

Chapter 1: Introduction to Database

What is DBMS?:

DBMS is a collection of interrelated data and a set of programs to access the data. It provides an environment that is both *convenient* and *efficient* to use.

Applications of DBMS

Following are the list of applications of DBMS:

- Banking: transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

University Database Example:

- Application program examples
 - Add new students, instructors, and courses
 - Register students for courses, and generate class rosters
 - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- In the early days, database applications were built directly on top of file systems

Student Information System : Ishik University

Monday, 8 October, 2018 | LOGOUT

ISHIK UNIVERSITY
THE FUTURE IS HERE

my.ishik.edu.iq
STUDENT INFORMATION SYSTEM

HOME ACADEMICS PERSONAL INFORMATION HELP

Registration Curriculum Courses I have selected Departmental Courses Course Search Transcript Grade Calculation Weekly Schedule Attendance Exam Dates Interim Grades New Grades Messages Academic Calendar Contact Us Documents and Forms

Curriculum


GPA: 3.4
Akademik Danışman: Alan Amin (Office: Telephone:)
Department: COMPUTER ENGINEERING

COLORS:
BLUE: Indicates untaken courses. GREEN: Indicates achieved courses. ORANGE: Indicates taken courses. RED: Indicates failed courses. GRAY: Indicates exempt courses.

SEMESTER 1							SEMESTER 2						
CODE	COURSE	TYPE	CR	ECTS	YEAR	GRADE STATUS	CODE	COURSE	TYPE	CR	ECTS	YEAR	GRADE STATUS
CMPE 121	COMPUTER PROGRAMMING I	Ana	4		2009	AA	CMPE 122	COMPUTER PROGRAMMING II	Ana	4		2009	AA
CMPE 161	CALCULUS I	Ana	4		2009	BB	CMPE 162	CALCULUS II	Ana	4		2009	AA
CMPE 171	PHYSICS I	Ana	4		2009	BB	CMPE 172	PHYSICS II	Ana	4		2009	BA
CMPE 101	COMPUTER ORIENTATION	Ana	1	2	2009	AA	CMPE 154	DISCRETE MATHEMATICS	Ana	3		2009	AA
ELT 157	ADVANCED ENGLISH I	Ana	3		2009	BA	ELT 158	ADVANCED ENGLISH II	Ana	3		2009	BB
KUR 105	كوردی، ئوردی	Ana	2	1	2009	CB	KUR 106	KURDOLOGY II (KURDISH)	Ana	2	1	2009	BB
KUR 105	كوردی، ئوردی	Ana	2	1	2009	CB	KUR 106	KURDOLOGY II (KURDISH)	Ana	2	1	2009	BB
BUS 103	INTRODUCTION TO BUSINESS AND MANAGEMENT I	NA/INT	3		2010	AA	IT 431	COMPUTER GRAPHICS (SOLID WORKS)	NA/INT	3		2010	AA

SEMESTER TOTAL CREDITS: 23 / 20

SEMESTER TOTAL CREDITS: 25 / 22



PERSONNEL INFORMATION SYSTEM

8 October 2018, Monday

[Home](#) : [TÜRKÇE](#) : [LOCAL](#) : [Contact Us](#) : [Logout](#)

Name Surname: MUSA M.AMEEN

Department:

Registration No: A-2140

E-mail: musa.ameen@ishik.edu.iq

IP Port: [REDACTED]

Room No: 232-Main Building

Telephone:

Internal No: 1275


To change main details
[Click Here](#)

ACADEMICS

- ▶ Courses
- ▶ Research Projects
- ▶ Articles ▶ Books
- ▶ Survey
- ▶ Syllabus
- ▶ Schedule
- ▶ Birthdays of Students
- ▶ Contact Information of Personnel
- ▶ My Weekly Schedule
- ▶ Messaging Service
- ▶ CV Preview
- ▶ Edit CV

Courses

WARNING



Problem reports: pbs@ishik.edu.iq Please include Year, semester, course code, section.

Please always keep a copy of your documents(syllabus, attendance, grades, etc.) on your computer.

Coordinator can fill the shared courses syllabus and their lecturers should present their syllabus plan to coordinator.

EXCEPT FIRST GRADE COURSES FALL AND YEARLY SYLLABUS FORM IS READY FOR SUBMISSION. LAST DATE IS 15/10/2018

2018/1 : Current system year/term.

2015 - Fall [Spring](#) [Summer](#) 2016 - Fall [Spring](#) [Summer](#) 2017 - Fall [Spring](#) [Summer](#)

2018 - [Fall](#) [Spring](#) [Summer](#)

Year/ Semester	Course	Syllabus Status	Student List	Attendance	Attendance Sheet	Attendance Entry	Minor Grades	Grades	Final Makeup Grades
2018/1 CMPE Active Survey Result	CMPE 121/A Computer Programming I	No Syllabus	Download Photos E-mails	Click Here	Click Here	% Click Here	Click Here	Click Here	Click Here
2018/1 CMPE Active Survey Result	CMPE 201/A Object Oriented Programming		Download Photos E-mails	Click Here	Click Here	% Click Here	Click Here	Click Here	Click Here
2018/1 IT Active	IT 215/A Database		Download Photos	Click Here	Click Here	% Click Here	Click Here	Click Here	Click Here

ADVISORY/COURSE REGS.

- ▶ Student List
- ▶ Search Student

DEPARTMENT M.

- ▶ New Students
- ▶ Unlock Terminate
- ▶ Grades by Classes
- ▶ Quotas of Courses
- ▶ Annual Courses
- ▶ Course Statistics
- ▶ Success Statistics
- ▶ Survey Statistics
- ▶ NA Statistics
- ▶ GPA Ranking
- ▶ Course Descriptions

ISO 9001:2015

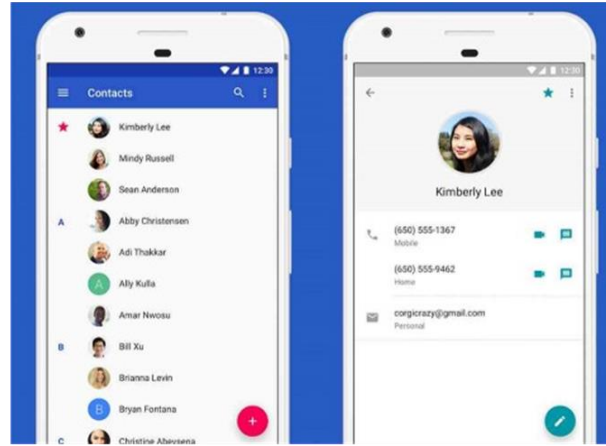
POS System Database Example:

[illegible]

Simple Database Example:

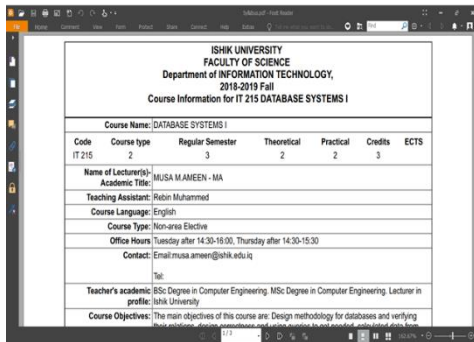


Dictionary App

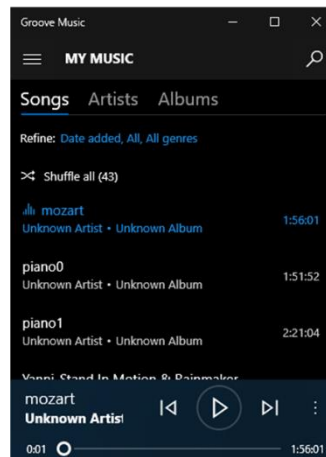


Phone Contacts

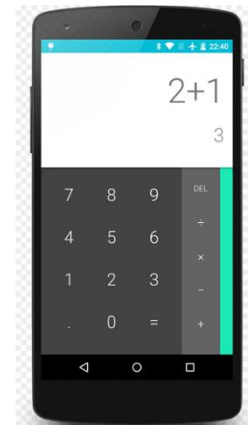
Apps without Database:



PDF Reader



Music Player



Calculator

Drawbacks of using file systems to store data:

Following are the drawbacks of File System:

- **Data redundancy and inconsistency:** Multiple file formats, duplication of information in different files
- **Difficulty in accessing data:** Need to write a new program to carry out each new task
- **Data isolation:** multiple files and formats
- **Integrity problems:** Hard to add new constraints or change existing ones
- **Atomicity of updates:** Failures may leave database in an inconsistent state with partial updates carried out
- **Concurrent access by multiple users:** Concurrent access needed for performance
- **Security problems:** Hard to provide user access to some, but not all, data.

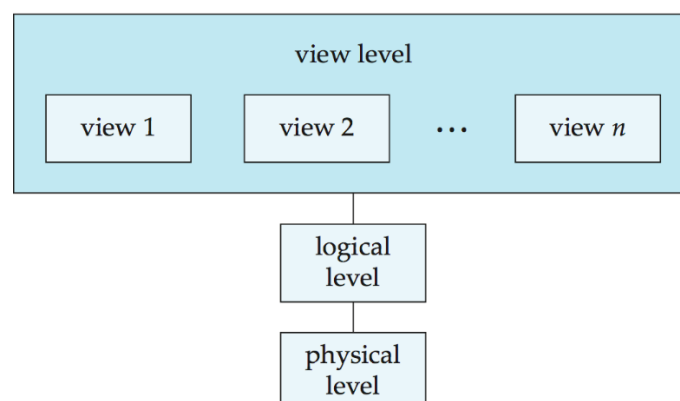
Database systems offer solutions to all the above problems

Levels of Abstraction:

- **Physical level:** describes how a record (e.g., customer) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.
 - **type instructor = record**
 - *ID* : string;
 - *name* : string;
 - *dept_name* : string;
 - *salary* : integer;
 - end;
- **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

View of Data:

An Architecture for a Database System:



Data Models:

- **Data models** define how the logical structure of a **database** is modelled. **Data Models** are fundamental entities to introduce abstraction in a **DBMS**.
- **Data models** define how **data** is connected to each other and how they are processed and stored inside the system.
- **Data Model** is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
- A data model provides a way to describe the design of a database at the physical, logical, and view levels.

The data models can be classified into four different categories:

- **Relational Model.** The relational model uses a collection of tables to represent both data and the relationships among those data.
- **Entity-Relationship Model.** The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and *relationships* among these objects.
- **Object-Based Data Model.** Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology.
- **Semi-structured Data Model.** The semi-structured data model permits the specification of data where individual data items of the same type may have different sets of attributes.

Historically, the **network data model** and the **hierarchical data model** preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modelling data.

Relational Model:

- The **relational model** uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name.
- Tables are also known as **relations**.
- The relational model is an example of a record-based model.
- Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type.
- Each record type defines a fixed number of fields, or attributes.
- The columns of the table correspond to the attributes of the record type.
- The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

Example of tabular data in the relational model:

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table**A Sample Relational Database:**

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Relational DBMS:

- A database management system that stores data in the form of related tables is called **Relational Database Management System**.
- **Edgar F. Codd** at IBM invented the relational database in 1970.
- Relational databases help solve problems as they are designed to create tables & then combine the information in interesting ways to create valid information.

