE

DATA DEFINITION LANGUAGE & DATA MANIPULATION LANGUAGE

Using Dafesty Database provided, try the following questions.

1. Create a Table called MemberCategories with the following fields

MemberCategory	nvarchar(2)
MemberCatDescription	nvarchar(200)

   None of the fields above can be null. Set the MemberCategory as the Primary key.

2. Add the following data into the MemberCategories Table:

MemberCategory	MemberCatDescription
A	Class A Members
B	Class B Members
C	Class C Members

3. Create a Table called GoodCustomers with the following fields:

CustomerName	nvarchar(50)
Address	nvarchar(65)
PhoneNumber	nvarchar(9)
MemberCategory	nvarchar(2)

   Only Customer Name and Phone Number is mandatory.
   Since there could be two customers having the same name, make CustomerName and Phone Number as a composite primary key.

   The MemberCategory should have a referential integrity to the MemberCategories Table so that only those categories that have been listed in MemberCategories Table could be entered.

4. Insert into GoodCustomer all records form Customer table with corresponding fields except Address, which is to be left Null. Only Customers having Member Category ‘A’ or ‘B’ are good customers hence the table should be inserted only those records from the Customers table.

5. Insert into GoodCustomers the following new customer.

   CustomerName = Tracy Tan
   PhoneNumber = 736572
   MemberCategory = 'B'

6. Insert into GoodCustomers table the following information for a new customer

Since all the columns are provided you may insert the record without specifying the column names.

CustomerName = Grace Leong
Address = 15 Bukit Purmei Road, Singapore 0904'
PhoneNumber = 278865
MemberCategory = 'A'

7. Insert into GoodCustomers table the following information for a new customer
Since all the columns are provided you may insert the record without specifying the column names.

   CustomerName = Lynn Lim
   Address = 15 Bukit Purmei Road, Singapore 0904'
   PhoneNumber = 278865
   MemberCategory = 'P'

   Does the command go through – It should not since member category 'P' is not defined in MemberCategories Table. (Violation of referential integrity)

8. Change the Address of Grace Leong so that the new address is '22 Bukit Purmei Road, Singapore 0904' in GoodCustomers table.

9. Change the Member Category to 'B' for customer whose Customer ID is 5108 in GoodCustomers table.

10. Remove Grace Leong from GoodCustomers table.

11. Remove customers with 'B' member category in GoodCustomers table.

12. Add column FaxNumber (nvarchar(25)) to GoodCustomers table.

13. Alter the column Address to nvarchar(80) in GoodCustomers table.

14. Add column ICNumber (nvarchar(10)) to GoodCustomers table.

15. Create a unique index ICIndex on table GoodCustomers bases on ICNumber. Notice that the column ICNumber have no values. Can you create the unique index successfully? Why?

16. Create an index on table GoodCustomers based on FaxNumber.

17. Drop the index created on FaxNumber.

18. Remove the column FaxNumber from GoodCustomer table.

19. Delete all records from GoodCustomers.

20. Drop the table GoodCustomers.

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

USER VIEWS

Chia Yuen Kwan

isscyk@nus.edu.sg

Objective

- By the end of this lesson, students should be able to:
  - Have an understanding of a user view and its purpose.
  - Understand and appreciate:
    - the use of SQL for userview definition and manipulation
    - the limitations of user views.

What Are Userviews?

- A view is a "**Virtual Table**"

- Tables versus Views :
  - Tables
    - Store actual rows of data
    - Occupies a particular amount of storage space
  - Userviews
    - Derived or virtual tables that are visible to users
    - Do not occupies any storage space

What Are Userviews?

Base Table: Movies

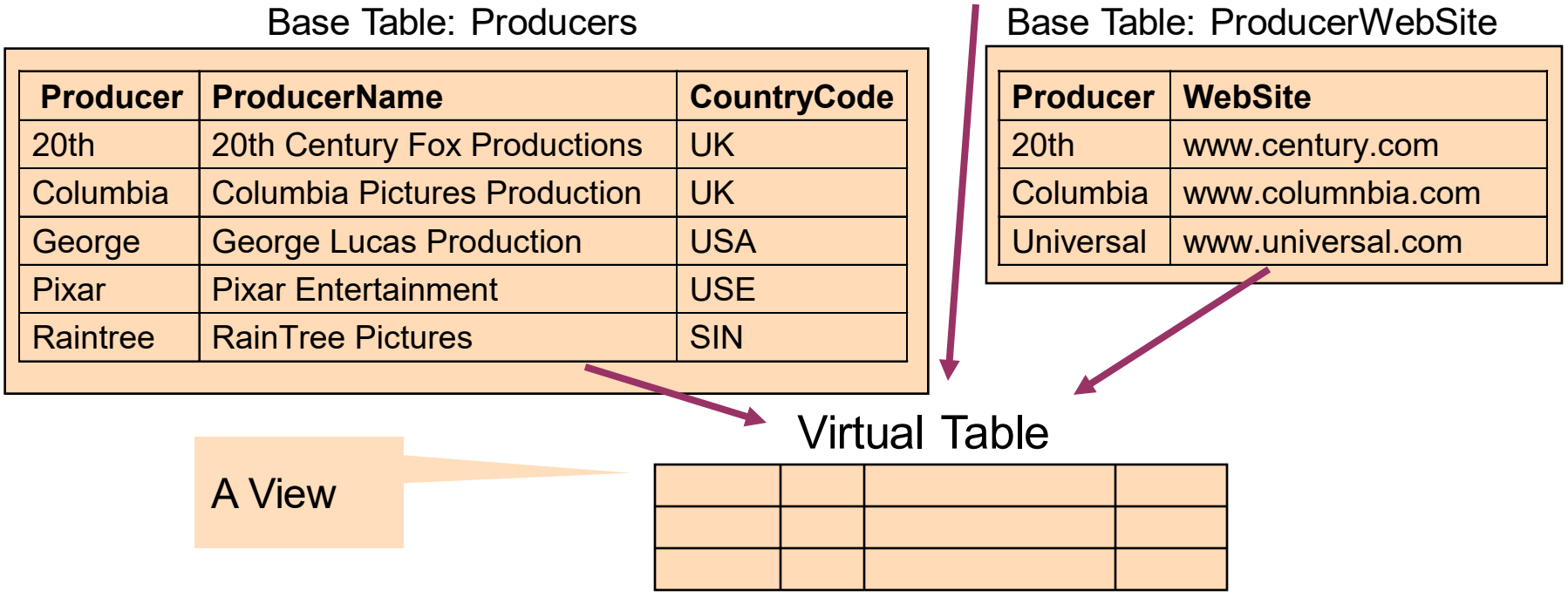
VideoCode	MovieTitle	MovieType	Rating	…	Producer
1	Star Trek 3: Search for Spock	Sci-fi	PG		Warner
2	Star Trek 4: The Voyage Home	Sci-fi	PG		Universal

