.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

UNDERSTANDING RELATIONAL DATABASE DESIGN

Topics

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

Why is it important to learn about databases?

- In today's digital world, businesses need to gather a vast amount of data to support decision making, data analysis and other organizational activities.

- Data is typically stored electronically on a computer which allows for easy access, management, manipulation, and updating of data.

- A database is an organized collection of data or information that is set up and stored for easy retrieval electronically in a computer system. Importantly, databases simplify data management.

Data and Information

Data

- raw facts
- no context
- numbers and text

Data is meaningless

Information

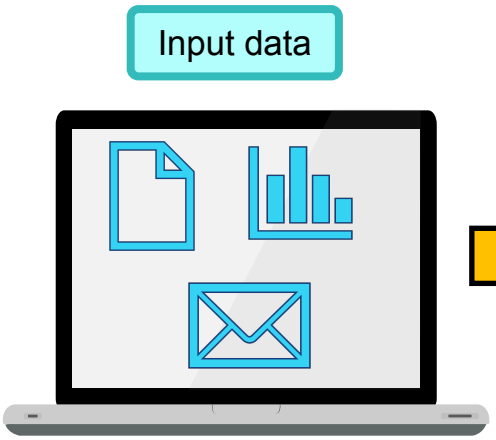
- data with context
- processed data
- value-added to data
  - ❖ summarized
  - ❖ organized
  - ❖ analyzed

Branch: Upper Boon Keng Road

Item: Oat Milk
Item Price: $11.95
No: of Item:2

Payment mode: Master Card

Database Management System



Input data

Data processing

Data is converted into information

```
SELECT *
FROM country;

SELECT *
FROM city;
```

100 %

Results | Messages

	id	country_name	country_name_eng	country_code
1	1	Deutschland	Germany	DEU
2	2	Srbija	Serbia	SRB
3	3	Hrvatska	Croatia	HRV
4	4	United Stated of America	United Stated of America	USA
5	5	Polska	Poland	POL

	id	city_name	lat	long	country_id
1	1	Berlin	52.520008	13.404954	1
2	2	Belgrade	44.787197	20.457273	2
3	3	Zagreb	45.815399	15.966568	3
4	4	New York	40.730610	-73.935242	4
5	5	Los Angeles	34.052235	-118.243683	4
6	6	Warsaw	52.237049	21.017532	5

A Database Management System (DBMS) is a software package designed to store and manage databases.

Popular DBMS

No. 1 Relational Database

- Commercial
- An expensive solution, suitable for large organizations
- Operating systems: Linux, OS X, Windows

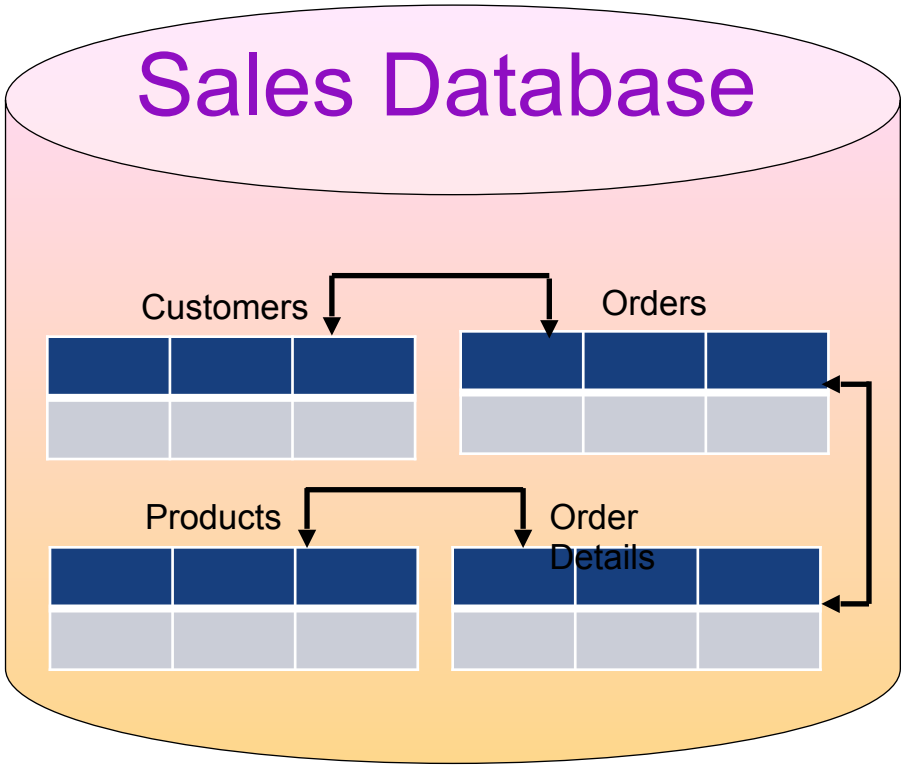
No.2 Relational Database

- Open Source
- Free, some features are missing e.g. stored procedures and check constraints
- Operating systems: Linux, OS X, Windows

No. 3 Relational Database

- Commercial
- An expensive solution, suitable for large organizations
- Operating systems: Linux, Windows

DBMS Concepts



A database contains many tables which have columns and rows. We refer tables as Entity.

Columns (Attributes/Fields): Characteristics of an entity. Each column has a distinct name and a datatype.

Rows: Records. Each row in a table must be uniquely identifiable

Relationships: describes an association among entities

- ➢ One-to-many relationship 1:M
- ➢ Many-to-many relationship M:M
- ➢ One-to-one relationship 1:1

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	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

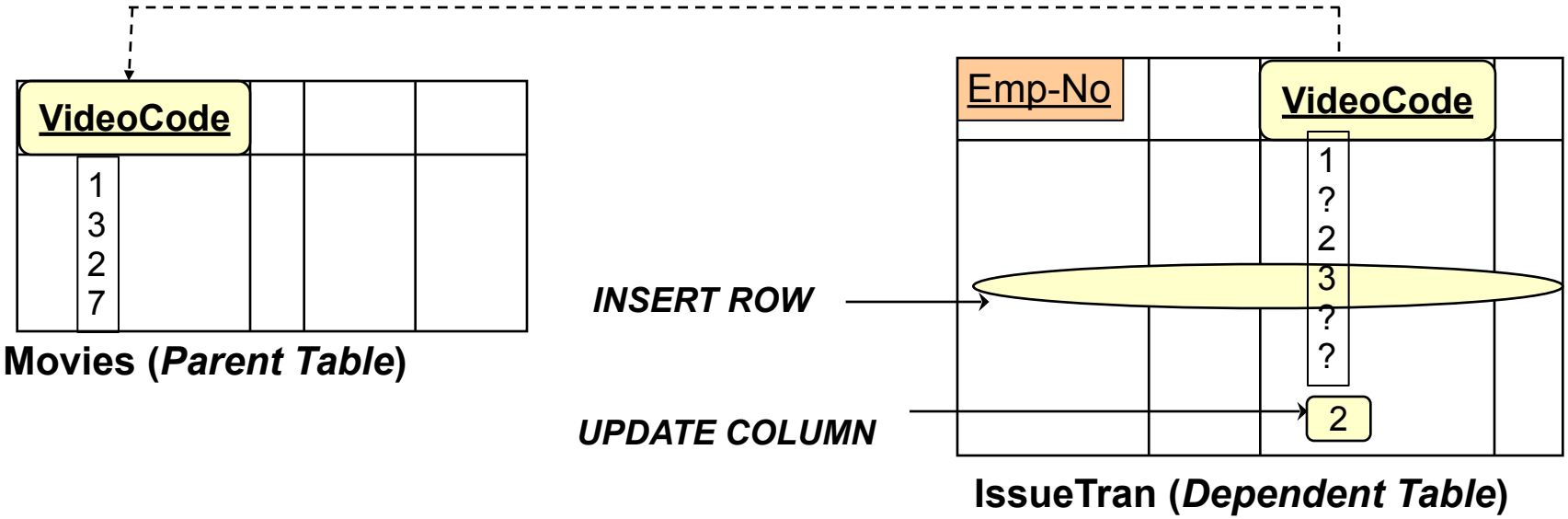
Composite Primary Key : Key with more than 1 column

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

Foreign Key

- Foreign key constraints
  - Enforcing Referential Integritiy



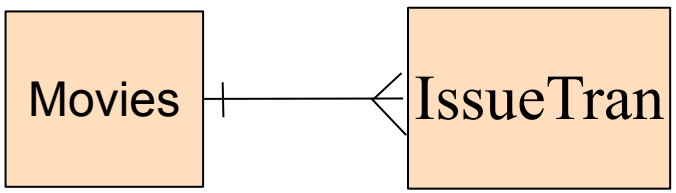
A row can be inserted or a column updated in the dependent table only if (1) there is a corresponding primary key value in the parent table, or (2) the foreign key value is set null.