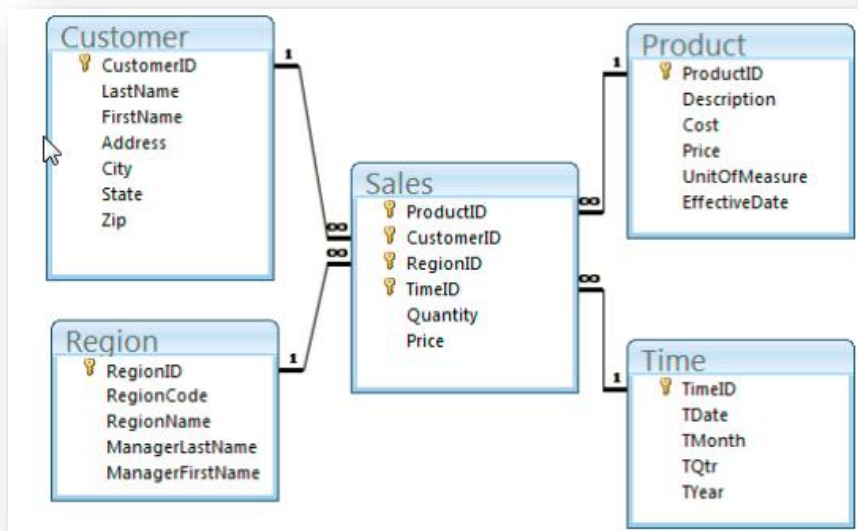
Typical example of Relational DBMS.

Typical examples of Relational DBMS are:

- Microsoft Access
- Microsoft SQL Server
- Sybase
- IBM DB2
- Oracle
- Ingres
- MySQL
- PostgreSQL

Database Schema:

- A database schema is a way to logically group objects such as tables, views, stored procedures etc.
- Think of a schema as a container of objects.



Chapter 2: Database Design Process

A database design process:

- **Step 1: Define the Purpose of the Database (Requirement Analysis)**
 - This helps prepare for the remaining steps.
 - Gather all of the types of information to record in the database, such as product name and order number.
- **Step 2: Find and organize the information required**
 - Divide information items into major entities or subjects, such as Products or Orders.
 - Each subject then becomes a table.
- **Step 3: Gather Data, Organize in tables and Specify the Keys**
 - Decide what information needs to be stored in each table.
 - Each item becomes a field, and is displayed as a column in the table.
 - Choose each table's primary key. The primary key is a column, or a set of columns, that is used to uniquely identify each row
- **Step 4: Create Relationships among Tables**
 - Look at each table and decide how the data in one table is related to the data in other tables.
- **Step 5: Refine & Normalize the Design**
 - Apply the so-called normalization rules to check whether your database is structurally correct and optimal.

Normalization:

- **Normalization** is the process of organizing the data in the database. Normalization is used to minimize the redundancy from a relation or set of relations.
- It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.

First Normal Form (1NF):

A table is 1NF if every cell contains a single value, not a list of values. This properties is known as atomic. 1NF also prohibits repeating group of columns such as item1, item2,..., item N. Instead, you should create another.

Second Normal Form (2NF):

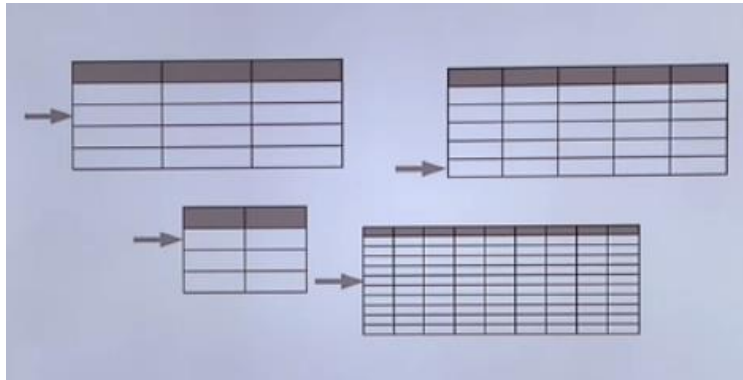
A table is 2NF, if it is 1NF and every non-key column is fully dependent on the primary key. Furthermore, if the primary key is made up of several columns, every non-key column shall depend on the entire set and not part of it.

Third Normal Form (3NF):

A table is 3NF, if it is 2NF and the non-key columns are independent of each other's. In other words, the non-key columns are dependent on primary key, only on the primary key and nothing else.

Unique Values and Primary Keys:

- Almost all table in a database require a key.
- The key is a way to identify just one particular row in a table



- A key is used to guarantee a unique column for a row
- If one of the column is defined as containing a unique value even if there were a million rows in this table, the same value cannot occur more than once in that column.

unique	not unique
153	
374	
352	
376	
153	

- Most of your columns don't need to be unique and shouldn't be.
- But some of the columns must be unique like Social Security Numbers or ISBN Numbers of books

FirstName (text)	LastName (text)	HireDate (date)	Grade (numeric)	Salary (currency)	SocialSecurity (text, unique)
Alice	Mann	4/4/2009	4	75000	55-65-1231
James	Black	3/1/2010	4	75000	55-65-1231
Calista	Guerra	10/1/2006	6	80000	54-23-1255
Fay	Fitzgerald	7/21/2002	7	100000	87-92-2341
John	Bowen	11/11/2011	3	45000	43-23-1234

- This kind of data are naturally unique and there should never be duplicates
- But much of the time there isn't one piece of naturally unique data.
- So you will make one instead. You will tell the database to generate a unique column.

EmployeeID (numeric, unique)	FirstName (text)	LastName (text)	HireDate (date)	Grade (numeric)	Salary (currency)	SocialSecurity (text, unique)
507	Alice	Mann	4/4/2009	4	75000	55-65-1231
602	James	Black	3/1/2010	4	75000	55-65-1231
312	Calista	Guerra	10/1/2006	6	80000	54-23-1255
78	Fay	Fitzgerald	7/21/2002	7	100000	87-92-2341
523	John	Bowen	11/11/2011	3	45000	43-23-1234

- Most DBMSs help you to generate those kind of columns with these kind of values.
- These column refers to Primary Key.

Primary Key (PK)

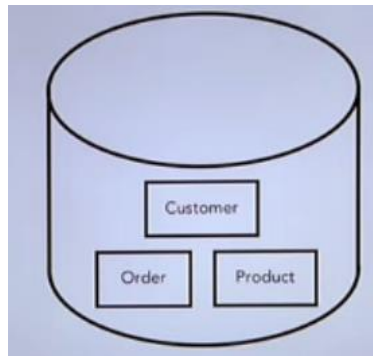
EmployeeID (numeric, unique)	FirstName (text)	LastName (text)	HireDate (date)	Grade (numeric)	Salary (currency)	SocialSecurity (text, unique)
507	Alice	Mann	4/4/2009	4	75000	55-65-1231
602	James	Black	3/1/2010	4	75000	55-65-1231
312	Calista	Guerra	10/1/2006	6	80000	54-23-1255
78	Fay	Fitzgerald	7/21/2002	7	100000	87-92-2341
523	John	Bowen	11/11/2011	3	45000	43-23-1234

What is Primary Key?:

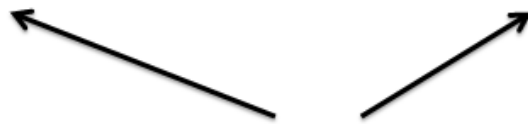
A **primary key** is a special relational database table column (or combination of columns) designated to uniquely identify all table records. A **primary key's** main features are: It must contain a unique value for each row of data. It cannot contain null values.

Defining Relationships:

- Any Database begins with defining tables, vital next step is add relationships among tables.
- Because much of your data is naturally connected.
- You are not trying to invent relationships that don't exist. You are trying to describe what's already there.



<u>sID</u>	sName	Age	<u>cID</u>	cName	Credit
111	Dara	24	IT215	Database	4
222	Nawzad	23	IT201	Web	3
333	Zara	24	IT301	Kurdology	2



**Two Primary
Keys??**

What's wrong??

Insert Anomaly:

<u>sID</u>	sName	Age	<u>cID</u>	cName	Credit
111	Dara	24	IT215	Database	4
222	Nawzad	23	IT201	Web	3
333	Zara	24	IT301	Kurdology	2
111	Dara	24	IT301	Kurdology	2

- Primary key conflict
- Multiple copy of data

Update Anomaly:

- Difficulty in updating all the data that's related to each other.

Delete Anomaly:

- What will happen if we delete **student 222**
- Deleting a row will delete all the records exist in that row.

What is the Solution??

- Separate them into two tables

<u>ID</u>	sName	Age
111	Dara	24
222	Nawzad	23
333	Zara	24

Student

<u>ID</u>	cName	Credit
IT215	Database	4
IT201	Web	3
IT301	Kurdology	2

Course**Relationship:**

A **relationship**, in the context of databases, is a situation that exists between two relational **database** tables when one table has a foreign key that references the primary key of the other table.

The various types of Relationship are:

- One-to-One
- One-to-Many
- Many-to-Many

One-to-One Relationship:

In a one-to-one relationship, one record in a table is associated with one and only one record in another table.

For example, in a school database, each student has only one student ID, and each student ID is assigned to only one person.

One-to-Many Relationship:

In a one-to-many relationship, one record in a table can be associated with one or more records in another table.

For Example, **One** department can have **Many** students. So the relation is **One-to-Many** between **department** and **student** table

Information about student and department are different but they are related to each other. So we need to formally describe relationships between our tables

<u>dID</u>	dName	Building
123	IT	A1
444	Computer	A1
555	Dentistry	B1

Department

<u>ID</u>	sName	Age
111	Dara	24
222	Nawzad	23
333	Zara	24

Student

<u>dID</u>	dName	Building
123	IT	A1
444	Computer	A1
555	Dentistry	B1

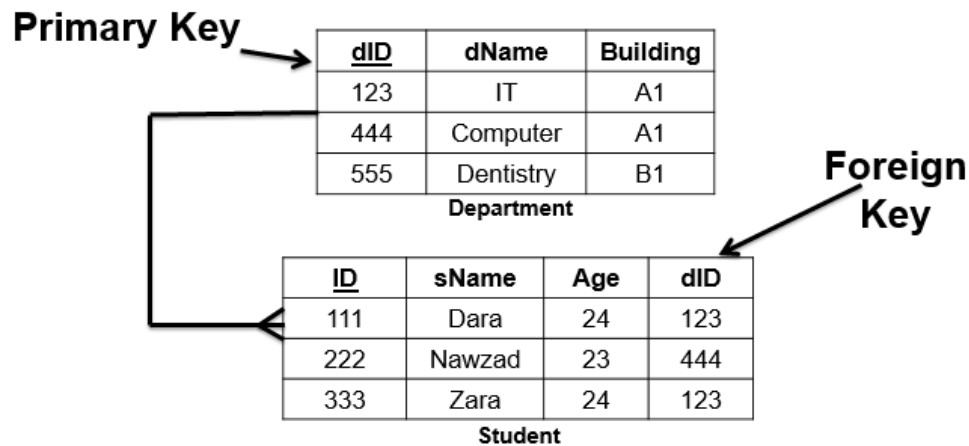
Department

<u>ID</u>	sName	Age	dID
111	Dara	24	123
222	Nawzad	23	444
333	Zara	24	123

Student

**Foreign
Key**



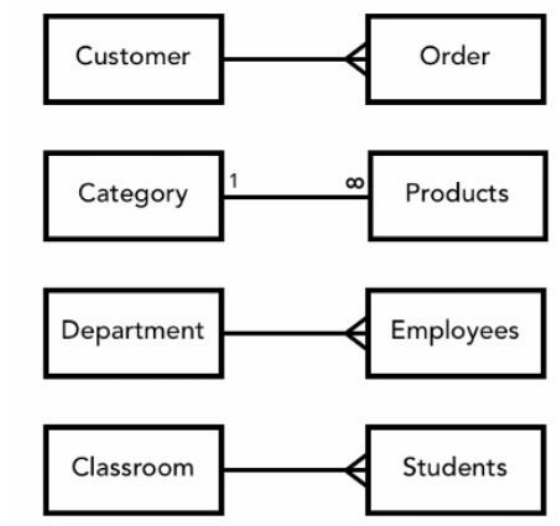


- **dID** in **Student** table is not a primary key but Foreign Key and it is not unique.