Base Table: Producers

Producer	ProducerName	CountryCode
20th	20th Century Fox Productions	UK
Columbia	Columbia Pictures Production	UK
George	George Lucas Production	USA
Pixar	Pixar Entertainment	USE
Raintree	RainTree Pictures	SIN

Base Table: ProducerWebSite

Producer	WebSite
20th	www.century.com
Columbia	www.columnbia.com
Universal	www.universal.com



Characteristics of Userviews

- Like base tables, views can be queried, updated, inserted into and deleted from, with a number of restrictions

- Limitations:
  - Keys and Integrity constraints cannot be defined explicitly for views

- Benefits of User views includes:
  - Security
    - Restrict accesses to actual table
  - Query Simplicity
    - Turning multiple table queries to single table queries against views, by drawing data from several tables.
      - Building up SELECT statements in several steps.

SQL for Userviews

- Create View Command :

```
CREATE VIEW view-name
(column-name,… )
AS  query
```

  - Example:

```
CREATE VIEW Nov20TranView AS
SELECT TransactionID,CustomerID,VideoCode FROM
IssueTran
WHERE DateIssue = '20 Nov 2000'
```

  - To query the userview:

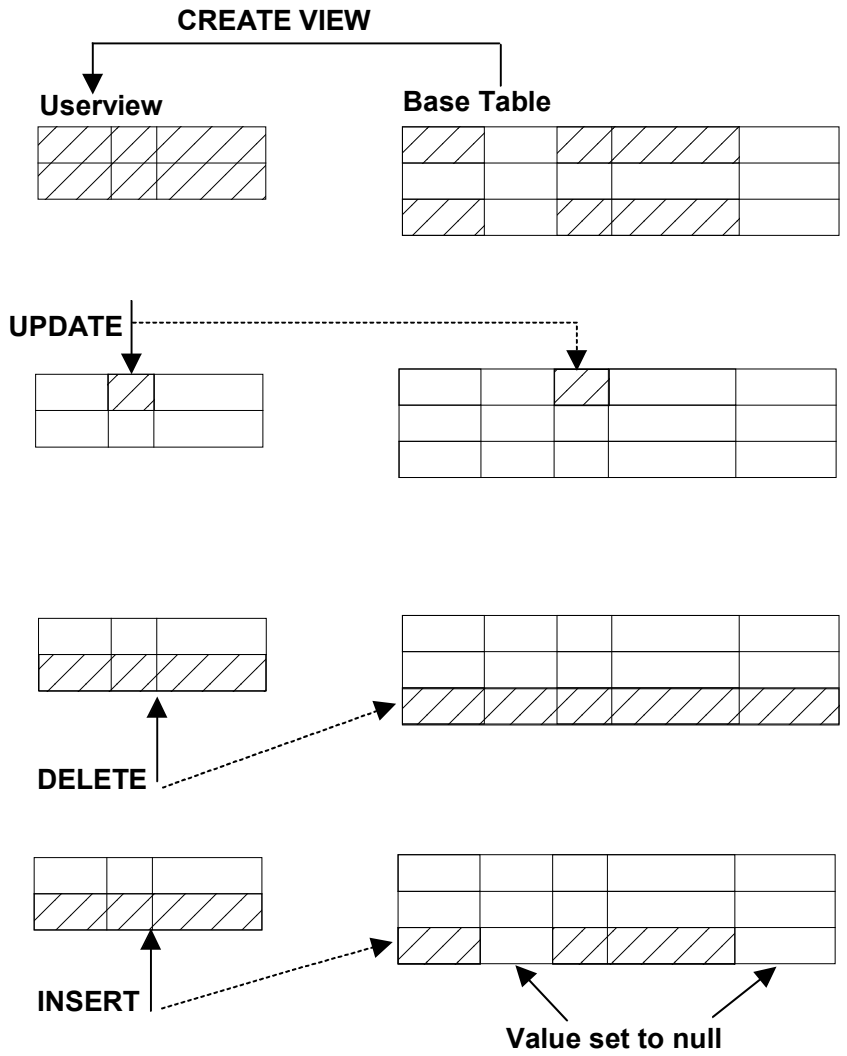
```
SELECT * FROM Nov20TranView
WHERE VideoCode = 55
```

the following query will be translated (by DBMS):

```
SELECT * FROM IssueTran
WHERE VideoCode = 55 AND DateIssue = '20 Nov 2000'
```