Topics

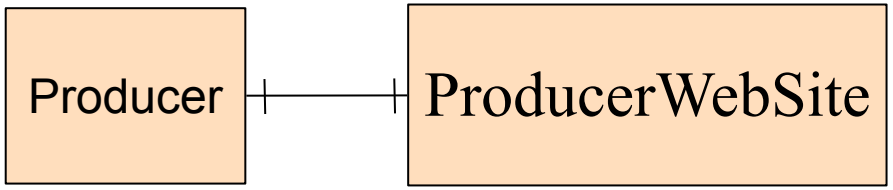
- Overview of Relational Database table Concepts
- Primary Key
- **Entity Relations Diagram**
- Foreign Key
- Structured Query Language (SQL)
- Summary

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

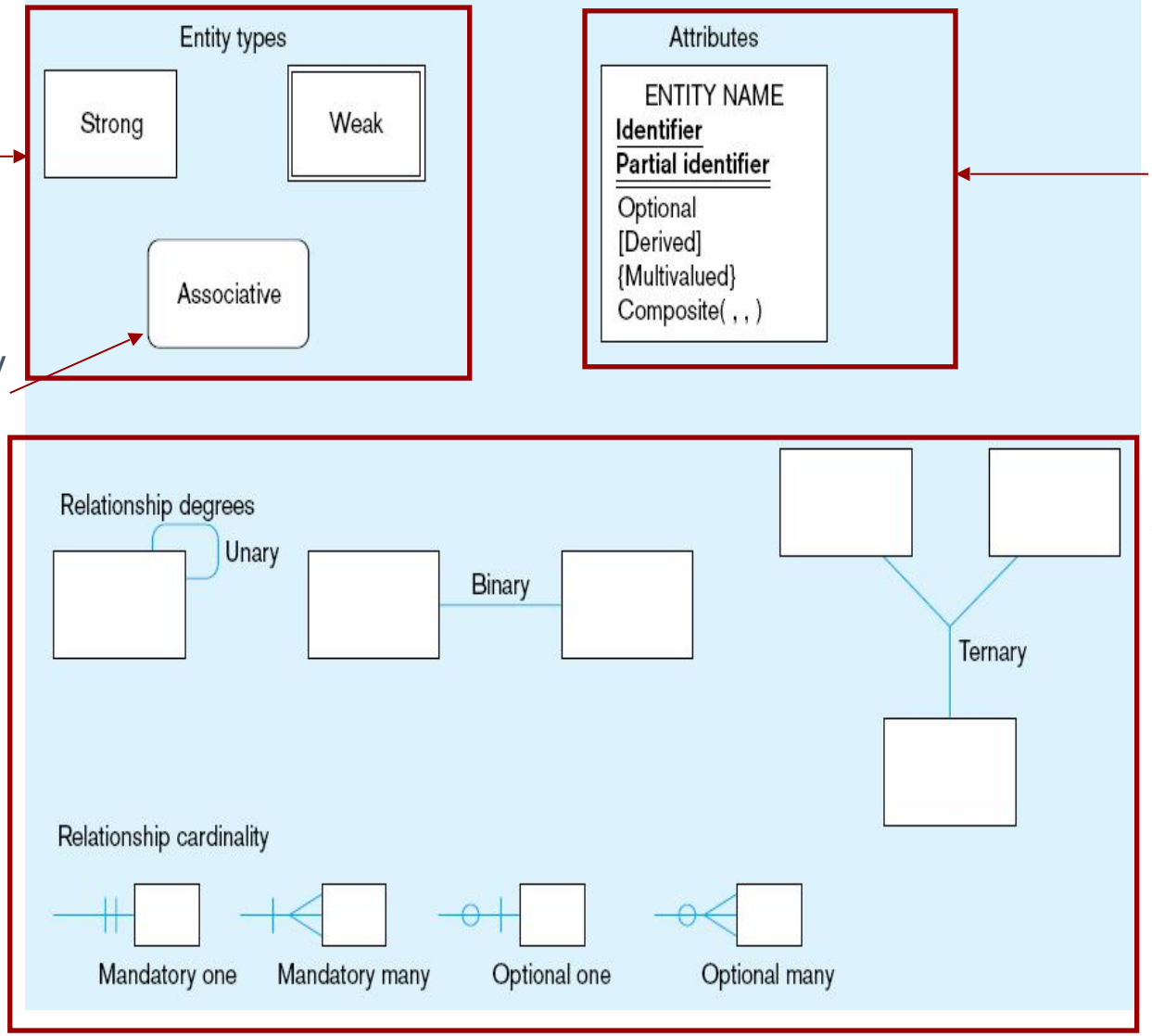
- Entities:
  - Entity: collection of attributes
    - Corresponds to a table
  - Entity instance: person, place, object, event, concept
    - often corresponds to a row in a table
    - Also termed as an element
- Relationships:
  - Relationship instance: link between entities
    - corresponds to primary key-foreign key equivalencies in related tables
- Attribute:
  - property or characteristic of an entity or relationship type
    - often corresponds to a field in a table