What is Foreign Key?:

A **foreign key** is a column or group of columns in a relational database table that provides a link between data in two tables. It acts as a cross-reference between tables because it references the primary **key** of another table, thereby establishing a link between them.

- Either we can go from the Department row or get dID then go to Student table and get all students of the department.
- Or we can go from the Student row and get dID then go to Department table and find which student is associated with that department

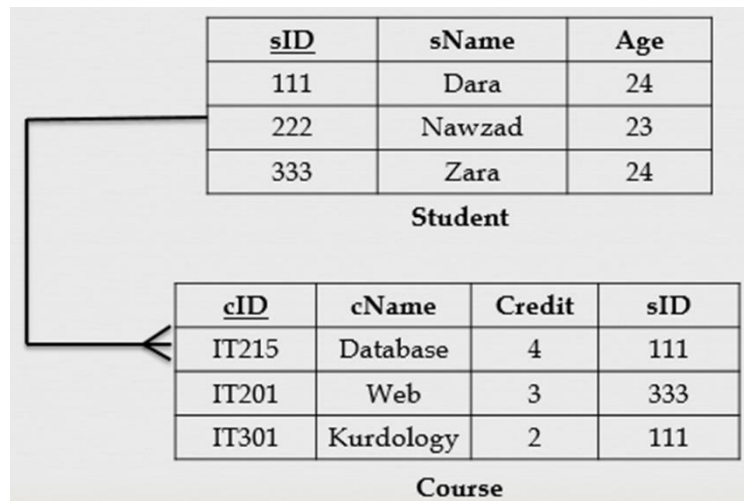


- It is very common to have this kind of relations between your tables.
- This kind of relation is called one to many

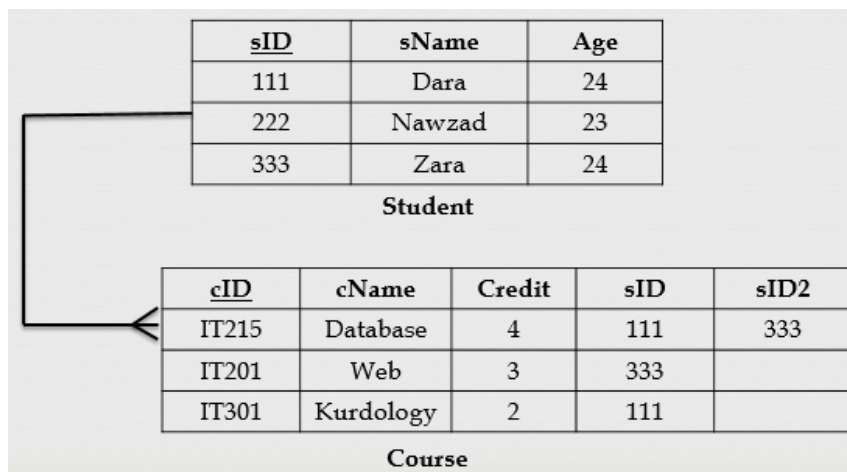
Many-to-Many Relationship:

A *many-to-many relationship* occurs when multiple records in a table are associated with multiple records in another table.

- It is not unusual to sometimes needed to describe a many to many relationship.
- We've got 2 problems here:
- If there are students for one or more course, we will create the relationship by adding sID column to Course Table

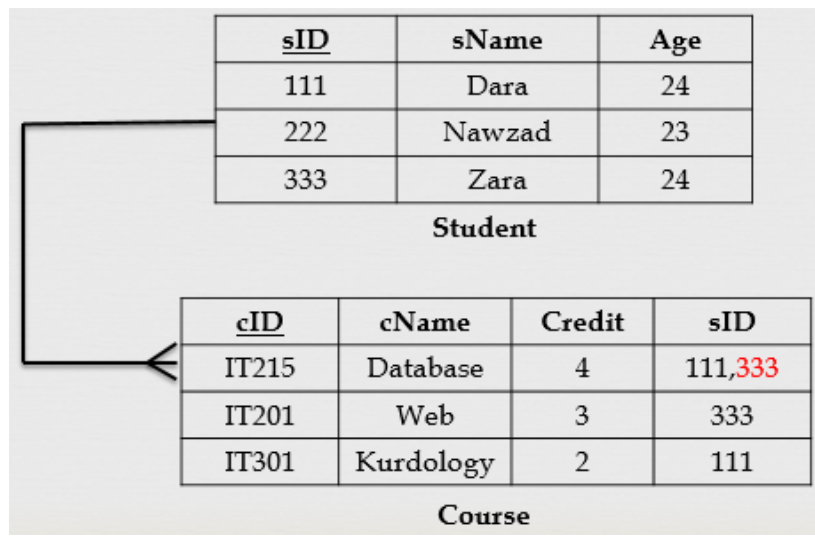


- So far, so good. But here is issue:
- What happens if a course is taken by two or more different students.
- The way we have this describe right now, we can't do that.
- We need a many to many relationship.
- One student can take many courses but also a course can be taken by many students.

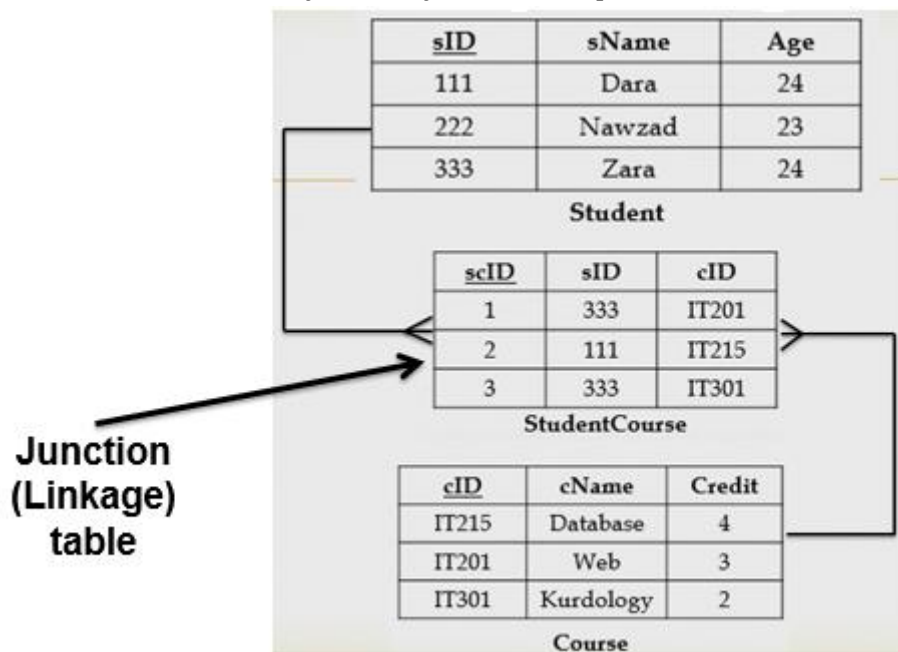


- Some people try this model, adding another column to the Book table.
- However, adding new columns to your table means repeating same information again.
- This is a bad idea and it is discouraged in database design.

- So we'll get rid of that technique.

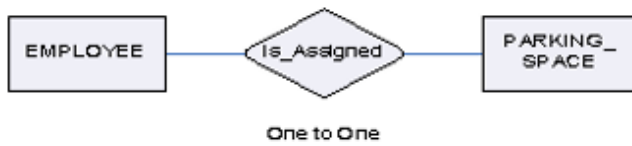


- Some other people think to cheat little bit. Something quick and dirty: Adding two values into that sID column
- Now AuthorID relates two author, But this's also cheat like adding a new column. This is highly discouraged as well.
- So how do we solve this?
- We go back to old tables with no official reference between them.
- What we do to create many to many relationship is we add another table.



- Only reason for this table to exist is to join the Author table and the Book table together.
- We set up two **one to many** relationship.
- By using this we can go from Student to StudentCourse, find a cID and map that to Course table.

- We can also do it the other way
- Officially there is another relationship.
- One to one, but it's not common



Not common



Very common



Occasionally required

Chapter 3: Constraints and Queries

About MS-Access:

Microsoft Access is a Database Management System (DBMS) from Microsoft that combines the relational Microsoft Jet Database Engine with a graphical user interface and software development tools. It is a member of the Microsoft Office suite of applications, included in the professional and higher editions.

MS Access uses "objects" to help the user list and organize information, as well as prepare specially designed reports. When you create a database, Access offers you Tables, Queries, Forms, Reports, Macros, and Modules. Databases in Access are composed of many objects but the following are the major objects –

- Tables
- Queries
- Forms
- Reports

Together, these objects allow you to enter, store, analyze, and compile your data.

MS-Access Data Types:

Every field in a table has properties and these properties define the field's characteristics and behaviour. The most important property for a field is its data type. A field's data type determines what kind of data it can store. MS Access supports different types of data, each with a specific purpose.

- The data type determines the kind of the values that users can store in any given field.
- Each field can store data consisting of only a single data type.

Here are some of the most common data types you will find used in a typical Microsoft Access database.

Type of Data	Description	Size
Short Text	Text or combinations of text and numbers, including numbers that do not require calculating (e.g. phone numbers).	Up to 255 characters.
Long Text	Lengthy text or combinations of text and numbers.	Up to 63, 999 characters.

Number	Numeric data used in mathematical calculations.	1, 2, 4, or 8 bytes (16 bytes if set to Replication ID).
Date/Time	Date and time values for the years 100 through 9999.	8 bytes
Currency	Currency values and numeric data used in mathematical calculations involving data with one to four decimal places.	8 bytes
AutoNumber	A unique sequential (incremented by 1) number or random number assigned by Microsoft Access whenever a new record is added to a table.	4 bytes (16 bytes if set to Replication ID).
Yes/No	Yes and No values and fields that contain only one of two values (Yes/No, True/False, or On/Off).	1 bit.

- If you use previous versions of Access, you will notice a difference for two of those data types.
- In Access 2013, we now have two data types — short text and long text. In previous versions of Access these data types were called text and memo.
- The text field is referred to as short text and your memo field is now called long text.

Here are some of the other more specialized data types, you can choose from in Access.

Data Types	Description	Size
Attachment	Files, such as digital photos. Multiple files can be attached per record. This data type is not available in earlier versions of Access.	Up to about 2 GB.
OLE objects	OLE objects can store pictures, audio, video, or other BLOBs (Binary Large Objects)	Up to about 2 GB.
Hyperlink	Text or combinations of text and numbers stored as text and used as a hyperlink address.	Up to 8,192 (each part of a Hyperlink data type can contain up to 2048 characters).