Insert / Update / Delete Views

- Update operation perform on the view will actually affect the base table.
- Updating a view is much simpler if the view is created from one base table (ie simple view)
- For inserting into view, note that the base tables' column that was not used (ie selected) by the view must either accept nulls or default values in order for the views to be updatable



Updating & Inserting Userview

- Example:

```
CREATE VIEW GoodCustomer
    AS SELECT CustomerID, CustomerName,
              MemberCategory,CountryCode
    FROM Customers
    WHERE MemberCategory IN ('A','B')
```

- Note that the following update and insert statement are such that MemberCategory is not 'A' or 'B'.

Are they allowed?

```
UPDATE GoodCustomer
SET MemberCategory = 'C'
WHERE CustomerID = 1000
```

```
INSERT INTO GoodCustomer (CustomerID,
                          MemberCategory, CountryCode)
VALUES ( '5000','C', 'USA' )
```

Updating & Inserting Userview

- For processing to be checked (to satisfy the view-defining condition) to limit what was inserted or updated into views
  - Include **WITH CHECK OPTION** is included in the view definition

- The new view for good customers (using WITH CHECK OPTION) is as follows:

```
CREATE VIEW GoodCustomer
AS SELECT CustomerID, CustomerName,
MemberCategory,                    CountryCode
FROM Customers
WHERE MemberCategory IN ('A','B')
WITH CHECK OPTION
```

Restrictions: Updating Userviews

- Issue on View Updatability
  - Views constructed from complicated table queries may not be updatable
- Commercial SQL Products
  - Different rules and implementation on this aspect
- ANSI/ISO Standards :
  - Views are updatable if the derivation of such views meet the following criteria:
    - Single source table
    - Simple column reference (no column functions, no virtual columns)
    - The WHERE clause does not include a subquery
    - DISTINCT must not be specified
    - No GROUP BY clause
    - No HAVING clause

Exercises

- Using Northwind database, try the following questions:
- Exercise Userview 1
  - Create a View Customer1998 containing Customer IDs and names, Product IDs and names for customers who have made orders on the year 1998.
- Exercise Userview 2
  - Using the View Customer1998, retrieve the Customer name, Product name and supplier names for the Customers who have made orders on the year 1998 according to Customer Name.
- Exercise Userview 3
  - Retrieve the Customer name and the number of products ordered by them in the year 1998.

Exercises

- Exercise Userview 4

  a) Create an Userview to represent total business made by each customer. The userview includes two columns:
     - The sum of product's unit price multiplied by quantity ordered by the customer
     - Customer id

  b) Using the userview created, retrieve the Average Amount of business that a northwind customer provides. The Average Business is total amount for each customer divided by the number of customer.

- Exercise Userview 5

  a) Create an Userview to represent employee details with employee id, last name and title

Summary

- Usage of user view
- Defining user view
- Restrictions on updating user view

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

STORED PROCEDURES

Chia Yuen Kwan

isscyk@nus.edu.sg

Objectives

- Upon completion of this lesson, students should be able to:
  - Understand the Merits and Demerits of stored procedures.
  - Create and execute stored procedures

Stored Procedure

- Stored procedure
  - a named collection of SQL statements and procedural logic that is **compiled** and **stored** in the server database.
- Advantages
  - Efficiency
    - Subsequent execution is fast since SQL statements are precompiled before execution
- Disadvantages
  - Non-standard
    - Different vendors implement differently
    - Not portable across vendor platforms

Creating Stored Procedure - SQLServer

- Syntax (simplified):

```
CREATE PROCEDURE procedure_name
AS
BEGIN
  ....
  SQL_statements
  ....
END
```