Basic E-R Notation

Entity symbols

A special entity that is also a relationship

Relationship degrees specify number of entity types involved

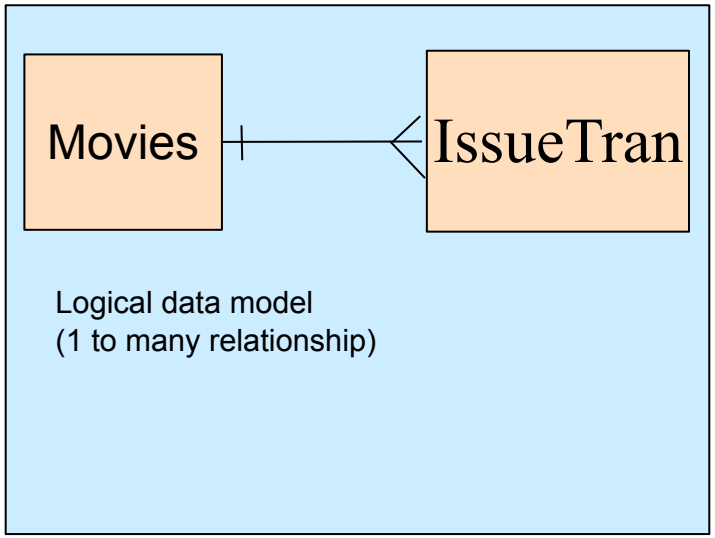


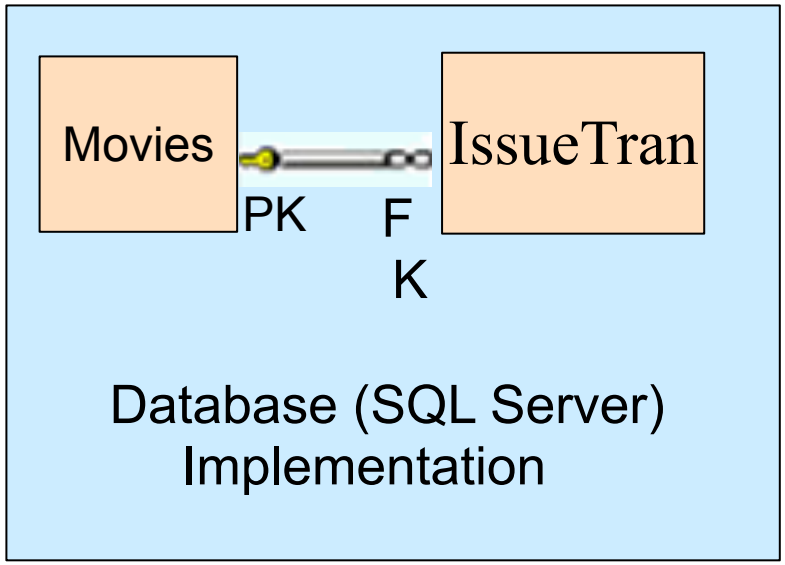
Attribute symbols

Relationship symbols

Relationship cardinalities specify how many of each entity type is allowed

Database Design and Implementation



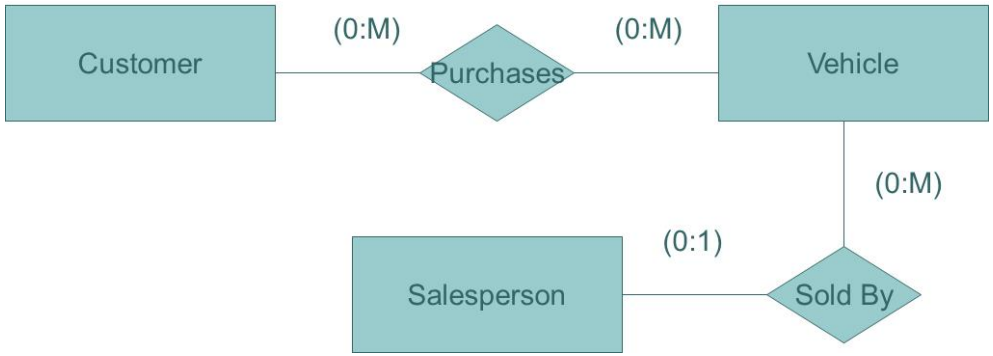


- Database does not enforce the creation of PK or FKs

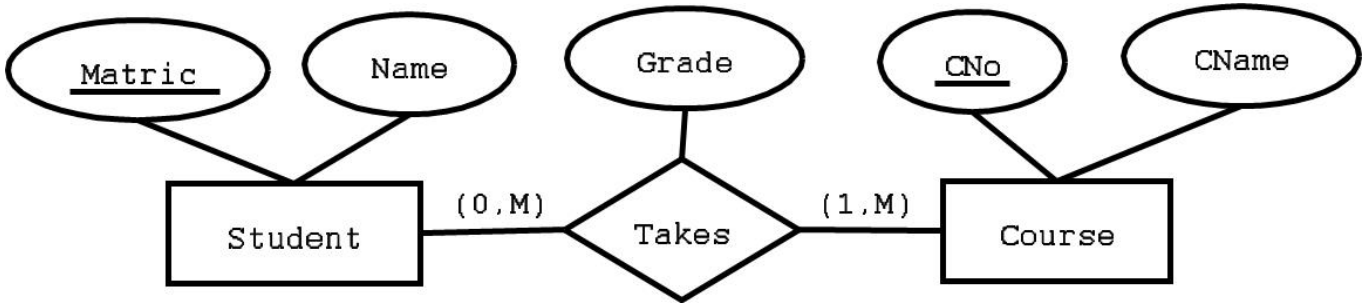
Different ERD Notations

Crow-foot Model

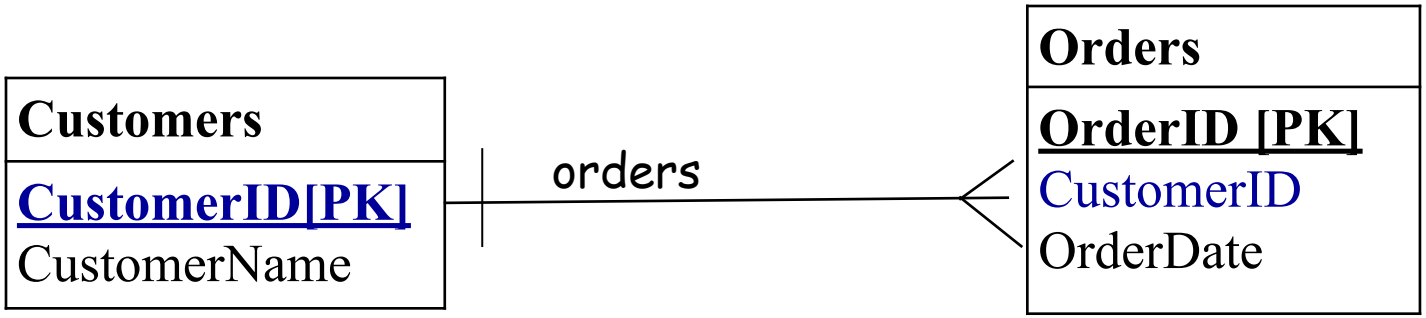
Chen Model



Modified Chen Model



Related Table - 1

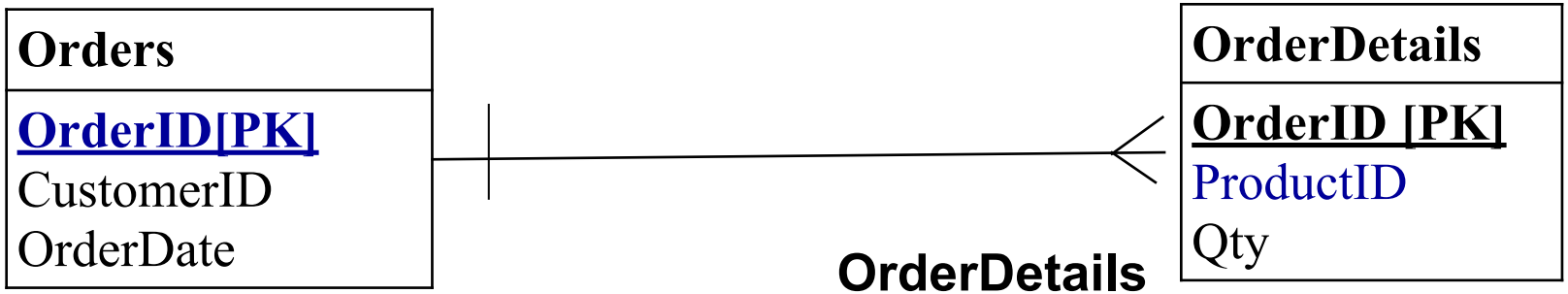


Tables are related through common attribute(s)

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

Related Table -2



Order

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

OrderDetails

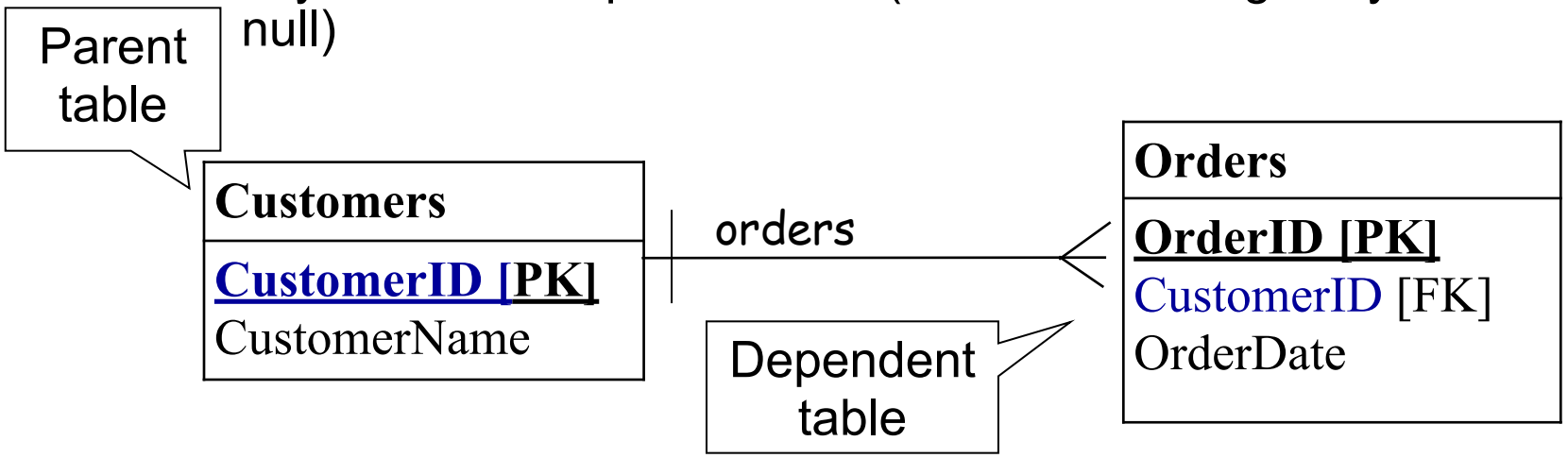
OrderID [PK]	ProductID [PK]	Qty
A1091	Pen	100
A1091	Pencil	200
A1091	Eraser	250
A1092	Pen	10
A1092	Pencil	50
A1093	Pen	80
A1093	Eraser	90

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

Database Diagram

In RDBMS, databases are illustrated using an ERD (entity relationship 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

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

SQLite Installation

About **Download** Blog Docs GitHub Gitter Stats Patreon DBHub.io

Downloads

(**Please** consider sponsoring us on Patreon 😄)

Windows

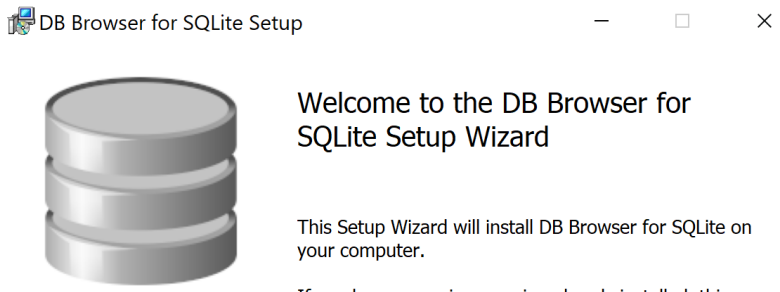
Our latest release (3.12.2) for Windows:

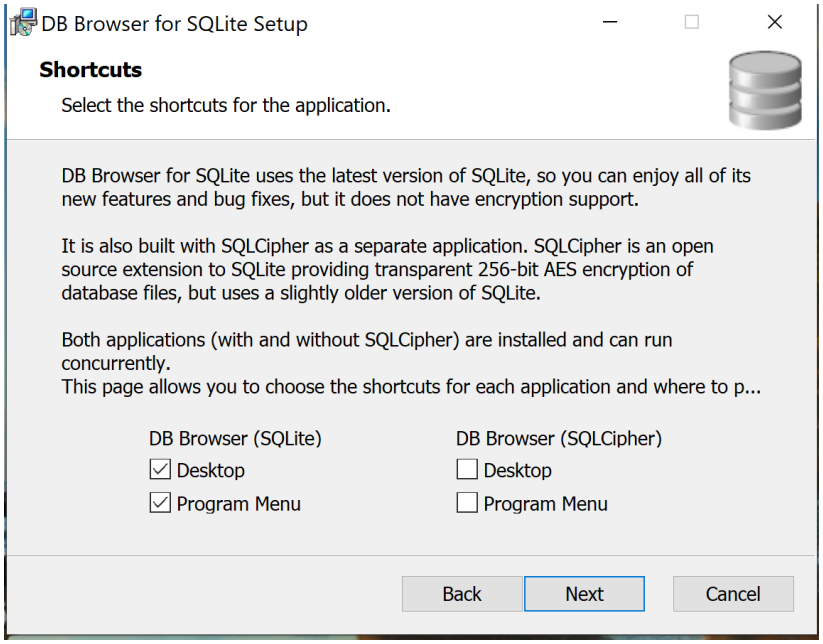
- DB Browser for SQLite - Standard installer for 32-bit Windows
- DB Browser for SQLite - .zip (no installer) for 32-bit Windows
- DB Browser for SQLite - Standard installer for 64-bit Windows
- DB Browser for SQLite - .zip (no installer) for 64-bit Windows