Lookup Wizard	<p>The Lookup Wizard entry in the Data Type column in the Design view is not actually a data type. When you choose this entry, a wizard starts to help you define either a simple or complex lookup field.</p> <p>A simple lookup field uses the contents of another table or a value list to validate the contents of a single value per row. A complex lookup field allows you to store multiple values of the same data type in each row.</p>	Dependent on the data type of the lookup field.
Calculated	<p>You can create an expression that uses data from one or more fields. You can designate different result data types from the expression.</p>	You can create an expression that uses data from one or more fields. You can designate different result data types from the expression.

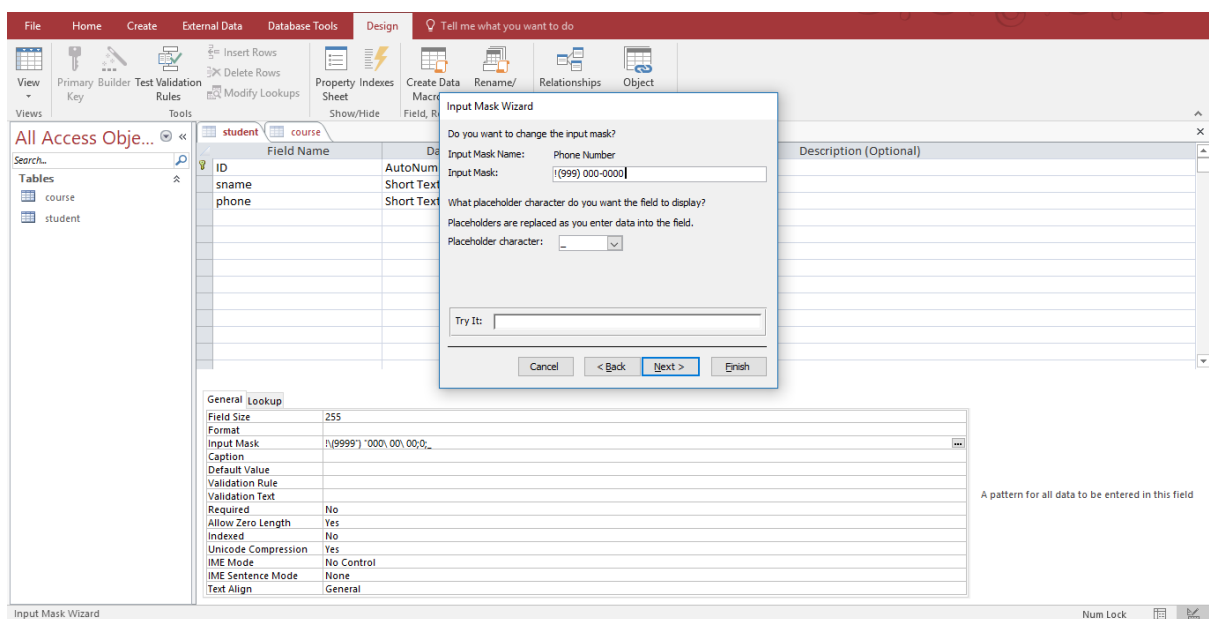
Attribute Constraints and Properties:

Field Property	Description
Field Size	You can specify the size of a field. For Text fields, The maximum number of characters (0 to 255) that can be entered in the field; the default setting is 255. For Number / Currency fields, stores the number as a Byte, Integer, Long Integer, Single, Double, or Replication ID. The default setting is Long Integer.
Format	How the data in the field will be displayed on the screen.
Input Mask	Make a pattern in which data must be entered.
Decimal Places	The number of decimal places in Number and Currency fields.
Caption	A label for the field that will appear on forms. If you do not enter a caption, Access will use the field name as the caption.
Default Value	A value that Access enters automatically in the field for new records.
Validation Rule	An expression that restricts or limits the values that can be entered in the field.
Validation Text	The error message that appears when an incorrect or restricted value is entered in a field with a validation rule.
Required	Specify whether or not a value must be entered in the field. The default is No.
Allow Zero Length	Determines whether or not the field allows zero-length text strings (a string containing no characters). Zero-length text strings are useful if you must enter data in a field, but no data exists. For example, if an ISBN number field requires data, but you do not know the ISBN number of a book, you can enter a zero-length text string in the field. To enter a zero-length text string type "" in the cell. The cell will appear empty. The default is No.
Indexed	Determines whether or not you want to index the field to speed up searches and sorts time. The default is No.

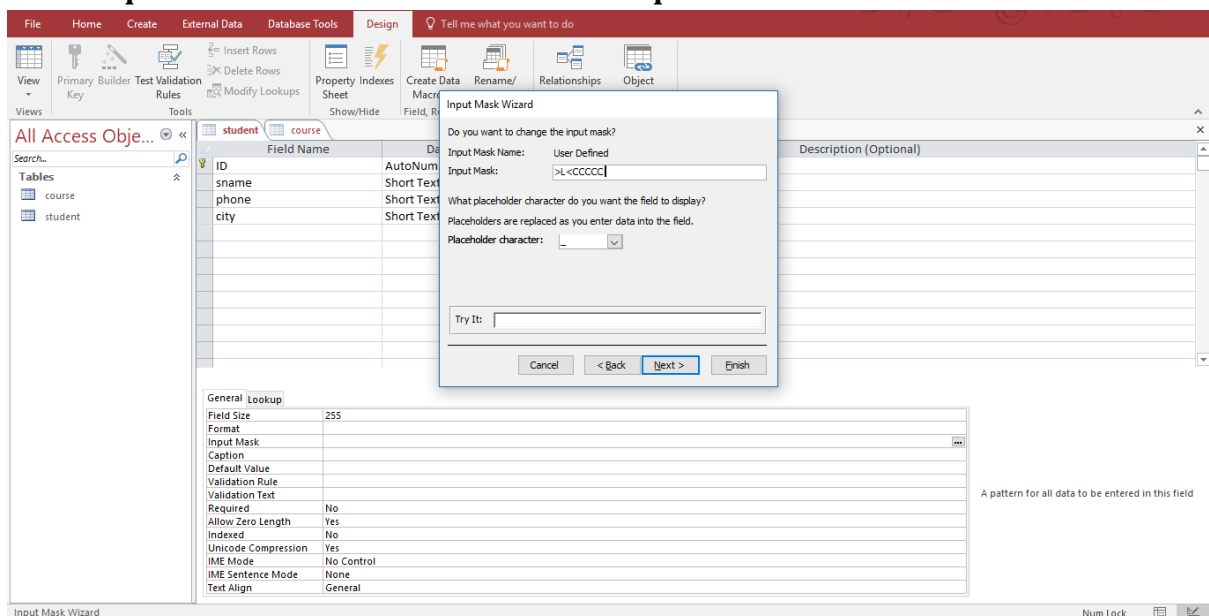
Input Mask:

An input mask is a string of characters that indicates the format of valid input values. You can use input masks in table fields, query fields, and controls on forms and reports. The input mask is stored as an object property.

You use an input mask when it's important that the format of the input values is consistent. For example, you might use an input mask with a field that stores phone numbers so that Access requires ten digits of input. If someone enters a phone number without the area code, Access won't write the data until the area code data is added.



0 for required numbers from 0-9 and 9 for optional numbers from 0-9



L for required characters from A-Z

C or ? for optional numbers from A-Z

Characters that define input masks

The following table lists the placeholder and literal characters for an input mask and explains how it controls data entry:

Some input mask Characters:

Character	Description
0	Digit (0 through 9, entry required; plus [+] and minus [-] signs not allowed).
#	Digit or space (entry not required; blank positions converted to spaces, plus and minus signs allowed).
?	Letter (A through Z, entry optional).
C	Any character or a space (entry optional).
<	Causes all characters that follow to be converted to lowercase.
>	Causes all characters that follow to be converted to uppercase.
!	Causes the input mask to display from right to left, rather than from left to right. Characters typed into the mask always fill it from left to right. You can include the exclamation point anywhere in the input mask.
Password	Password creates a password entry text box. Any character typed in the text box is stored as the character but is displayed as an asterisk (*).

Input Mask Examples:

Input Mask Examples:

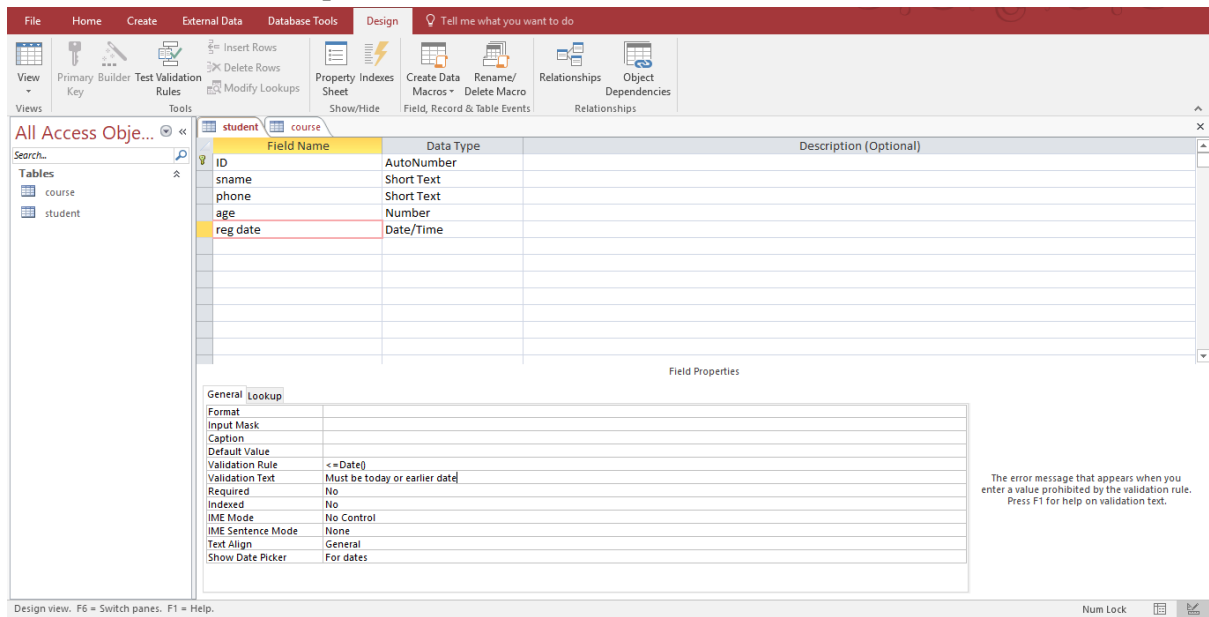
Input mask definition	Examples of values	Input mask definition	Examples of values
(000) 000-0000	(308) 444-0247	(000) AAA-AAAA	(206) 555-TELE
(999) 999-9999!	(308) 444-0247 () 444-0247	#999	-10 1000
>LOL 0L0	K3M 8N3	00000-9999	67223 - 67223-4009
>LL00000-0000	KB71351-0037	ISBN 0-&&&&&&&-0	ISBN 1-55514-607-9 ISBN 0-13-765261-3

Validation Rule:

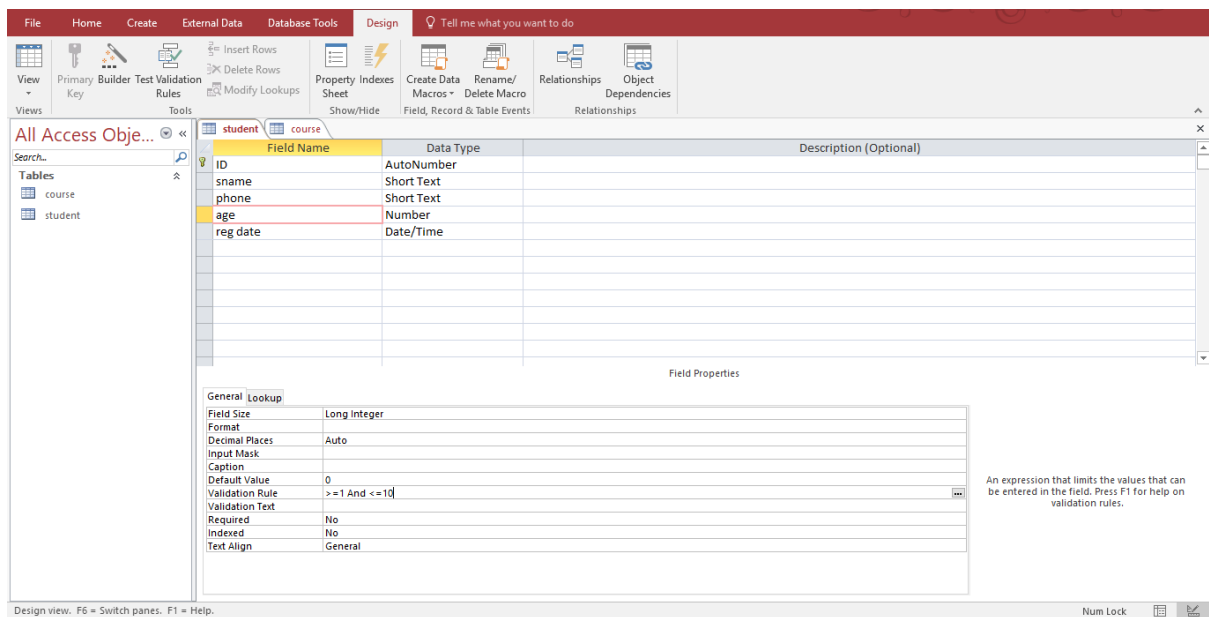
You can vet or validate data in Access desktop databases as you enter it by using validation rules. You can use the expression builder to help you format the rule correctly. Validation rules can be set in either table design or table datasheet view. There are three types of validation rules in Access:

- 1. Field Validation Rule:** You can use a field validation rule to specify a criterion that all valid field values must meet.
- 2. Record Validation Rule:** You can use a record validation rule to specify a condition that all valid records must satisfy.
- 3. Validation on a form:** You can use the **Validation Rule** property of a control on a form to specify a criterion that all values input to that control must meet.

Validation Rule Examples:



<= Date() Today's date or earlier date is acceptable only



>= 1 and <=10 number must be between 1 and 10

Some Validation Rule Examples:

Validation Rule	Description
<>0	Please enter a nonzero value.
1 or >5	Value must be either 1 or over 5.
<#1/1/2010#	Enter a date before 2010.
>=#1/1/2000# and <#1/1/2001#	Date must be in 2000.

Basic Query Structure:

Data Manipulation Language (DML):

The SQL **data-manipulation language (DML)** provides the ability to query information like select, insert, delete and update tuples.

A typical SQL query has the form:

```
select  $A_1, A_2, \dots, A_n$ 
from  $r_1, r_2, \dots, r_m$ 
where  $P$ 
```

A_i represents an attribute

R_i represents a relation

P is a predicate.

The result of an SQL query is a relation.

The Select Clause:

The **select** clause list the attributes desired in the result of a query corresponds to the projection operation of the relational algebra

Example: find the names of all instructors:

```
select name
from instructor
```

NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)

- E.g. *Name* \equiv *NAME* \equiv *name*
- Some people use upper case wherever we use bold font.

Example: find the names of all departments:

```
select dName, Building
from Department
```

or

```
select Department.dName, Department.Building
from Department
```

<u>dID</u>	dName	Building
123	IT	A1
444	Computer	A1
555	Dentistry	B1

Department

Query result

dName	Building
IT	A1
Computer	A1
Dentistry	B1

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the names of all departments with instructor, and remove duplicates

```
select distinct dept_name
from instructor
```

- The keyword **all** specifies that duplicates not be removed.

```
select all dName
from Department
```

- An asterisk in the select clause denotes “all attributes”

```
select *
from instructor
```

- The select clause can contain arithmetic expressions involving the operation, +, -, *, and /, and operating on constants or attributes of tuples.

- The query:

```
select ID, name, salary/12
from instructor
```

- would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.


The Where Clause:

- The where clause specifies conditions that the result must satisfy Corresponds to the selection predicate of the relational algebra.

- To find all Students with Age of 24

```
select ID, sName, Age
from Student
where Age = 24
```

- Comparison results can be combined using the logical connectives and, or, and not.
- Comparisons can be applied to results of arithmetic expressions.



ID	sName	Age	Click to Add
1	Dara	25	
2	Zara	24	
3	Ahmed	26	

Student

ID	sName	Age
2	Zara	24

- The **where** clause specifies conditions that the result must satisfy
 - Corresponds to the selection predicate of the relational algebra.

- To find all instructors in Comp. Sci. dept with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 80000
```

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**.
- Comparisons can be applied to results of arithmetic expressions.

The From Clause:

- The **from** clause lists the relations involved in the query
 - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor* \times *teaches*

```
select *
from instructor, teaches
```

 - generates every possible instructor – teaches pair, with all attributes from both relations
- Cartesian product not very useful directly, but useful combined with where-clause condition

Cartesian Product: *instructor* \times *teaches*:

<i>instructor</i>				<i>teaches</i>				
ID	name	dept_name	salary	ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
32343	El Said	History	60000	15151	MU-199	1	Spring	2010
				22222	PHY-101	1	Fall	2009

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...
...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...
...

Joins:

- For all instructors who have taught some course, find their names and the course ID of the courses they taught.


```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
```
- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

```

select section.course_id, semester, year, title
from section, course
where section.course_id = course.course_id and
      dept_name = 'Comp. Sci.'

```

**Example:**

List the names of instructors along with the course ID of the courses that they taught.

```

select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID;

```

The Rename Operation:

The SQL allows renaming relations and attributes using the **as** clause:

old-name as new-name

E.g.

```

select ID, name, salary/12 as monthly_salary
from instructor

```

Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci.'.

```

select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Comp. Sci.'

```

Keyword **as** is optional and may be omitted

instructor as T \equiv *instructor T*

Keyword **as** must be omitted in Oracle

String Operations:

SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using special characters:

percent (*). The * character matches any substring.

Find the names of all instructors whose name includes the substring "dar".

```

select name
  from instructor
 where name like '*dar*'

```

Kind of match	Pattern	Match	No match
Multiple characters	a*a	aa, aBa, aBBBa	aBC
	ab	abc, AABb, Xab	aZb, bac
Special character	a[*]a	a*a	aaa
Multiple characters	ab*	abcdefg, abc	cab, aab
Single character	a?a	aaa, a3a, aBa	aBBBa
Single digit	a#a	a0a, a1a, a2a	aaa, a10a
Range of characters	[a-z]	f, p, j	2, &

Ordering the Display of Tuples:

List in alphabetic order the names of all instructors

```

select distinct name
  from instructor
 order by name

```

We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.

Example: **order by name desc**

Can sort on multiple attributes

Example: **order by dept_name, name**

Where Clause Predicates:

SQL includes a **between** comparison operator

Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, $\geq \$90,000$ and $\leq \$100,000$)

```

select name
  from instructor
 where salary between 90000 and 100000

```

Null Values:

It is possible for tuples to have a null value, denoted by *null*, for some of their attributes. *null* signifies an unknown value or that a value does not exist.

The predicate **is null** can be used to check for null values.

Example: Find all instructors whose salary is null.

```

select name
from instructor
where salary is null

```

Aggregate Functions:

These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values

Find the average salary of instructors in the Computer Science department

```

select avg (salary)
from instructor
where dept_name= 'Comp. Sci.';

```

Find the total number of instructors who teach a course in the Spring 2010 semester

```

select count (ID)
from teaches
where semester = 'Spring' and year = 2010

```

Find the number of tuples in the *course* relation

```

select count (*)
from course;

```

Find the average salary of instructors in each department

```

select dept_name, avg (salary)
from instructor
group by dept_name;

```

Note: departments with no instructor will not appear in result

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

<i>dept_name</i>	<i>avg_salary</i>
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Null Values and Aggregates:

Total all salaries

```
select sum (salary )  
from instructor
```

- Above statement ignores null amounts
- Result is *null* if there is no non-null amount

All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes

What if collection has only null values?

- count returns 0
- all other aggregates return null

Chapter 4: Queries

Data Definition Language:

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- Creating and editing databases
- Creating and editing tables

Modification of the Database:

- Deletion of tuples from a given relation
- Insertion of new tuples into a given relation
- Updating values in some tuples in a given relation

Modification of the Database – Insertion:

A typical SQL insert query has the form:

insert into r (A_i)
values v_i

- A_i represents an attribute
- r represents a relation
- v is values to be inserted

Examples:

- Add a new tuple to *course*
insert into *department*
values (6, 'IT', 'B1', 24000);
- Add a new tuple to *department* with *building* set to null
insert into *department*
values (7, 'Civil', null, 25000);
- or equivalently
insert into *deptment* (*id*, *dname*, *budget*)
values (7, 'Civil', 25000);

Modification of the Database – Deletion:

- Delete all instructors
delete from *instructor*

- Delete all instructors from the Finance department
delete from *instructor*
 where *dept_name*= 'Finance';

Modification of the Database – Updates:

Increase salaries of instructors whose salary is over \$1000 by 3%, and all others receive a 5% raise

Write two update statements:

```
update instructor
set salary = salary * 1.03
where salary > 1000;
update instructor
set salary = salary * 1.05
where salary <= 1000;
```

The order is important

Referential Integrity:

■ **Cascade Update Related Fields**

- ☒ (Checked) When primary key fields are updated, then foreign key fields will be updated too
- ☐ (Unchecked) When primary key fields are updated, then foreign key fields will not be updated

■ **Cascade Delete Related Records**

- ☒ (Checked) When primary key records are deleted, then all the foreign key records related to it will be deleted too
- ☐ (Unchecked) When primary key records are deleted, then all the foreign key records related to it will not be deleted

The screenshot shows the 'Edit Relationships' dialog box. It has two columns: 'Table/Query:' and 'Related Table/Query:'. Under 'Table/Query:', 'department' is selected. Under 'Related Table/Query:', 'instructor' is selected. Below these, a table shows the relationship between fields: 'ID' (with a dropdown arrow) is linked to 'fkdid'. To the right of this table are buttons: 'OK', 'Cancel', 'Join Type..', and 'Create New..'. Below the table, there are three checked checkboxes: 'Enforce Referential Integrity', 'Cascade Update Related Fields', and 'Cascade Delete Related Records'. At the bottom, 'Relationship Type:' is set to 'One-To-Many'.

Chapter 5: Introduction to SQL

History:

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
 - SQL-86, SQL-89, SQL-92
 - SQL:1999, SQL:2003, SQL:2008
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
 - Not all examples here may work on your particular system.

Data Definition Language:

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- Creating and editing databases
- Creating and editing tables

Domain Types in SQL:

- **char(*n*)**. Fixed length character string, with user-specified length *n*.
- **varchar(*n*)**. Variable length character strings, with user-specified maximum length *n*.
- **int(*n*)** Integer (a finite subset of the integers that is machine-dependent).
- **smallint**. Small integer (a machine-dependent subset of the integer domain type).
- **numeric(*p,d*)** or **decimal(*p,d*)** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.
- **real** or **double**. Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float** Floating point number, with user-specified precision of at least *n* digits.

Type	Minimum Value (Signed)	Maximum Value (Signed)	Minimum Value (Unsigned)	Maximum Value (Unsigned)
TINYINT	-128	127	0	255
SMALLINT	-32768	32767	0	65535
MEDIUMINT	-8388608	8388607 to	0	16777215
INT	-2147483648	2147483647	0	4294967295
BIGINT	- 9223372036854775 808	92233720368 54775807	0	184467440737 09551615

Type	Min Value (Signed)	Max Value (Signed)	Min Value (Unsigned)	Max Value (Unsigned)
FLOAT	-3.402823466E+38	-1.175494351E-38	1.17549435 1E-38	3.402823466 E+38
DOUBLE	-1.7976931348623 157E+308	-2.2250738585072014E- 308	0, and 2.22507385 85072014E- 308	1.797693134 8623157E+ 308

- date: Dates, containing a (4 digit) year, month and date
 - Example: date '2005-7-27'
- time: Time of day, in hours, minutes and seconds.
 - Example: time '09:00:30' time '09:00:30.75'
- datetime: date plus time of day
 - Example: timestamp '2005-7-27 09:00:30.75'

Create Database Construct:

An SQL database is defined using the **create database** command:

create database *d*

- *d* is the name of the database

Example: **create database university;**

An SQL database is activated using the **use** command:

use *d*

- *d* is the name of the database

Example: **use university;**

An SQL database is activated using the **drop database** command:

drop database *d*

- *d* is the name of the database

Example: **drop database university;**

Create Table Construct:

An SQL relation is defined using the **create table** command:

```
create table r (A1 D1, A2 D2, ..., An Dn,
               (integrity-constraint1),
               ...,
               (integrity-constraintk))
```

- *r* is the name of the relation
- each *A_i* is an attribute name in the schema of relation *r*
- *D_i* is the data type of values in the domain of attribute *A_i*

Example:

```
create table instructor (
    ID          char(5),
    name        varchar(20) not null,
    dept_name   varchar(20),
    salary      numeric(8,2))
```

```
insert into instructor values ('10211', 'Smith', 'Biology', 66000);
```

```
insert into instructor values ('10211', null, 'Biology', 66000);
```

Integrity Constraints in Create Table:

- **not null**
- **primary key** (*A*₁, ..., *A_n*)
- **foreign key** (*A_m*, ..., *A_n*) **references** *r* (*A*)

Example: Declare *dept_name* as the primary key for *department*

```
create table instructor (
    ID          char(5),
    name        varchar(20) not null,
    dept_name   varchar(20),
    salary      numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department (dname)
);
```

primary key declaration on an attribute automatically ensures **not null**.

Drop and Alter Table Constructs:

- **drop table** *student*
 - Deletes the table and its contents
- **delete from** *student*
 - Deletes all contents of table, but retains table
- **alter table**
 - alter table *r* add *A D*

- where A is the name of the attribute to be added to relation r and D is the domain of A .
- All tuples in the relation are assigned *null* as the value for the new attribute.
- alter table r drop A
 - where A is the name of an attribute of relation r
 - Dropping of attributes not supported by many databases
- **ALTER TABLE** `tableName` **CHANGE COLUMN** `oldColumnName`
`newColumnName` **DATA TYPE**
- ALTER TABLE `department` CHANGE COLUMN `Building` `Buildings` CHAR(2)

Basic Query Structure:

- The SQL **data-manipulation language (DML)** provides the ability to query information, and insert, delete and update tuples
- A typical SQL query has the form:

select A_1, A_2, \dots, A_n
from r_1, r_2, \dots, r_m
where P

- A_i represents an attribute
- R_i represents a relation
- P is a predicate.
- The result of an SQL query is a relation.

Modification of the Database:

- Deletion of tuples from a given relation
- Insertion of new tuples into a given relation
- Updating values in some tuples in a given relation

Modification of the Database – Insertion:

A typical SQL insert query has the form:

insert into r (A_i)
values v_i

- A_i represents an attribute
- r represents a relation
- v is values to be inserted

Examples:

- Add a new tuple to *course*

insert into *department*
values (6, 'IT', 'B1', 24000);

- Add a new tuple to *department* with *building* set to null

insert into *department*
values (7, 'Civil', null, 25000);

- or equivalently

insert into *deptment* (*id*, *dname*, *budget*)
values (7, 'Civil', 25000);

Modification of the Database – Deletion:

- Delete all instructors

delete from *instructor*

- Delete all instructors from the Finance department

delete from *instructor*
 where *dept_name*= 'Finance';

Modification of the Database – Updates:

Increase salaries of instructors whose salary is over \$1000 by 3%, and all others receive a 5% raise

Write two update statements:

update *instructor*
set *salary* = *salary* * 1.03
 where *salary* > 1000;
update *instructor*
set *salary* = *salary* * 1.05
 where *salary* <= 1000;

The order is important

The Select Clause:

- The **select** clause list the attributes desired in the result of a query
 - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

select *name*
from *instructor*
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
 - E.g. *Name* \equiv *NAME* \equiv *name*
 - Some people use upper case wherever we use bold font.
- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.

- Find the names of all departments with instructor, and remove duplicates

```
select distinct dept_name
from instructor
```
- The keyword **all** specifies that duplicates not be removed.

```
select all dept_name
from instructor
```
- An asterisk (*) in the select clause denotes “all attributes”

```
select *
from instructor
```
- The select clause can contain arithmetic expressions involving the operation, +, -, *, and /, and operating on constants or attributes of tuples.
- The query:

```
select ID, name, salary/12
from instructor
```

would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.

The Where Clause:

- The **where** clause specifies conditions that the result must satisfy
 - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 80000
```
- Comparison results can be combined using the logical connectives and, or, and not.
- Comparisons can be applied to results of arithmetic expressions.

The From Clause:

- The from clause lists the relations involved in the query
 - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor* × *teaches*

```
select *
from instructor, teaches
```

 - generates every possible instructor – teaches pair, with all attributes from both relations
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

Cartesian Product: *instructor* × *teaches*

<i>instructor</i>				<i>teaches</i>				
ID	name	dept_name	salary	ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
32343	El Said	History	60000	15151	MU-199	1	Spring	2010
22222	PHY-101			22222	PHY-101	1	Fall	2009

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...
...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...
...

Joins:

- For all instructors who have taught some course, find their names and the course ID of the courses they taught.
select *name, course_id*
from *instructor, teaches*
where *instructor.ID = teaches.ID*
- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department
select *section.course_id, semester, year, title*
from *section, course*
where *section.course_id = course.course_id* and *dept_name = 'Comp. Sci.'*

Natural Join:

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
select *
from *instructor natural join teaches;*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

- List the names of instructors along with the course ID of the courses that they taught.

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID;
```

```
select name, course_id
from instructor natural join teaches;
```

The Rename Operation:

- The SQL allows renaming relations and attributes using the as clause:
old-name as new-name
- E.g.

```
select ID, name, salary/12 as monthly_salary
from instructor
```
- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
–

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Comp. Sci.'
```
- Keyword **as** is optional and may be omitted
instructor as T \equiv *instructor T*

String Operations:

- SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters:
 - percent (%)**. The % character matches any substring.
 - underscore (_)**. The _ character matches any character.

- Find the names of all instructors whose name includes the substring “dar”.

```
select name
from instructor
where name like '%dar%'
```
- Match the string “100 %”

```
like '100 \%' escape '\'
```
- Patterns are case sensitive.
- Pattern matching examples:
 - ‘Intro%’ matches any string beginning with “Intro”.
 - ‘%Comp%’ matches any string containing “Comp” as a substring.
 - ‘___’ matches any string of exactly three characters.
 - ‘___%’ matches any string of at least three characters.
- SQL supports a variety of string operations such as

```
select concat(fname,' ',lname) as fullname, age
from student
```

Ordering the Display of Tuples:

- List in alphabetic order the names of all instructors

```
select distinct name
from instructor
order by name
```
- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
 - Example: order by *name* desc
- Can sort on multiple attributes
 - Example: order by *dept_name*, *name*

Where Clause Predicates:

- SQL includes a between comparison operator
- Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, >=\$90,000 and <=\$100,000)
 - ```
select name
from instructor
where salary between 90000 and 100000
```
- Tuple comparison
  - ```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

Aggregate Functions:

These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values

Find the average salary of instructors in the Computer Science department

```
select avg (salary)
from instructor
where dept_name= 'Comp. Sci.';
```

Find the total number of instructors who teach a course in the Spring 2010 semester

```
select count (ID)
from teaches
where semester = 'Spring' and year = 2010
```

Find the number of tuples in the *course* relation

```
select count (*)
from course;
```

Find the average salary of instructors in each department

```
select dept_name, avg (salary)
from instructor
group by dept_name;
```

Note: departments with no instructor will not appear in result

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

<i>dept_name</i>	<i>avg_salary</i>
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Null Values and Aggregates:

Total all salaries