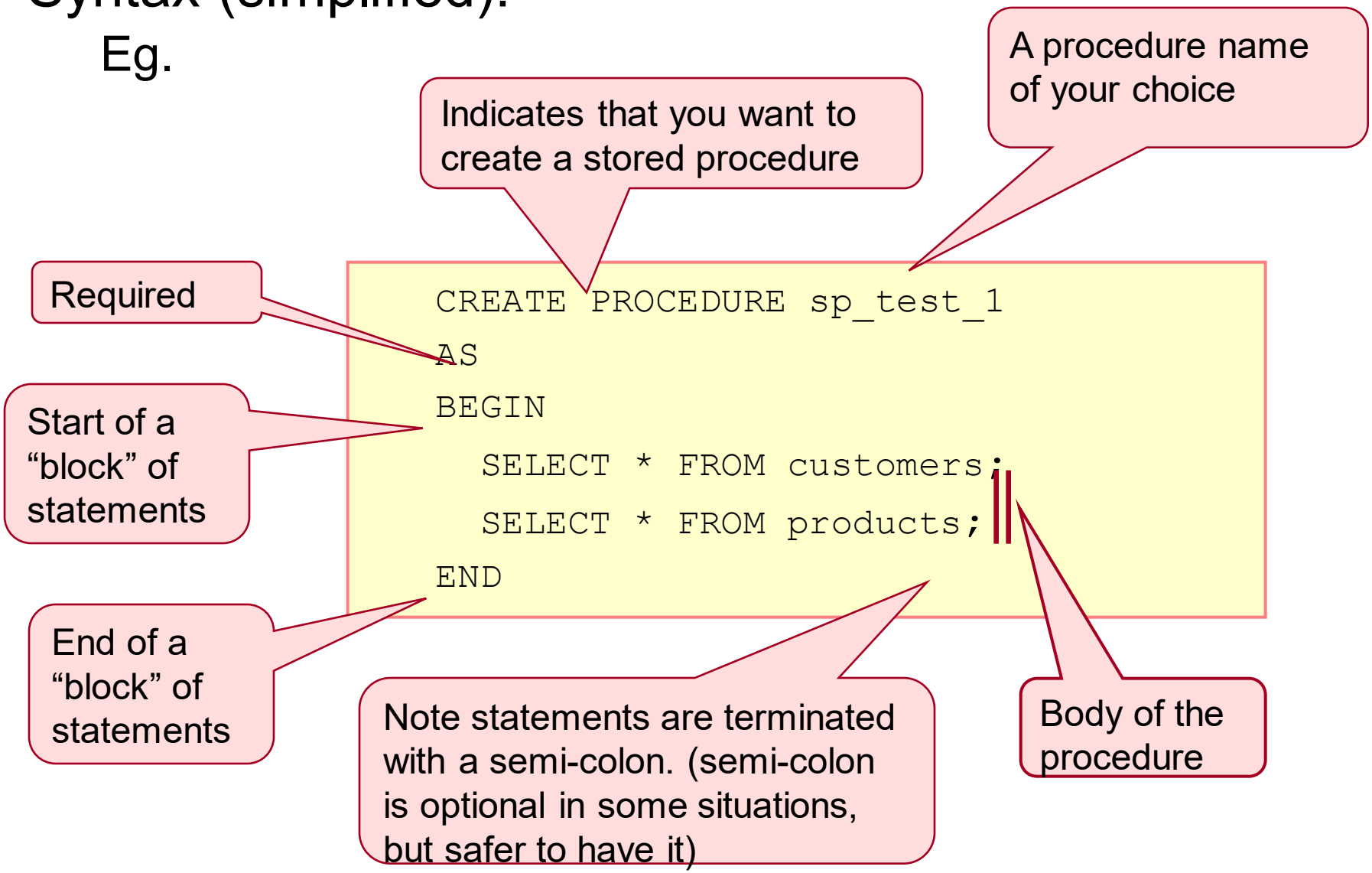
  - There can be multiple SQL statements

Note: For full Syntax refer MSDN or SQL Books Online

Creating Stored Procedure - SQLServer

- Syntax (simplified):

  Eg.



Procedures with Parameters - SQLServer

- Declaring Stored Procedures that takes arguments:
  - *Stored Procedure*

```
CREATE PROCEDURE MyProcedure (@var1 DataType,  @var2
   DataType)
AS
BEGIN
   …
   …
END
```

  - *Calling Program*

```
Exec MyProcedure  val1, val2
```

OR

```
Exec MyProcedure @var1= val1, @var2 = val2
```

Procedures with Parameters - SQLServer

- Declaring Stored Procedures that takes arguments:
  - *Stored Procedure*

```
CREATE PROCEDURE MyProcedure (@var1 CHAR(2), @var2 INTEGER)
AS
    SELECT *
    FROM Movies
    WHERE Rating= @var1 and
          TotalStock > @var2;
```

  - *Executing stored procedure*

```
Exec MyProcedure  'PG', 1
```

  - *Or*

```
Exec MyProcedure  @var1='PG', @var2=1
```

Workshop (1)

- Exercise Stored Procedure 1
  - Write a stored procedure that would list all members who belong to ‘A’ category.
  - Write statements to call this procedure.

- Exercise Stored Procedure 2
  - Write stored procedure that would take as parameter (argument) a member category and list all the members belonging to that category.
  - Write calling Statements to call the procedure and test the stored procedure for various inputs.
    - What is the output if the argument is ‘B’?
    - What is the output if the argument is ‘Z’?

Workshop (2)

- Exercise Stored Procedure 3
  - Write a stored procedure that update customer address based on the customer name.
  - Write statements to test this procedure.

- Exercise Stored Procedure 4
  - Write stored procedure that insert customer record, with input parameters as customer id, customer name, member category, address and postal code.
  - Write statements to test this procedure.

Workshop (3)

- Exercise Stored Procedure 5
  - Write a stored procedure that delete a customer record based on the customer id.
  - Write statements to test this procedure.

- Exercise Stored Procedure 6
  - Write a stored procedure to retrieve all video code rented by a customer, with customer name as the input parameter.
  - Write statements to test this procedure.