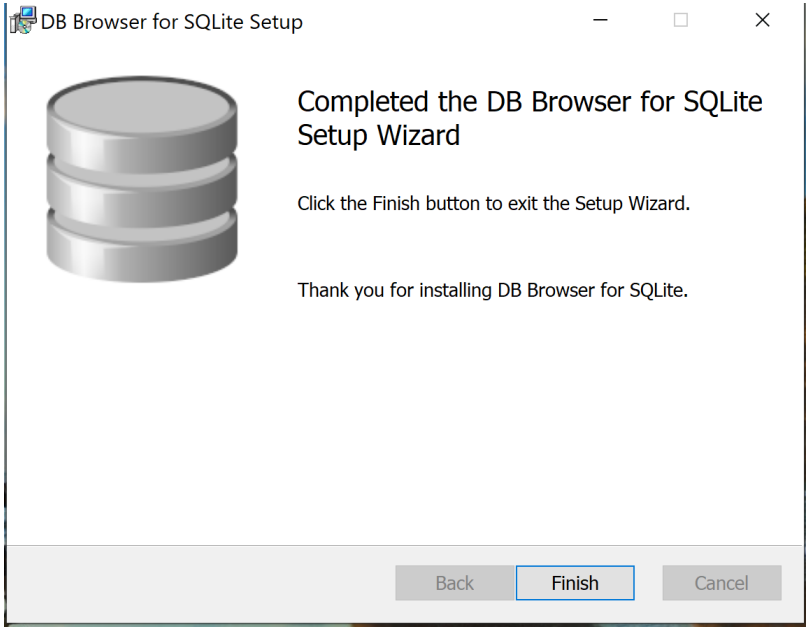
macOS

Our latest release (3.12.2) for macOS:

- DB Browser for SQLite (Intel)
- DB Browser for SQLite (Apple Silicon)


DB Browser for SQLite Setup
Welcome to the DB Browser for SQLite Setup Wizard
This Setup Wizard will install DB Browser for SQLite on your computer.
If you have a previous version already installed, this installation process will update it.
Back
Next
Cancel


DB Browser for SQLite Setup
Shortcuts
Select the shortcuts for the application.
DB Browser for SQLite uses the latest version of SQLite, so you can enjoy all of its new features and bug fixes, but it does not have encryption support.
It is also built with SQLCipher as a separate application. SQLCipher is an open source extension to SQLite providing transparent 256-bit AES encryption of database files, but uses a slightly older version of SQLite.
Both applications (with and without SQLCipher) are installed and can run concurrently.
This page allows you to choose the shortcuts for each application and where to p...
DB Browser (SQLite)
Desktop
Program Menu
DB Browser (SQLCipher)
Desktop
Program Menu
Back
Next
Cancel


DB Browser for SQLite Setup
Completed the DB Browser for SQLite Setup Wizard
Click the Finish button to exit the Setup Wizard.
Thank you for installing DB Browser for SQLite.
Back
Finish
Cancel

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

SQL SELECT STATEMENTS

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)

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

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 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

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

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

This statement retrieves five columns from the movies Table

- All rows in the Table is retrieved since there are no conditions stipulated

SQL Query Output

Project: Select a DISTINCT vertical subset of the Table

```
SELECT DISTINCT Rating
FROM Movies
```

Output:

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

	Rating
1	PG
2	R
3	U

Records Affected: 3

This statement retrieves a column from movies Table

- DISTINCT requests that the data displayed have no duplicated row(s)

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

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

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

This statement retrieves the specified columns and rows from Movies Table

- The Table retrieved consists of 5 columns that satisfy the conditions in the WHERE clause

SQL Query Output

NOT Condition

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

Output:

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

	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 Output

Using Alias: AS

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

Columns' Name have changed to Movie_Titles

Output:

	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

This statement retrieves a column from movies Table

- This statement demonstrates the retrieval of selected AS redefines the name of the result set column.(ie Movie_Title).

SQL Query Output

AND condition

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

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

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	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

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

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 Output

BETWEEN condition

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

The BETWEEN keyword allows conditions, in the WHERE clause, to be defined as a range

Output:

	TransactionID	CustomerID	VideoCode	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 Output

LIKE condition

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

String Pattern
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)

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 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 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

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

Using Sub Queries

The SELECT statement in brackets are called sub query

```
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

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 should match.

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 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 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

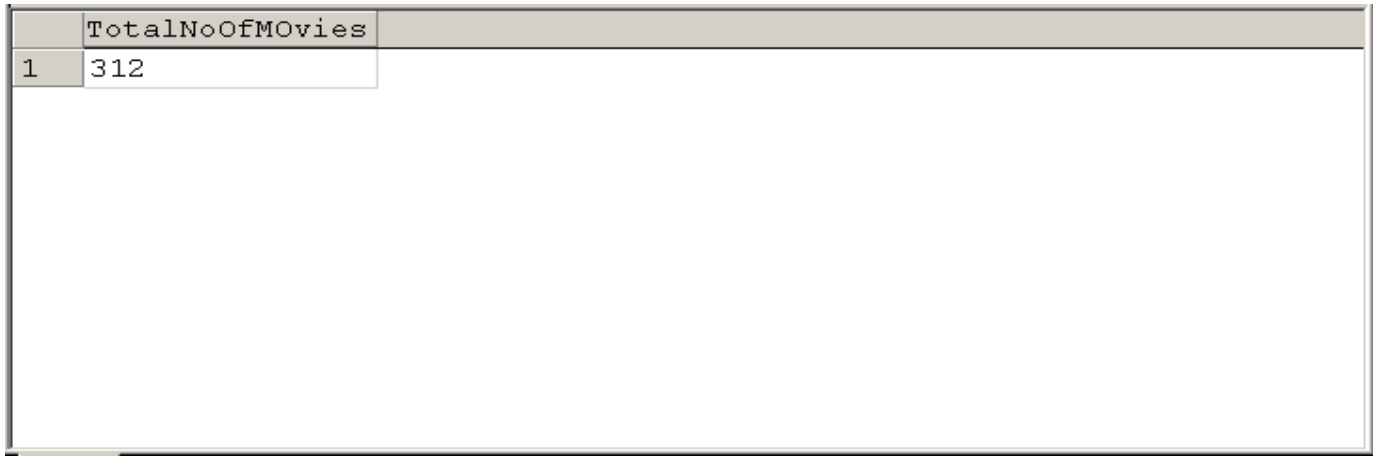
SQL Query Output

Using alias allows the naming of the columns that are derived.

COUNT function

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

Output:



	TotalNoOfMOvies
1	312

Records Affected: 312

This statement retrieves all data from Movies Table

- The COUNT function returns the “number of rows” that satisfy the query command : “SELECT * FROM Movies”

SQL Query Output

COUNT function

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

Output:

	DirectorsInMovies
1	311

Records Affected: 311

SQL Query Output

SUM function

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

Output:

	(No column name)
1	1098

Records Affected: 1

This statement retrieves one columns from Movies Table

- 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

AVG function

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

Output:

	(No column name)
1	1.8621794871794872

Records Affected: 1

This statement retrieves one columns from Movies Table

- 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.

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"

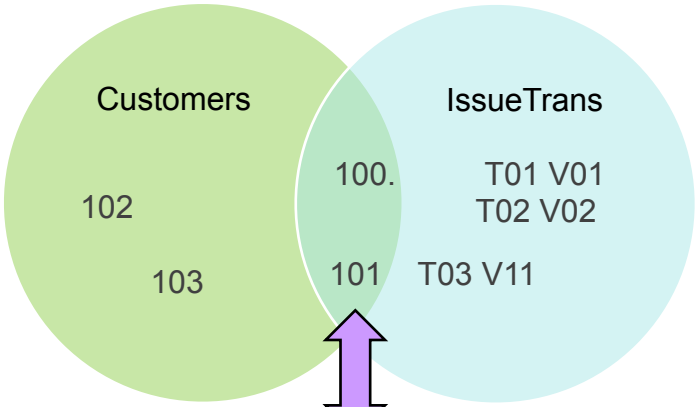
Id	Name
100	Chia
101	Derek
102	Esther
103	Venkat

table "IssueTrans"

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

Result

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



Inner join retrieve rows whose values exists in both tables

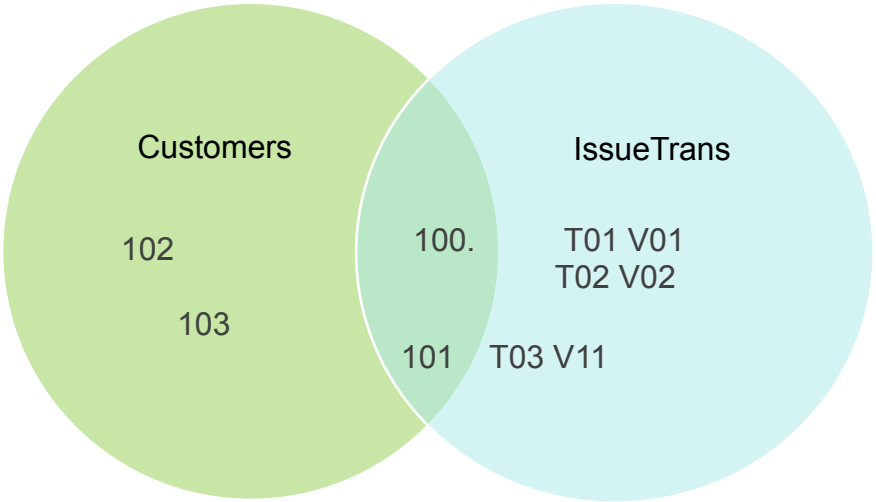
Left Outer Join (1)

- 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

Result

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

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
05	Ester	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

The database table "Staff"

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

The database table "Supervisor"

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

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

Duplicated Virtual Tables

self join:
the table joins with itself

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 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')
```

list the different country codes

CountryCode
SIN
IND

CountryCode	Num
SIN	3

Result

CountryCode	Num
IND	1

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

Top

- SELECT CustID FROM Customers
  LIMIT 2

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 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