```
select sum (salary )  
from instructor
```

- Above statement ignores null amounts
- Result is *null* if there is no non-null amount

All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes

What if collection has only null values?

- count returns 0
- all other aggregates return null

Chapter 6: Entity-Relationship Model

Modeling:

A *database* can be modeled as:

- a collection of entities,
- relationship among entities.

An **entity** is an object that exists and is distinguishable from other objects.

Example: specific person, company, event, plant

Entities have **attributes**

Example: people have *names* and *addresses*

An **entity set** is a set of entities of the same type that share the same properties.

Example: set of all persons, companies, trees, holidays

Entity Sets *instructor* and *student*

instructor_ID	instructor_name	student-ID	student_name
76766	Crick	98988	Tanaka
45565	Katz	12345	Shankar
10101	Srinivasan	00128	Zhang
98345	Kim	76543	Brown
76543	Singh	76653	Aoi
22222	Einstein	23121	Chavez
		44553	Peltier

instructor *student*

Relationship Sets:

A **relationship** is an association among several entities

Example:

44553 (<u>Peltier</u>)	<u>advisor</u>	22222 (<u>Einstein</u>)
<i>student</i> entity	relationship set	<i>instructor</i> entity

A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

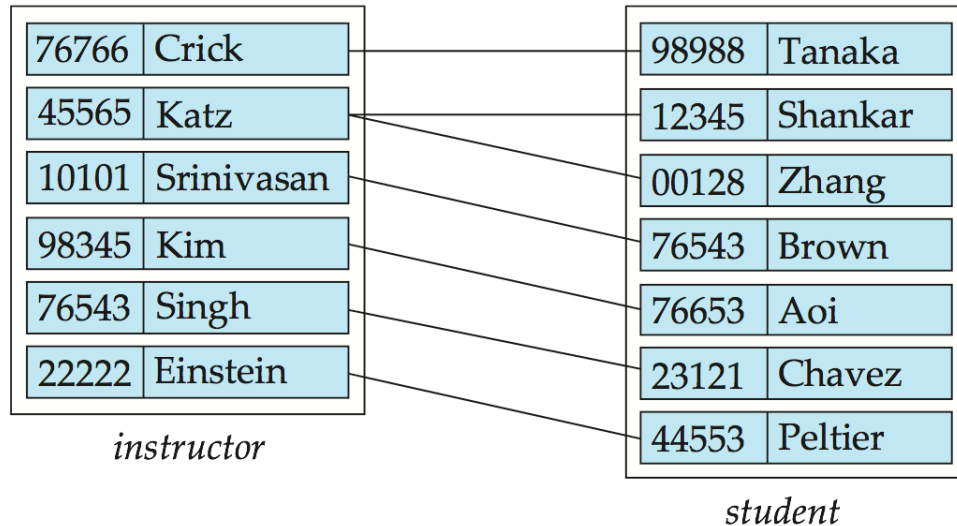
$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

where (e_1, e_2, \dots, e_n) is a relationship

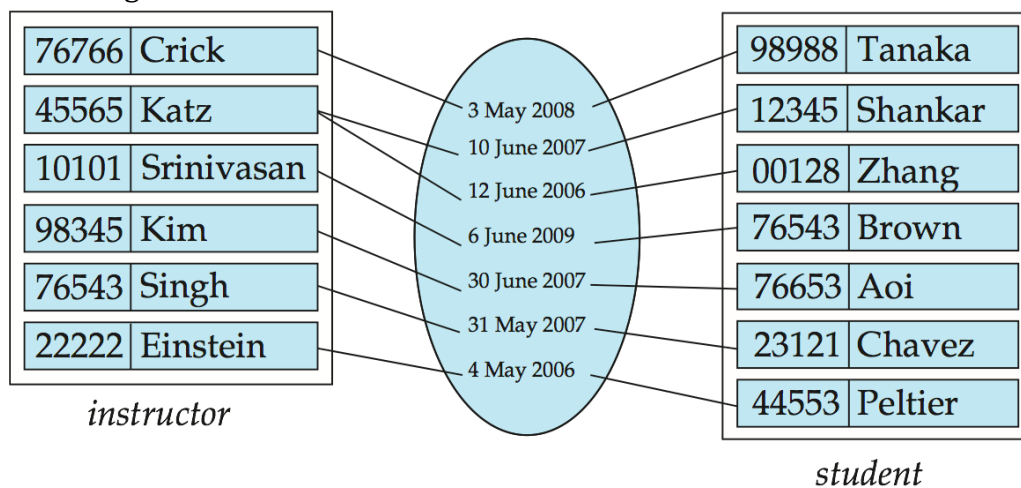
Example:

$(44553, 22222) \in \text{advisor}$

Relationship Set *advisor*



- An **attribute** can also be property of a relationship set.
- For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor

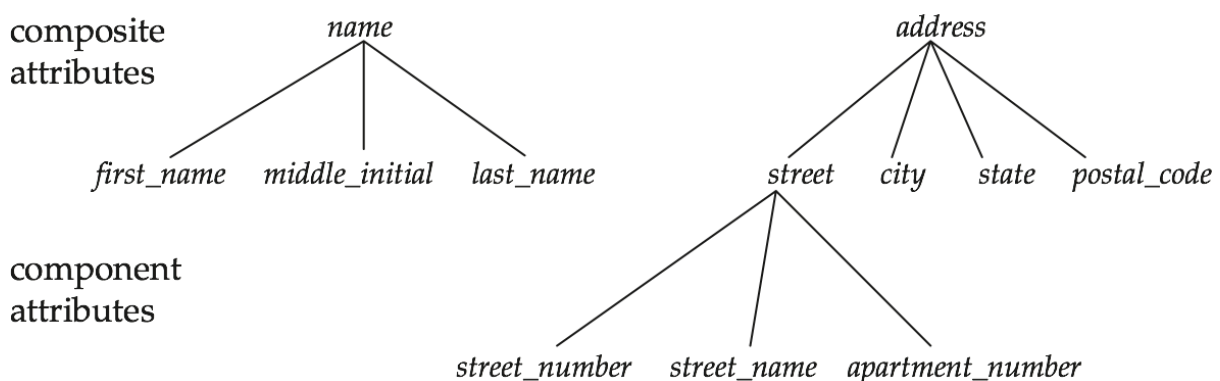


Degree of a Relationship Set:

- **binary relationship**
 - involve two entity sets (or degree two).
 - most relationship sets in a database system are binary.
- Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)
 - Example: *students* work on research *projects* under the guidance of an *instructor*.
 - relationship *proj_guide* is a ternary relationship between *instructor*, *student*, and *project*

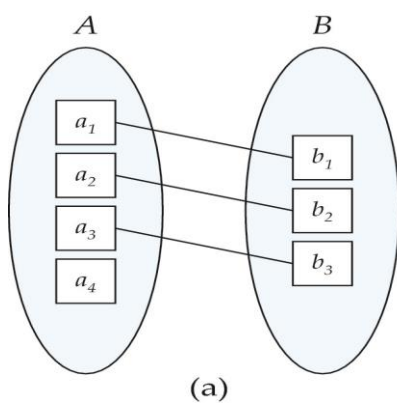
Attributes:

- An entity is represented by a set of attributes, which is descriptive properties possessed by all members of an entity set.
 - Example:
 - *instructor* = (*ID*, *name*, *street*, *city*, *salary*)
 - *course* = (*course_id*, *title*, *credits*)
- **Domain** – the set of permitted values for each attribute
- Attribute types:
 - **Simple** and **composite** attributes.
 - **Single-valued** and **multivalued** attributes
 - Example: multivalued attribute: *phone_numbers*
 - **Derived** attributes
 - Can be computed from other attributes
 - Example: *age*, given *date_of_birth*

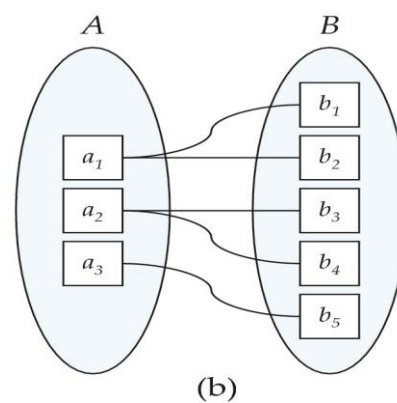
Composite Attributes

Mapping Cardinality Constraints:

- It express the number of entities to which another entity can be associated via a relationship set.
- It is most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
 - One to one
 - One to many
 - Many to one
 - Many to many

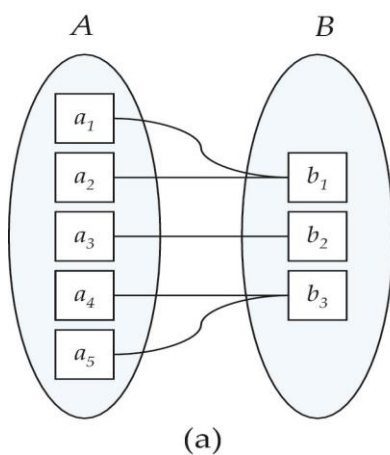


One to one

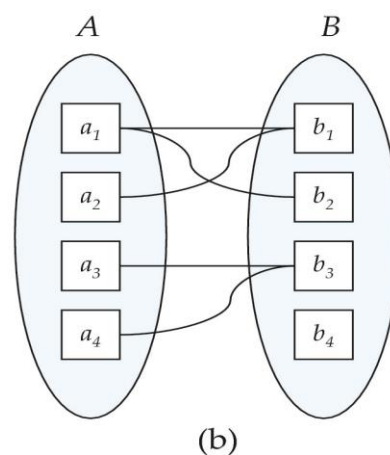


One to many

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set



Many to one



Many to many

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

Keys:

- A **super key** of an entity set is a set of one or more attributes whose values uniquely determine each entity.
- A **candidate key** of an entity set is a minimal super key
 - *ID* is candidate key of *instructor*
 - *course_id* is candidate key of *course*
- Although several candidate keys may exist, one of the candidate keys is selected to be the **primary key**.
- The combination of primary keys of the participating entity sets forms a super key of a relationship set.
 - (*s_id, i_id*) is the super key of *advisor*
 - **NOTE: this means a pair of entity sets can have at most one relationship in a particular relationship set.**
 - Example: if we wish to track multiple meeting dates between a student and her advisor, we cannot assume a relationship for each meeting. We can use a multivalued attribute though
- Must consider the mapping cardinality of the relationship set when deciding what are the candidate keys
- Need to consider semantics of relationship set in selecting the *primary key* in case of more than one candidate key

Redundant Attributes:

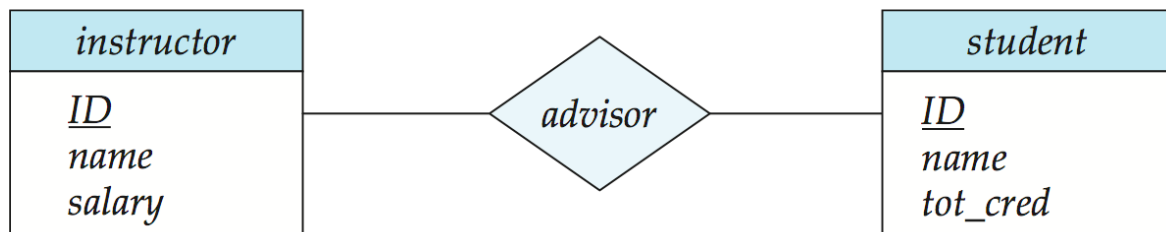