Summary

- Stored Procedures:
  - Collection of compiled SQL statements
- Advantages:
  - Efficiency
- Disadvantages:
  - Different implementation across different platforms

Appendix - User Defined Functions

- What is a User-Defined Function?
  - User defined Functions are server side programs create at the database.
  - They are similar to Stored Procedures as far as the programming constructs are concerned.
  - However, unlike the stored procedures, functions return a single value.
    - Remember stored procedures do not return a value
    - In stored procedures, parameters may sometimes pass back value.
  - While stored procedure is like a void method in C# or Java, Functions are similar to methods that have a definite return value and type
- Why do we need Functions? Will Stored Procedures not suffice?
  - In most situations Stored procedures would suffice
  - Functions become useful since they can be directly used in Expressions or Select Statements, while the stored procedures cannot.
    - Remember how the IsNull, Upper, or Round Library Functions can be used in Statements.

Creating Functions

- User Defined Functions were introduced in **SQL Server**
- Syntax for creating User Defined Functions

```
CREATE FUNCTION Name_Of_Function (PARAMETER LIST)
RETURNS (return_type_spec) AS
BEGIN
   (FUNCTION BODY - SQL Statements)
   RETURN Rtn_Value
END
```

  - Name_Of_Function is any valid identifier name
  - Return_Type_Spec is the data type like varchar(n), int, float etc.
  - Rtn_Value is the value that this function returns.

Using Functions (SQLServer)

- User defined function can be used in expressions or select statements, the same way as the library functions.
- Examples of use:
  - Assume that FN1 is a function that takes an integer argument an returns an string
  - FN1 will return “Even” if the argument is an even number, “Odd” if it is an odd number.

```
DECLARE @var1 VARCHAR(10);
 SET @var1 = FN1(5) + ' Number ';

PRINT dbo.FN1(28);

SELECT MovieTitle, dbo.FN1(NumberRented) FROM Movies;
```

Example of Function (SQLServer)

- Creating a function:
  - The following function takes as argument a number (intended to be stock quantity) and returns a string which would the number itself if the stock is positive, the word “Out of Stock” if the argument is zero and the phrase “Error in Qty ” if the argument is negative.

```
CREATE Function FNStock (@var1 int)
RETURNS varchar(15)
AS
BEGIN
DECLARE @TmpVar varchar(15);
  if (@var1 > 0)
    SET @TmpVar = str(@var1);
  else if (@var1 = 0)
    SET @TmpVar = 'Out of Stock';
  else
    SET @TmpVar = 'Error in Qty'
RETURN @TmpVar;
END
```

Example of Function (SQLServer)

- Calling the Function:

```
PRINT dbo.FNStock(5);
PRINT dbo.FNStock(-2);
PRINT dbo.FNStock(0);
```

- Output:

```
5
Error in Qty
Out of Stock
```

Example of Function (SQLServer)

- Using the function in a Select Statement:
  - FNStock is used with the field QtyInSock field of Movies Table as it's argument

```
SELECT VideoCode, MovieTitle, dbo.FNStock(NumberRented) AS
Stock
FROM Movies
```

- Output:

VideCode	MovieTitle	Stock
58	Warlock	3
132	Lawn Mower Man	1
133	Sleepwalkers	Out of Stock
172	Warlock: The Armageddon	Error in Qty

This is Zero

This is -2

Common Attributes

Northwind Database

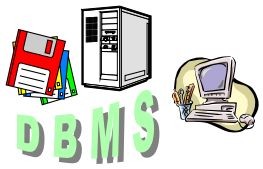
Parent Table (PK)	Dependant (FK)
Employees (EmployeeID)	Orders (EmployeeID)
Employees (EmployeeID)	Employee (ReportsTo)
Shippers(ShipperID)	Orders(ShipVia)
Orders(OrderID)	[Order Details](OrderID)
Products(ProductID)	[Order Details](ProductID)
Customers(CustomerID)	Orders(CustomerID)
Categories (CategoryID)	Products (CategoryID)

The above exhibit 1-to-many relationships

Dafesty Video Rental

Parent Table (PK)	Dependant (FK)
Customers (CustomerID)	IssueTrans (CustomerID)
Movies (VideoCode)	IssueTrans (VideoCode)
CountryCode (CountryCode)	Customers (CountryCode)
Movies (VideoCode)	StockAdjustments (VideoCode) *
Producers (Producer)	Movie (Producer)
Producers (Producer)	ProducerWebSite (Producer) *
Country (CountryCode)	Producers (CountryCode)

The above exhibit 1-to-many or 1-1 (*) relationships



SQL Programming and DBMS

TRANSACT SQL (DML) REFERENCE

Institute of Systems Science
National University of Singapore

SQL Command Syntax

Symbol	Meaning
[ ]	Items enclosed in brackets are optional
{ }	One item must be selected from the items enclosed in braces
[ONE|TWO]	A vertical bar in brackets implies ONE or TWO (or neither) may be selected
{ONE|TWO}	A vertical bar in braces implies ONE or TWO must be selected
[<u>ONE</u>|TWO]	A underscore in brackets implies the option underscored become the default if neither is selected
{<u>ONE</u>|TWO}	A underscore in braces implies the option underscored become the default if neither is selected
ABC	Uppercase characters indicate a command name, keyword or reserved word. It must be typed as shown in the syntax
abc	Lower case characters indicate a substitute. It is to be specified by the user.
…	The term preceding the ellipsis may be repeated