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 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

This statement retrieves a column from Movies table

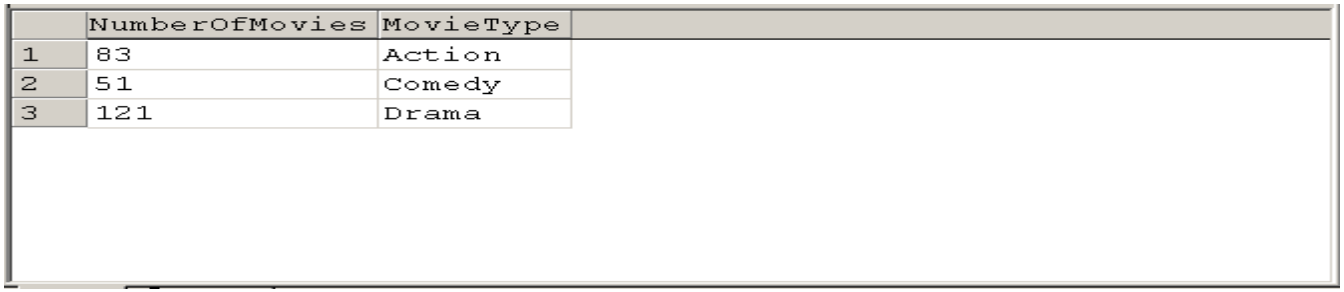
- COUNT(MovieType) in this case returns the “number of rows” aggregated for each returned value in MovieType column

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

This statement retrieves a column from Movies Table

- HAVING specifies a search condition that will be factored in the rows returned after the GROUP BY clause has taken effect

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

This statement retrieves two columns from Movies Table

- 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

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

This statement retrieves two columns from Movies Table

- 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

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
```

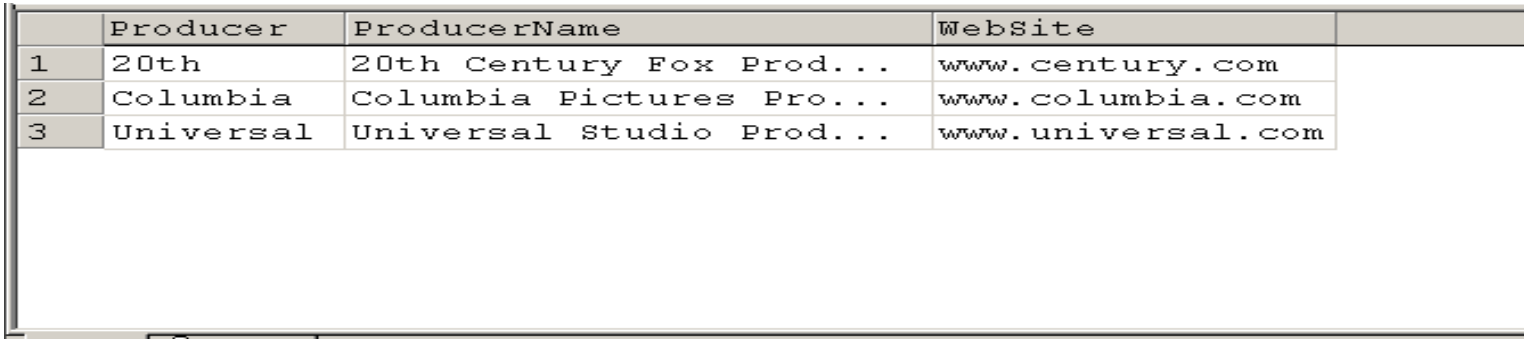
SQL Query Output

Joining Tables: INNER JOIN

Note: INNER JOIN can be represent with just JOIN because the default JOIN is the 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

This statement retrieves three columns from ProducerWebSite and Producers Table.

- Inner join retrieved rows whose value exists in *both* Tables.
- The WHERE condition acts as a condition for joining. It is often referred as JOIN condition.

SQL Query Output

Joining Tables: LEFT OUTER JOIN

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

Left Table

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

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

Are they any movies which have prequel?

DB Browser for SQLite - C:\Users\jocho\Desktop\NUS\video.db

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes | Open Project | Save Project | Attach Database | Close Database

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Documentaries | Filter in any column

	VideoCode	VideoTitle	MovieType	Rating	Price	Producer	Director	Media	TotalStock	NumberRented	PreviousEpisode
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1000	Lions Of Saharah	Animal ...	G	1.5	NULL	NULL	NULL	5	NULL	NULL
2	1001	Poverty in Russia 1	Real Life ...	G	2.0	NULL	NULL	NULL	4	NULL	NULL
3	1002	Lonely Planet: Japan	Tour Documentary	G	1.5	NULL	NULL	NULL	6	NULL	NULL
4	1003	Lonely Planet: Paris 1	Tour Documentary	G	2.5	NULL	NULL	NULL	3	NULL	NULL
5	1004	[redacted]									
6	1005	Lonely Planet: Turkey	Tour Documentary	G	3.5	NULL	NULL	NULL	3	NULL	NULL
7	1006	Birds of Prey	Animal ...	G	4.0	NULL	NULL	NULL	4	NULL	NULL
8	1007	Sharks: Predator or Prey	Animal ...	G	2.5	NULL	NULL	NULL	3	NULL	NULL
9	1008	[redacted]									
10	1009	Greatest Buildings on Earth	Architecture ...	G	2.5	NULL	NULL	NULL	5	NULL	NULL
11	1010	Greatest Wonders of the World	Architecture ...	G	2.0	NULL	NULL	NULL	6	NULL	NULL
12	1011	Crop Circles: Fact or Fiction	Mystery ...	G	1.5	NULL	NULL	NULL	2	NULL	NULL
13	1012	Animal Kindom of Amazon	Animal ...	G	3.0	NULL	NULL	NULL	5	NULL	NULL
14	1013	African Elephants	Animal ...	G	3.5	NULL	NULL	NULL	3	NULL	NULL

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.

```
select d1.VideoCode,VideoTitle as documentarytitle,MovieType, Rating,Price, Producer, Director, Media,TotalStock,NumberRented,PreviousEpisode
from Documentaries d1
```

	VideoCode	documentarytitle	MovieType	Rating	Price	Producer	Director	Media	TotalStock	NumberRented	PreviousEpisode
1	1000	Lions Of Saharah	Animal ...	G	1.5	NULL	NULL	NULL	5	NULL	NULL
2	1001	Poverty in Russia 1	Real Life ...	G	2.0	NULL	NULL	NULL	4	NULL	NULL
3	1002	Lonely Planet: ...	Tour Documentary	G	1.5	NULL	NULL	NULL	6	NULL	NULL
4	1003	Lonely Planet: Par...	Tour Documentary	G	2.5	NULL	NULL	NULL	3	NULL	NULL
5	1004	Poverty in Russia 2	Real Life ...	G	3.0	NULL	NULL	NULL	5	NULL	1001
6	1005	Lonely Planet: ...	Tour Documentary	G	3.5	NULL	NULL	NULL	3	NULL	NULL
7	1006	Birds of Prey	Animal ...	G	4.0	NULL	NULL	NULL	4	NULL	NULL
8	1007	Sharks: Predator ...	Animal ...	G	2.5	NULL	NULL	NULL	3	NULL	NULL
9	1008	Lonely Planet: Par...	Tour Documentary	G	1.5	NULL	NULL	NULL	3	NULL	1003
10	1009	Greatest Buildings...	Architecture ...	G	2.5	NULL	NULL	NULL	5	NULL	NULL
11	1010	Greatest Wonders...	Architecture ...	G	2.0	NULL	NULL	NULL	6	NULL	NULL
12	1011	Crop Circles: Fact ...	Mystery ...	G	1.5	NULL	NULL	NULL	2	NULL	NULL
13	1012	Animal Kindom of...	Animal ...	G	3.0	NULL	NULL	NULL	5	NULL	NULL
14	1013	African Elephants	Animal ...	G	3.5	NULL	NULL	NULL	3	NULL	NULL

Duplicate Table: D1

```
select d2.VideoCode,VideoTitle as documentaryprequel,MovieType, Rating,Price, Producer, Director, Media,TotalStock,NumberRented,PreviousEpisode
from Documentaries d2
```

	VideoCode	documentarypreque	MovieType	Rating	Price	Producer	Director	Media	TotalStock	NumberRented	PreviousEpisode
1	1000	Lions Of Saharah	Animal ...	G	1.5	NULL	NULL	NULL	5	NULL	NULL
2	1001	Poverty in Russia 1	Real Life ...	G	2.0	NULL	NULL	NULL	4	NULL	NULL
3	1002	Lonely Planet: ...	Tour Documentary	G	1.5	NULL	NULL	NULL	6	NULL	NULL
4	1003	Lonely Planet: Par...	Tour Documentary	G	2.5	NULL	NULL	NULL	3	NULL	NULL
5	1004	Poverty in Russia 2	Real Life ...	G	3.0	NULL	NULL	NULL	5	NULL	1001
6	1005	Lonely Planet: ...	Tour Documentary	G	3.5	NULL	NULL	NULL	3	NULL	NULL
7	1006	Birds of Prey	Animal ...	G	4.0	NULL	NULL	NULL	4	NULL	NULL
8	1007	Sharks: Predator ...	Animal ...	G	2.5	NULL	NULL	NULL	3	NULL	NULL
9	1008	Lonely Planet: Par...	Tour Documentary	G	1.5	NULL	NULL	NULL	3	NULL	1003
10	1009	Greatest Buildings...	Architecture ...	G	2.5	NULL	NULL	NULL	5	NULL	NULL
11	1010	Greatest Wonders...	Architecture ...	G	2.0	NULL	NULL	NULL	6	NULL	NULL
12	1011	Crop Circles: Fact ...	Mystery ...	G	1.5	NULL	NULL	NULL	2	NULL	NULL
13	1012	Animal Kindom of...	Animal ...	G	3.0	NULL	NULL	NULL	5	NULL	NULL
14	1013	African Elephants	Animal ...	G	3.5	NULL	NULL	NULL	3	NULL	NULL

Duplicate Table: D2

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

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 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

This statement retrieves a column from Movies and Documentaries Table.

- 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

Limiting Result Set: TOP n

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

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

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

- Some considerations in deciding to use Sub query vs Joins:
  - Use Join when data is drawn from more tables
  - 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. Use sub query when comparison is made to an aggregation of second table
  - For example:

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

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

DATA MANIPULATION LANGUAGE

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 ('Columbia', 'www.Columbia.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.

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

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))
```

- 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

table-name (column-name)

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))
```

PK

foreign key column

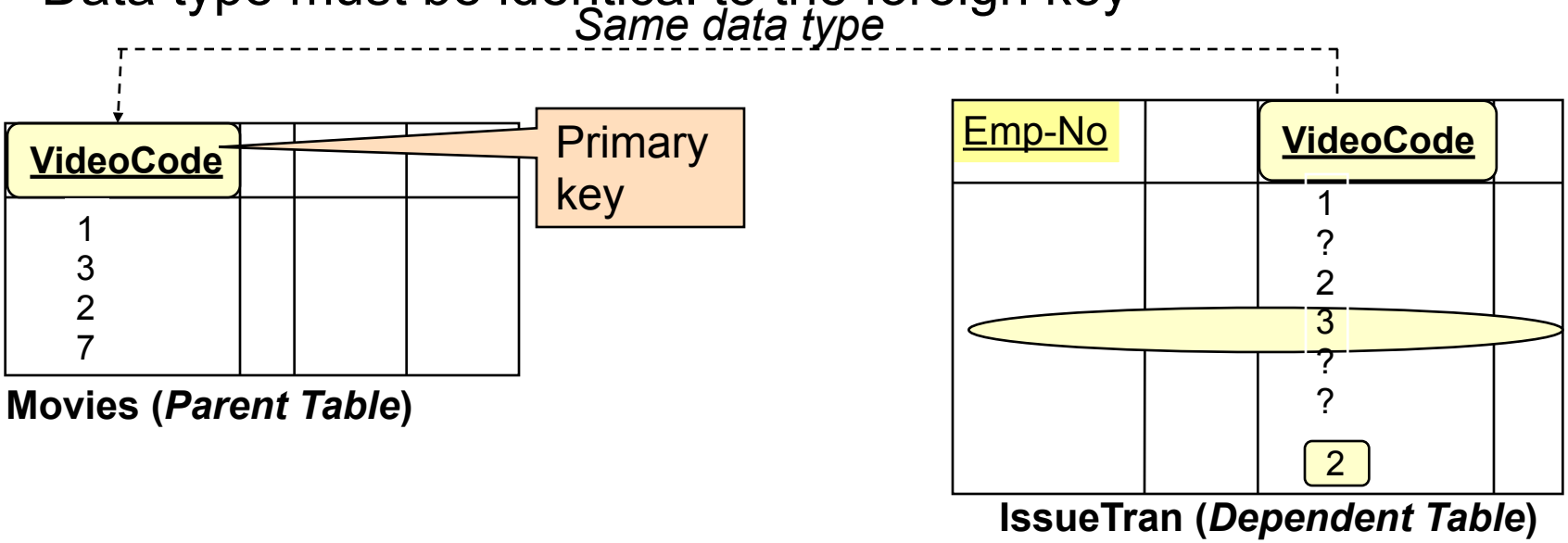
Constraint name

Parent Table - Producers

- 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.

How do you find a book in a library

How to look for a book?

- ➢ Find right category
- ➢ Lookup Index, find location
- ➢ Walk to aisle.
- ➢ Scan book titles.

Books(BID, name, year, publisher)
On which attributes would you build indexes?

An index speeds up selection on search key(s)

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.

Index built by DBMS

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

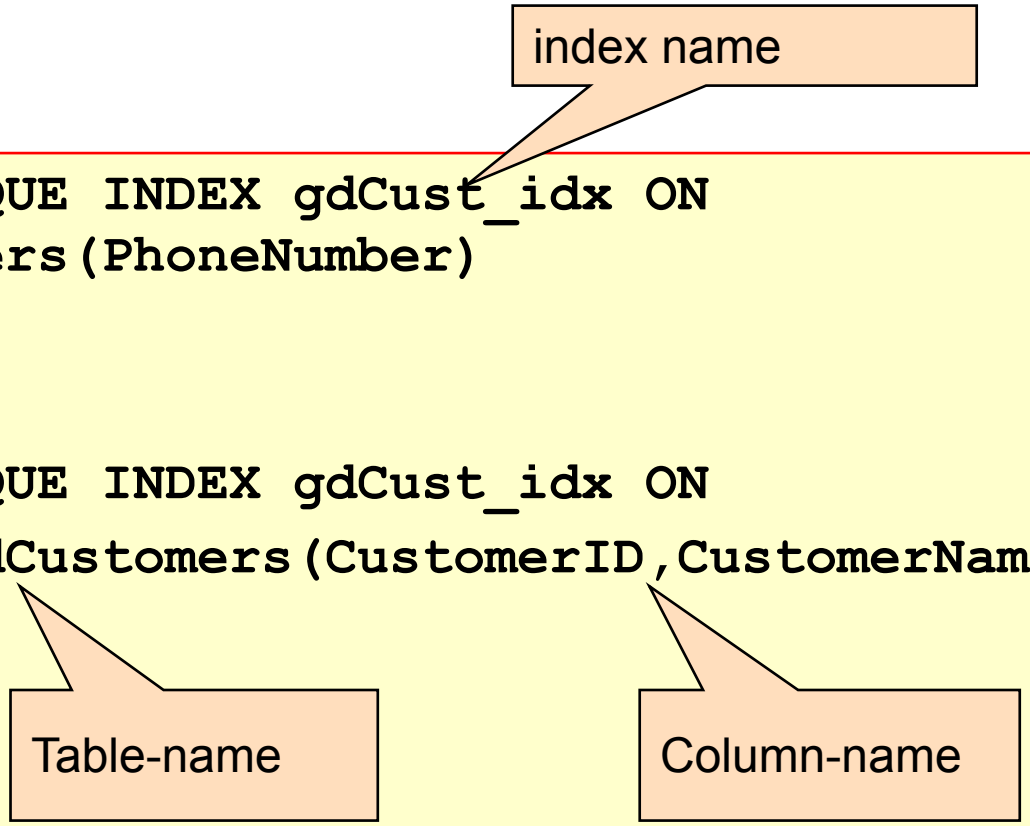
CREATE INDEX

- Example:

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


CREATE UNIQUE INDEX gdCust_idx ON
       GoodCustomers(CustomerID,CustomerName)


CREATE INDEX Cust_idx ON Customers(Address)
```



Disadvantages of indexes

- Every index increases the storage space in the database

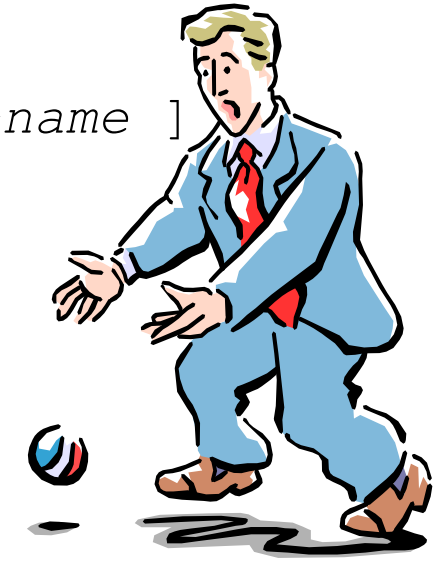
- When data are inserted, updated or deleted, the index must be updated. An index saves time in retrieval of data, but it costs time in insert, update or delete operation