SQL Reserved Words

SQL-92 standard defines a list of reserved keywords as follows:

ABSOLUTE	EXEC	OVERLAPS
ACTION	EXECUTE	PAD
ADA	EXISTS	PARTIAL
ADD	EXTERNAL	PASCAL
ALL	EXTRACT	POSITION
ALLOCATE	FALSE	PRECISION
ALTER	FETCH	PREPARE
AND	FIRST	PRESERVE
ANY	FLOAT	PRIMARY
ARE	FOR	PRIOR
AS	FOREIGN	PRIVILEGES
ASC	FORTRAN	PROCEDURE
ASSERTION	FOUND	PUBLIC
AT	FROM	READ
AUTHORIZATION	FULL	REAL
AVG	GET	REFERENCES
BEGIN	GLOBAL	RELATIVE
BETWEEN	GO	RESTRICT
BIT	GOTO	REVOKE
BIT_LENGTH	GRANT	RIGHT
BOTH	GROUP	ROLLBACK
BY	HAVING	ROWS
CASCADE	HOUR	SCHEMA
CASCADED	IDENTITY	SCROLL
CASE	IMMEDIATE	SECOND
CAST	IN	SECTION
CATALOG	INCLUDE	SELECT
CHAR	INDEX	SESSION
CHAR_LENGTH	INDICATOR	SESSION_USER
CHARACTER	INITIALLY	SET
CHARACTER_LENGTH	INNER	SIZE
CHECK	INPUT	SMALLINT
CLOSE	INSENSITIVE	SOME
COALESCE	INSERT	SPACE
COLLATE	INT	SQL
COLLATION	INTEGER	SQLCA

COLUMN	INTERSECT	SQLCODE
COMMIT	INTERVAL	SQLERROR
CONNECT	INTO	SQLSTATE
CONNECTION	IS	SQLWARNING
CONSTRAINT	ISOLATION	SUBSTRING
CONSTRAINTS	JOIN	SUM
CONTINUE	KEY	SYSTEM_USER
CONVERT	LANGUAGE	TABLE
CORRESPONDING	LAST	TEMPORARY
COUNT	LEADING	THEN
CREATE	LEFT	TIME
CROSS	LEVEL	TIMESTAMP
CURRENT	LIKE	TIMEZONE_HOUR
CURRENT_DATE	LOCAL	TIMEZONE_MINUTE
CURRENT_TIME	LOWER	TO
CURRENT_TIMESTAMP	MATCH	TRAILING
CURRENT_USER	MAX	TRANSACTION
CURSOR	MIN	TRANSLATE
DATE	MINUTE	TRANSLATION
DAY	MODULE	TRIM
DEALLOCATE	MONTH	TRUE
DEC	NAMES	UNION
DECIMAL	NATIONAL	UNIQUE
DECLARE	NATURAL	UNKNOWN
DEFAULT	NCHAR	UPDATE
DEFERRABLE	NEXT	UPPER
DEFERRED	NO	USAGE
DELETE	NONE	USER
DESC	NOT	USING
DESCRIBE	NULL	VALUE
DESCRIPTOR	NULLIF	VALUES
DIAGNOSTICS	NUMERIC	VARCHAR
DISCONNECT	OCTET_LENGTH	VARYING
DISTINCT	OF	VIEW
DOMAIN	ON	WHEN
DOUBLE	ONLY	WHENEVER
DROP	OPEN	WHERE
ELSE	OPTION	WITH
END	OR	WORK
END-EXEC	ORDER	WRITE

ESCAPE	OUTER	YEAR
EXCEPT	OUTPUT	ZONE
EXCEPTION		

Data Manipulation Language (DML)

Consists of SQL statements used to retrieve, insert, update and delete records from database tables. DML statements can be rolled back or recovered.

- SELECT
- INSERT
- UPDATE
- DELETE

SQL SELECT COMMAND

SELECT is a SQL command that queries (retrieves) data from one or more tables..

```
SELECT [ALL | DISTINCT]  {select-list | *}
        FROM [creator.]table-name
        [WHERE search-condition]
        [GROUP BY column-name [HAVING search-condition]]
        [ORDER BY colspec [ASC | DESC] [, colspec [ASC | DESC]] … ]
```

ALL:

Specifies all values (including duplicates) are to be selected. If ALL or DISTINCT is not specified, the default is ALL.

DISTINCT:

Specifies only distinct values (no duplicates) are to be selected

Select-list:

- One or more items (columns), separated by commas, are to be selected.
- Arithmetic operators like +, -, * and / may be used to connect numeric data types.
- Built-in functions like AVG, SUM, MIN, MAX and COUNT may be specified. (see page 6)

*:

All columns in the tables are to be selected.

creator:

Is the user-id of the owner of the table.

table-name:

Is the name of the table. The name must begin with a letter, number, $, # or @. The maximum length ranges from 16 –32 characters (dependent on the RDBMS used). These characters can be upper and lowercase letters, numbers, $, #, @ and underscores (_).

WHERE search-condition:

One or more conditions to apply in selecting data. (see Page 7)

GROUP BY column-name:

Allows grouping of rows of common matching values in a column and return only one resulting row for each group. Only used together with built-in functions.

SQL SELECT COMMAND

HAVING search-condition:
One or more conditions to apply to groups (must be use with GROUP BY). Normally built-in functions are specified in the search condition.