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)
```

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

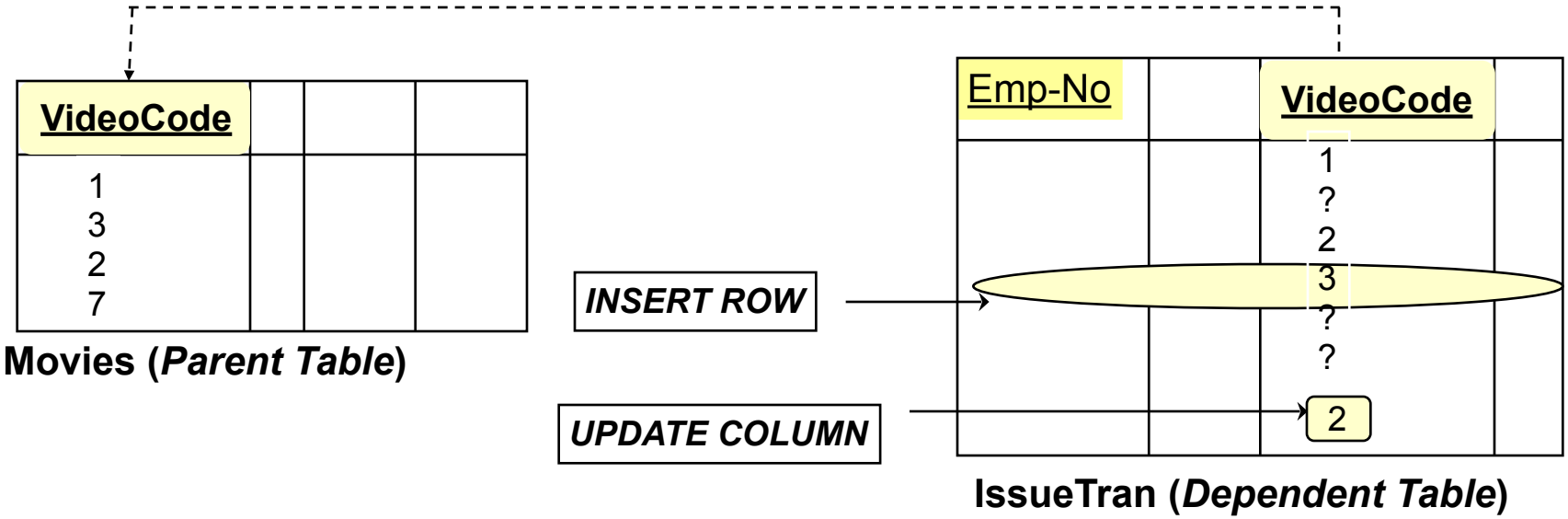
```
CREATE TABLE Producers
(Producer              nvarchar(50)   not null,
 ProducerName          nvarchar(50)   not null       UNIQUE,
 CountryCode           nvarchar(3)    not null,
 PRIMARY KEY(Producer,ProducerName),
 FOREIGN KEY(CountryCode)REFERENCES Country(CountryCode)
                      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

Use the ON DELETE CASCADE option to specify whether you want rows deleted in a child table when corresponding rows are deleted in the parent table. If you do not specify cascading deletes, the default behavior of the database server prevents you from deleting data in a table if other tables reference it.

If you specify this option, later when you delete a row in the parent table, the database server also deletes any rows associated with that row (foreign keys) in a child table. The principal advantage to the cascading-deletes feature is that it allows you to reduce the quantity of SQL statements you need to perform delete actions.

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.*

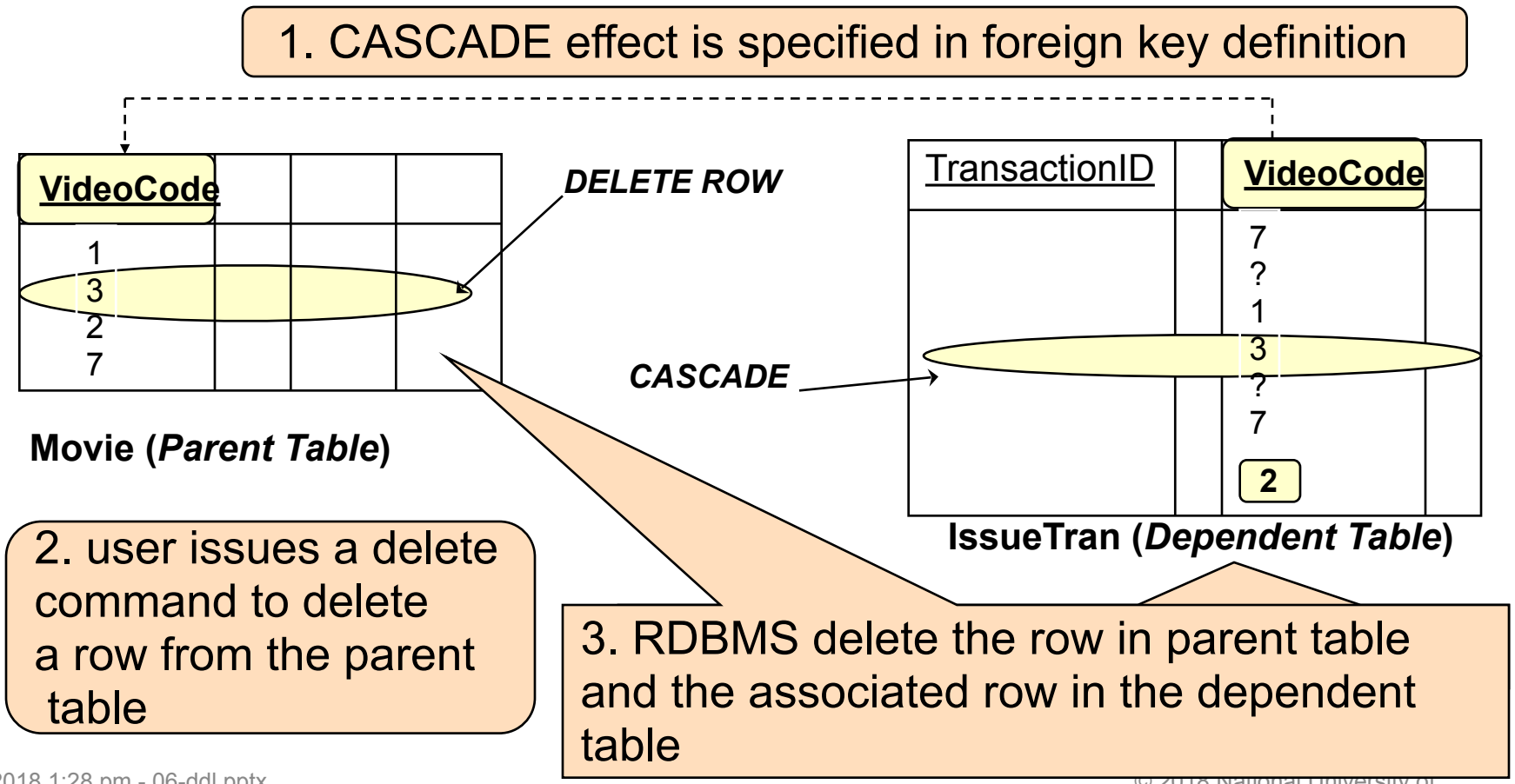


A row can be inserted or a column updated in the dependent table only if (1) there is a corresponding primary key value in the parent table, or (2) the foreign key value is set null.

Referential Integrity – Effects for Deletes Operation

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



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)
```

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

.NET PROGRAMMING
SQL PROGRAMMING AND DBMS

USER VIEWS

Objective

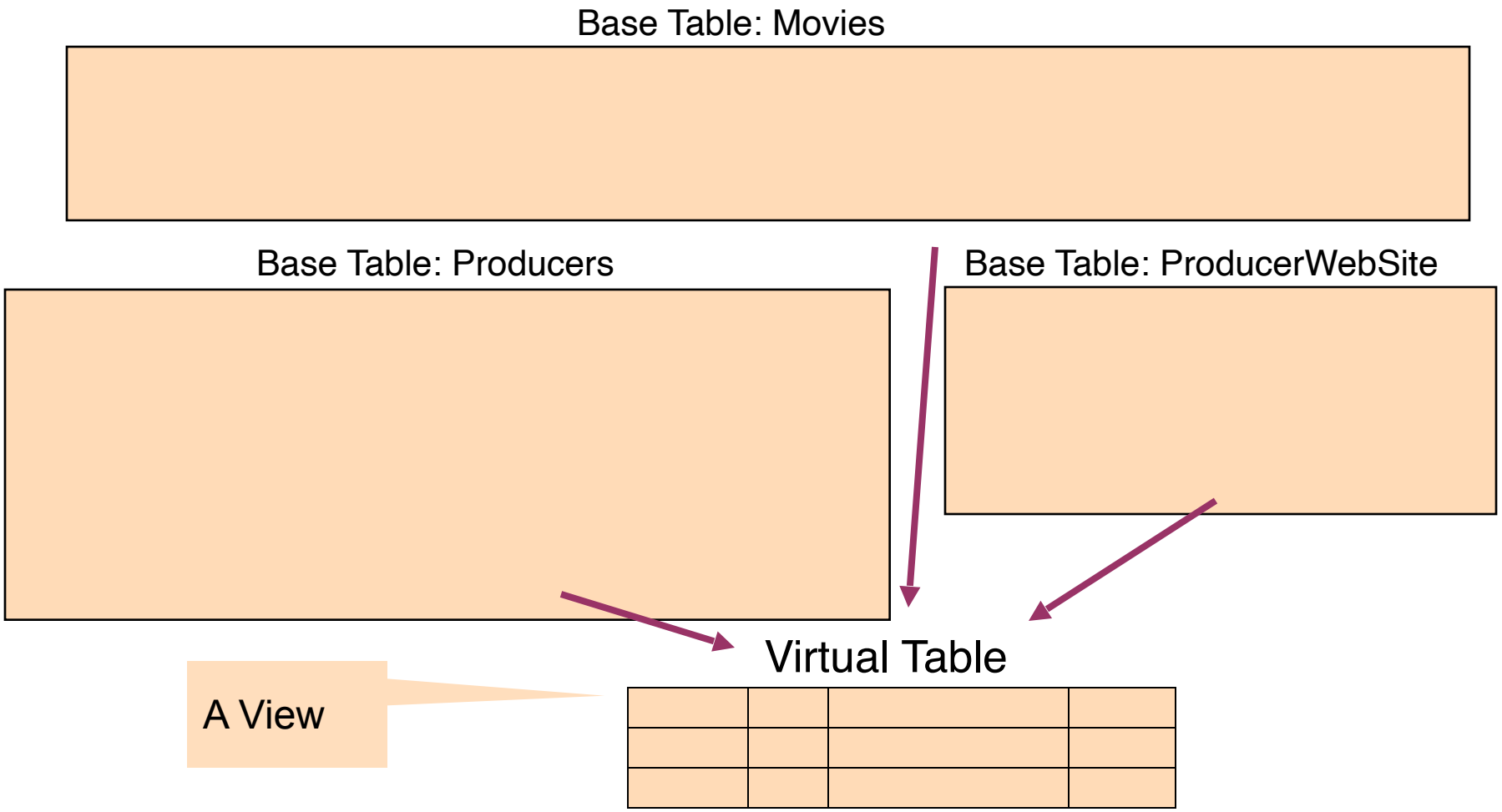
- By the end of this lesson, students should be able to:
  - Have an understanding of a user view and its purpose.
  - Understand and appreciate:
    - the use of SQL for userview definition and manipulation
    - the limitations of user views.

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?



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'.

```
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
```

Exercises

- Using Northwind database, try the following questions:
- Exercise Userview 1
  - Create a View Customer2016 containing Customer IDs and names, Product IDs and names for customers who have made orders on the year 2016.
- Exercise Userview 2
  - Using the View Customer2016, retrieve the Customer name, Product name and supplier names for the Customers who have made orders on the year 2016 according to Customer Name.
- Exercise Userview 3
  - Retrieve the Customer name and the number of products ordered by them in the year 2016.

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.

Summary

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

.NET PROGRAMMING

SQL PROGRAMMING AND DBMS

STORED PROCEDURES

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
```

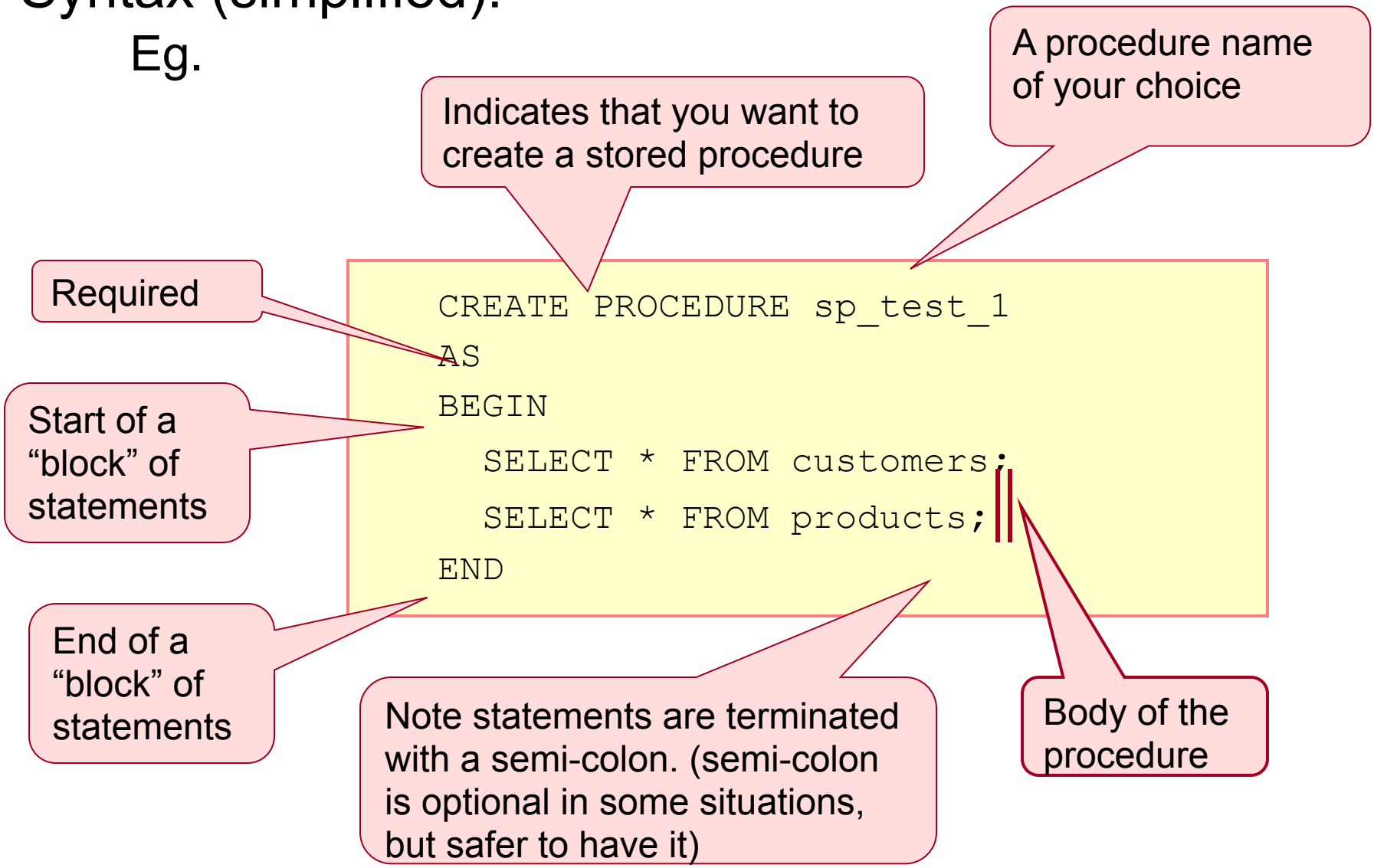
  - There can be multiple SQL statements

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
BEGIN
   SELECT *
   FROM Movies
   WHERE Rating= @var1 and
         TotalStock > @var2;
END
```

Or

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

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

Business Cases

- 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’?

Summary

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

Appendix A: Advance Query

- Apart from the Left Outer Join, we can also use a Right Outer Join, Full Join and Cross Join on tables/views for retrieving records.
- We will be discussing these Joins in the following order:
  - Right Outer Join
  - Full Join
  - Cross Join

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
```

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

Output:

	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 A

Joining Tables: FULL JOIN

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

Right Table

- This statement retrieves three columns from ProducerWebSite and Producers Table.
  - The FULL JOIN is similar to the equivalent of a LEFT OUTER JOIN followed by a RIGHT OUTER JOIN
  - There is no difference between the left and right table when a Full Join is applied to the tables to be joined.
  - There is no alternate syntax for a FULL JOIN.

Appendix A

Joining Tables: FULL JOIN

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

The results between a Full Join and a Right Outer Join is the same in this case because there is no Producer that is found in ProducerWebSite Table but not Producers Table.

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

Appendix A

Joining Tables: CROSS JOIN

```
SELECT P1.Producer, P2.WebSite
FROM ProducerWebSite P1
CROSS JOIN ProducerWebSite P2
```

- This statement retrieves two columns from the ProducerWebSite Table.
  - A CROSS JOIN produces a Cartesian product (ie all possible combinations) of the records on both side of the JOIN.
  - In other words if there are three records from each table respectively, a Cartesian product of all records would return a results set having the multiple of three and three (ie. nine) records.
  - Note that there is no Joining Condition (ie ON clause) for the CROSS JOIN because we are showing all possible combinations in the result set.

Appendix A

Joining Tables: CROSS JOIN

```
SELECT P1.Producer, P2.WebSite
FROM ProducerWebSite P1
CROSS JOIN ProducerWebSite P2
```

Since there are three records in the ProducerWebSite Table, a Cartesian Product of same Employees Table would result in a result set with nine records.
Notice that the Producer "20th" and the website "www.century.com" appears three times each.

Output:

	Producer	WebSite
1	20th	www.century.com
2	Columbia	www.century.com
3	Universal	www.century.com
4	20th	www.columbia.com
5	Columbia	www.columbia.com
6	Universal	www.columbia.com
7	20th	www.universal.com

Records Affected: 81

Appendix A

- Alternate syntax of a CROSS JOIN
  - You will be surprise that the syntax of a cross join is as follows:

```
SELECT P1.Producer, P2.WebSite
FROM ProducerWebSite P1
CROSS JOIN ProducerWebSite P2
```

  - Note that there are no WHERE clause in the Query statement.

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 follows:

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.

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

End of Appendix

End of Appendix