ORDER BY colspec:
Sorts the resulting rows by the column(s) specified.

ASC | DESC:
Indicates either ascending or descending order for sorting retrieved data.

BUILT-IN FUNCTIONS

Five commonly used built-in functions that can be specified to data retrieved from a table:

- AVG provides the average value:
  - AVG(item)
  - AVG(DISTINCT item)
- SUM provides the total value:
  - SUM(item)
  - SUM(DISTINCT item)
- MIN provides the minimum value:
  - MIN(item)
- MAX provides the maximum value:
  - MAX(item)
- COUNT provides the number of values:
  - COUNT(*)
  - COUNT(item)
  - COUNT(DISTINCT item)

SQL SELECT COMMAND

Search Conditions:

These are one or more conditions to apply in selecting, updating or deleting data. The following types of search conditions are supported:

- Logical operators or Comparisons :
  - = Equal to
  - != or <> Not equal to
  - < Less than to
  - <= Less than or equal to
  - > Greater than to
  - >= Greater than or equal to
- ANY operator :
  - ANY (selected list of items)
- Arithmetic operators :
  - + (add)
  - - (subtract)
  - * (multiply)
  - / (divide)
  - % (modulo)
- Connecting operator for multiple conditions :
  - AND (Logical AND)
  - OR (logical OR)
  - NOT (logical NOT)
- Parenthesis () can be used to group conditions
  - = Equal to
- BETWEEN operator
  - *Expression:* [NOT] BETWEEN *expression-2* AND *expression-3*
  - Example: Weight BETWEEN 50 AND 60

SQL SELECT COMMAND

- Set Membership ([NOT] IN)
  - *Expression:* [NOT] IN (list of items)
  - Example 1: WHERE SerialNo NOT IN (“S1”, “S2”, “S3’)
  - Example 2: WHERE StudentNo IN (*Select statement*)

- Null Value (IS [NOT] NULL)
  - *Expression:* column-name IS [NOT] NULL
  - Example: WHERE ClassID IS NOT NULL

- Pattern Matching ([NOT] LIKE)
  - *Expression:* column-name [NOT] LIKE quoted-string (see below)
  - Example1: WHERE CompanyName LIKE “NATION%”
  - Example2: WHERE MovieTitle LIKE “STAR_ _ _ _”

Quoted String:

A quoted string is used in conjunction with the LIKE operator in a search condition to perform special selection of rows.

Characteristics of quoted strings :

- It may contain any character string, with special meanings reserved for the character “_” (underscore) and “%” (percent).
- The “_” character represents “any single character”
- The “%” represents “any string of zero or more characters”.

SQL SELECT COMMAND

Examples of SELECT command

1. SELECT ALL PartNo FROM Quotations

2. SELECT DISTINCT PartNo FROM Quotations

3. SELECT SuppNo, PartNo FROM Quotations

4. SELECT SuppNo, PartNo, Price*QtyOnOrder FROM Quotations

5. SELECT COUNT(*), MIN(Price), MAX(Price), AVG(Price),
   SUM(QtyOnOrder) FROM Quotations
   WHERE PartNo = 221

6. SELECT * FROM Quotations

7. SELECT SuppNo, Name, Address FROM Suppliers
   WHERE SuppNo = ANY (SELECT SuppNo FROM Quotations
   WHERE PartNo = 222)

8. SELECT * FROM Quotations
   WHERE DeliveryTime*2 > 28

9. SELECT * FROM Quotations
   WHERE PartNo = 221 AND QtyOnOrder > 0

10. SELECT * FROM Quotations
    WHERE SuppNo = 54 OR SuppNo = 64

11. SELECT * FROM Quotations
    WHERE SuppNo = 54 OR SuppNo = 64
    AND NOT PartNo = 209

12. SELECT * FROM Quotations
    WHERE SuppNo = 64 OR
    (SuppNo = 54 AND NOT PartNo = 209)

13. SELECT * FROM Quotations
    WHERE (SuppNo = 54 OR SuppNo = 64)
    AND NOT PartNo = 209

14. SELECT * FROM Quotations
    WHERE Price BETWEEN 1.00 AND 10.00

15. SELECT * FROM Inventory
    WHERE PartDesc IN (‘NUT’, ‘BOLT’, ‘WASHER’)

16. SELECT * FROM Inventory
    WHERE PartNo IS NULL

17. SELECT * FROM Inventory
    WHERE PartDesc R LIKE ‘B_ _ _’

18. SELECT Name, Address FROM SUPPLIERS
    WHERE Address LIKE ‘%NY%

SQL SELECT COMMAND

Joining Tables

Any SQL SELECT command that retrieves data from **more than one table** is called a join.

Characteristics of joins:

- A maximum of number of tables that can be joined varies with different vendors' implementations.
- A join can be performed on the same table.
- All tables that participate in a join are specified in the FROM clause (separated by commas) of the SELECT command.
- All items to be retrieved (from any table) are specified in the select-list of the SELECT command
- When the tables to be joined have a common column-name, the column-name must be prefixed with the table-name in the select-list.

Joining Conditions

Any search condition involving some relationship between tables in the FROM clause is called a join condition.

Characteristics of join condition:

- If a join condition is specified, only rows that satisfied the join condition is selected.
- If no join condition is specified, all possible combinations of rows from the tables are selected.
- When the column-names specified in a join condition have a common column-name, each of the column-name must be prefixed with the table-name of that column.

SQL SELECT COMMAND

Examples of SELECT command for joining 2 or more tables

1.
```
SELECT Inventory.PartNo, PartDesc, Price
FROM Inventory, Quotations
```

ALTERNATIVE SYNTAX:

```
SELECT Inventory.PartNo, PartDesc, Price
FROM Inventory,
JOIN Quotations
```

2.
```
SELECT Inventory.PartNo, PartDesc, Price
FROM Inventory, Quotations
WHERE Inventory.PartNo = Quotation. PartNo
```

ALTERNATIVE SYNTAX

```
SELECT Inventory.PartNo, PartDesc, Price
FROM Inventory
JOIN Quotations
ON Inventory.PartNo = Quotation.PartNo
```

3.
```
SELECT Inventory.PartNo, PartDesc, Price
FROM Inventory, Quotations
WHERE Inventory.PartNo = Quotation.PartNo
ORDER BY Inventory.PartNo
```

ALTERNATIVE SYNTAX

```
SELECT Inventory.PartNo, PartDesc, Price
FROM Inventory
JOIN Quotations
ON Inventory.PartNo = Quotation.PartNo
ORDER BY Inventory.PartNo
```

4.
```
SELECT * FROM Inventory, Quotations
WHERE Inventory.PartNo = Quotations.PartNo
ORDER BY Inventory.PartNo
```

ALTERNATIVE SYNTAX

```
SELECT * FROM Inventory
JOIN Quotations
ON Inventory.PartNo = Quotations.PartNo
ORDER BY Inventory.PartNo
```

SQL SELECT COMMAND

SUBQUERIES

In a search condition of a SELECT, UPDATE or DELETE command, another SELECT command may be specified. This SELECT command is called a subquery.

Characteristics of subqueries:

- The resulting list of values from the subquery is directly substituted into the outer-level query.
- The subquery must be enclosed in parenthesis ().
- A subquery may include another subquery.
- A subquery may include a join
- A query may consist of multiple subqueries.

Examples of Subqueries

1.
```
SELECT * FROM Inventory
WHERE PartNo IN (SELECT Quotations,PartNo, SuppNo
                 FROM Quotations WHERE SuppNo = 53)
```

2.
```
SELECT * FROM Inventory
WHERE PartNo IN (SELECT PartNo FROM Quotations
                 WHERE SuppNo IN (SELECT SuppNo FROM Suppliers
                                  WHERE Name = 'Atlantis Co Ltd'))
ORDER BY PartNo
```

3.
```
SELECT SuppNo, PartNo, Price FROM Quotations
    WHERE Price < (SELECT AVG(Price) FROM Quotations
                   WHERE PartNo = 207
```

SQL INSERT COMMAND

INSERT is a SQL command that inserts/places one or more new rows into a table.

Format 1

```
INSERT INTO [creator.] {table-name} [ (column-names) ]
        VALUES (data-items)
```

Format 2

```
INSERT INTO [creator.] {table-name} [ (column-names) ]
        select-statement
```

creator:
the userid of the owner of the table.

table-name :
the name of the table.

column-names:
one or more column-names, separated by commas, that identify the column(s) into which data-items that follow are to be inserted.

data-items :
one or more values, separated by commas, to be inserted into a new row for the column(s) specified in the column-names.

select-statement:
a subquery (SELECT command) that selects the data to be inserted. Typically used for multiple row inserts.

Examples:

1.
```
INSERT INTO Inventory
        VALUES (207, 'Gear', 75)
```

2.
```
INSERT INTO Inventory (PartDesc, PartNo, QtyOnHand)
        VALUES ('Screw', 208, 25)
```

3.
```
INSERT INTO Production
        SELECT PartNo, QtyOnHand FROM Inventory
        WHERE PartDesc = 'Bolt'
```

SQL UPDATE COMMAND

UPDATE is a SQL command that changes the values of one or more values in one or more rows of a table. Also has a special format (UPDATE WHERE CURRENT OF) for embedded SQL.

```
UPDATE [creator.] table-name
        SET column-name-1 = expression-1
                [, column-name-2 = expression-2] …
        [WHERE search-condition]
```

creator:
the userid of the owner of the table.

table-name :
the name of the table.

column-names:
the name of column-name to be updated.

expression :
- the new value to be placed in the specified column.
- an expression may contain constant, NULL, column-name and arithmetic operators like +, -, * and /.

WHERE search-condition:
one or more conditions to apply in selecting data.

Examples:

1.
```
UPDATE Inventory
        SET PartDesc = 'Inactive', QtyOnHand = NULL
        WHERE PartNo = 295
```

2.
```
UPDATE Quotations
        SET QtyOnOrder = 0
        WHERE PartNo = 222 AND SuppNo = 61
```

SQL DELETE COMMAND

DELETE is a SQL command that removes one or more rows from a table.

```
DELETE FROM [creator.] table-name
        [WHERE search-condition]
```

creator:
the userid of the owner of the table.

table-name :
the name of the table.

WHERE search-condition:
one or more conditions to apply in selecting data.

Examples:

1. DELETE FROM Quotations

2. DELETE FROM Quotations
   WHERE SuppNo = 64 AND PartNo = 207