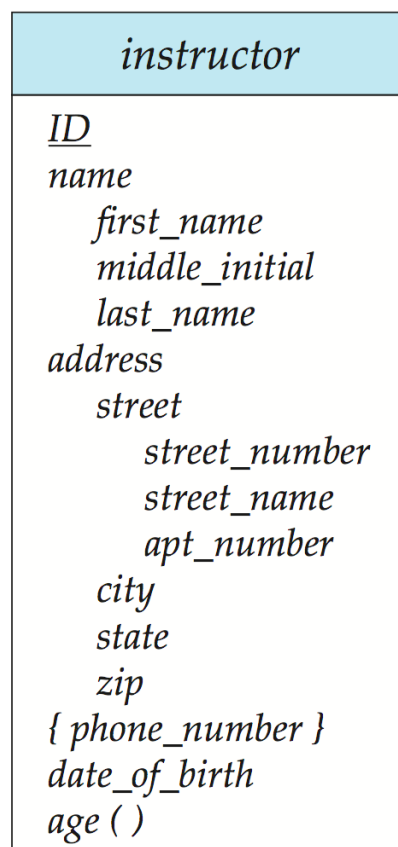
- Suppose we have entity sets
 - *instructor*, with attributes including *dept_name*
 - *department*
- and a relationship
 - *inst_dept* relating *instructor* and *department*
- Attribute *dept_name* in entity *instructor* is redundant since there is an explicit relationship *inst_dept* which relates instructors to departments
 - The attribute replicates information present in the relationship, and should be removed from *instructor*
 - BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see.

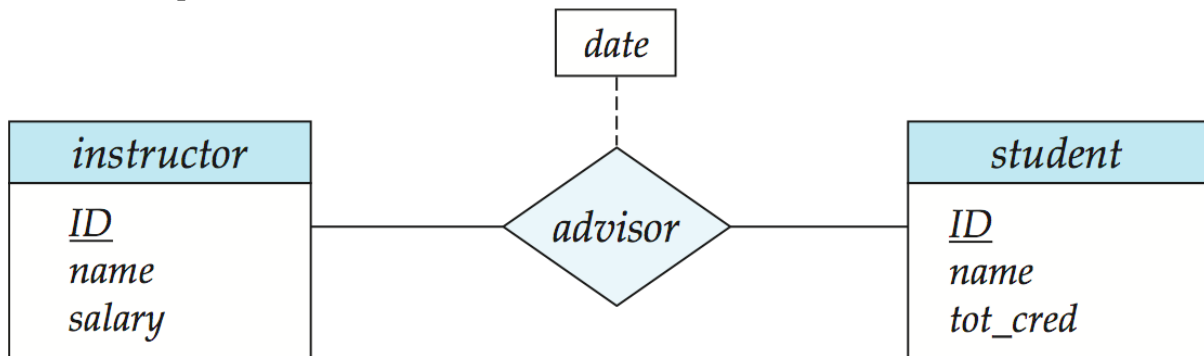
E-R Diagrams:

Entity-Relationship (E-R) diagram is a graphical/ pictorial representations of Entities and their Relationship.

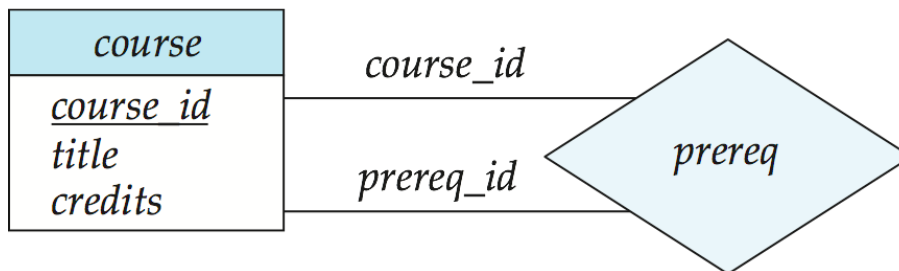
Following symbols were used to represent/ draw E-R Diagram:

- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Attributes listed inside entity rectangle
- Underline indicates primary key attributes

E-R Diagram showing Binary Relationship:**Entity With Composite, Multivalued, and Derived Attributes:**

Relationship Sets with Attributes:**Roles:**

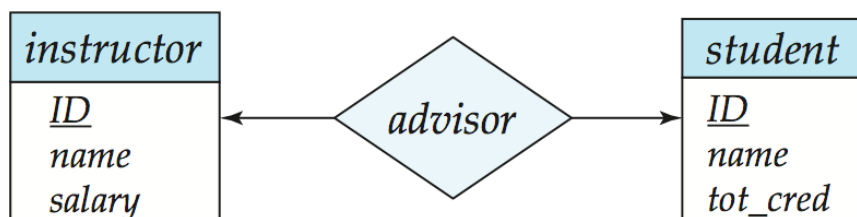
- Entity sets of a relationship need not be distinct
 - Each occurrence of an entity set plays a “role” in the relationship
- The labels “*course_id*” and “*prereq_id*” are called **roles**.

**Cardinality Constraints:**

- We express cardinality constraints by drawing either a directed line (\rightarrow), signifying “one,” or an undirected line ($-$), signifying “many,” between the relationship set and the entity set.
- One-to-one relationship:
 - A *student* is associated with at most one *instructor* via the relationship *advisor*
 - A *student* is associated with at most one *department* via *stud_dept*

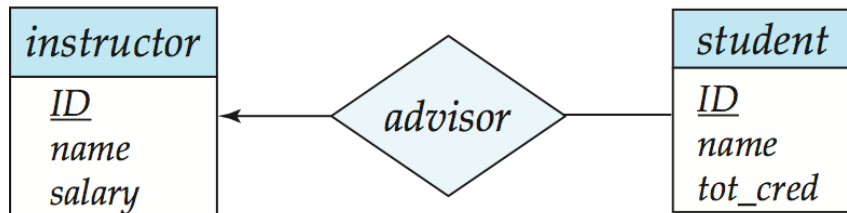
One-to-One Relationship:

- one-to-one relationship between an *instructor* and a *student*
 - an *instructor* is associated with at most one *student* via *advisor*
 - and a *student* is associated with at most one *instructor* via *advisor*

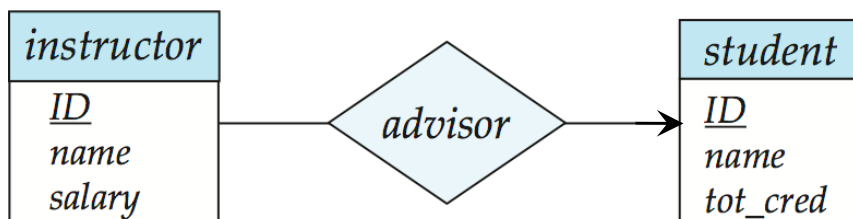


One-to-Many Relationship:

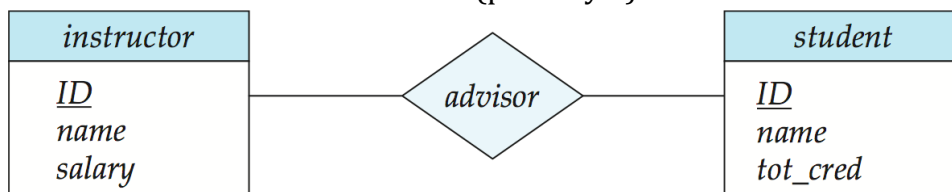
- one-to-many relationship between an *instructor* and a *student*
 - an instructor is associated with several (including 0) students via *advisor*
 - a student is associated with at most one instructor via *advisor*

**Many-to-One Relationships:**

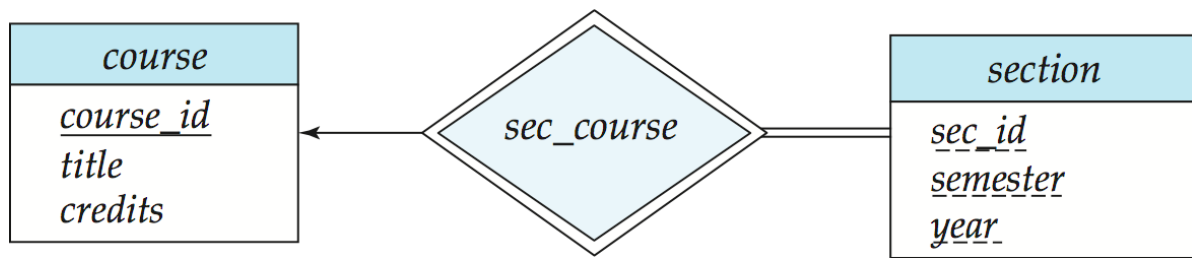
- In a many-to-one relationship between an *instructor* and a *student*,
 - an instructor is associated with at most one student via *advisor*,
 - and a student is associated with several (including 0) instructors via *advisor*

**Many-to-Many Relationship:**

- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*

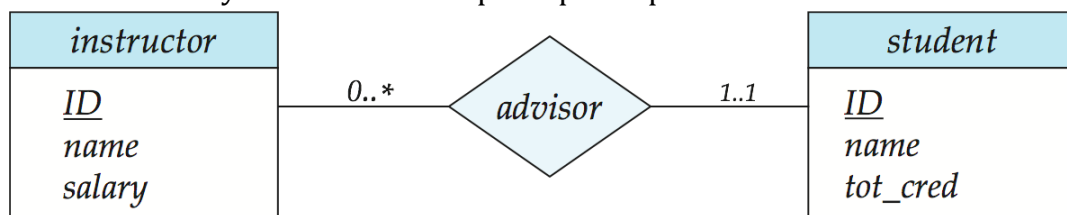
**Participation of an Entity Set in a Relationship Set:**

- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
 - E.g., participation of *section* in *sec_course* is total
 - every *section* must have an associated course
- Partial participation: some entities may not participate in any relationship in the relationship set
 - Example: participation of *instructor* in *advisor* is partial

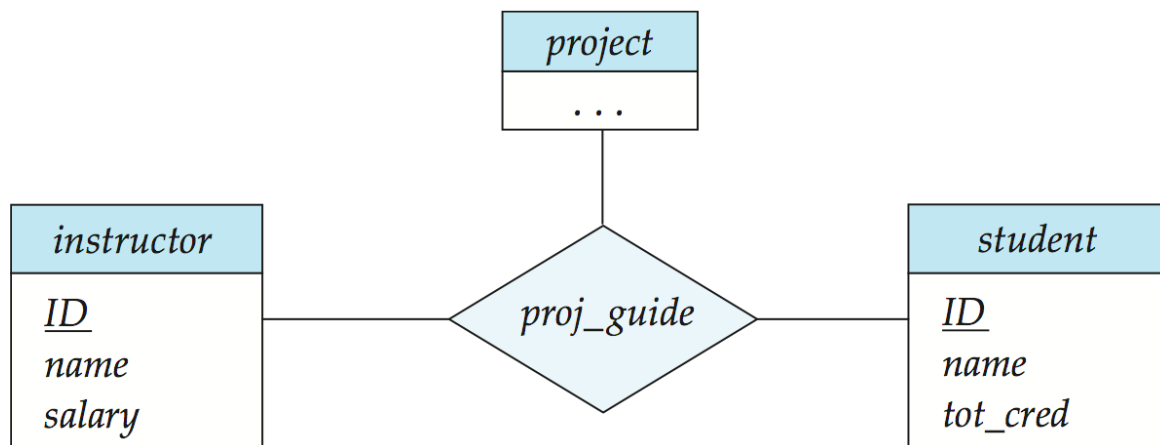


Alternative Notation for Cardinality Limits:

- Cardinality limits can also express participation constraints



E-R Diagram with a Ternary Relationship:

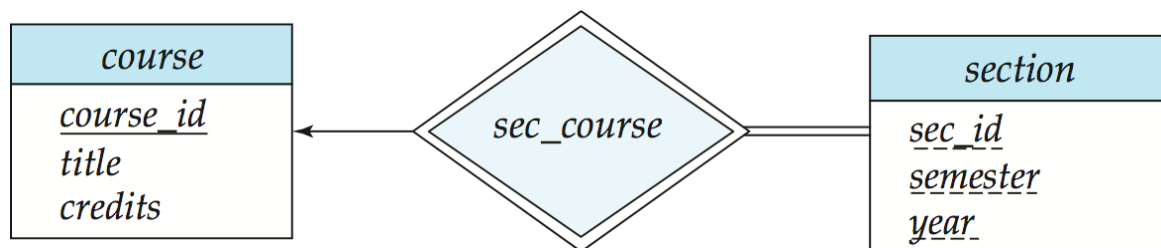


Cardinality Constraints on Ternary Relationship:

- We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint
- E.g., an arrow from *proj_guide* to *instructor* indicates each student has at most one guide for a project
- If there is more than one arrow, there are two ways of defining the meaning.
 - E.g., a ternary relationship *R* between *A*, *B* and *C* with arrows to *B* and *C* could mean
 - each *A* entity is associated with a unique entity from *B* and *C* or
 - each pair of entities from (*A*, *B*) is associated with a unique *C* entity, and each pair (*A*, *C*) is associated with a unique *B*
 - Each alternative has been used in different formalisms
 - To avoid confusion we outlaw more than one arrow

Weak Entity Sets:

- An entity set that does not have a primary key is referred to as a **weak entity set**.
- The existence of a weak entity set depends on the existence of a **identifying entity set**
 - It must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
 - **Identifying relationship** depicted using a double diamond
- The **discriminator** (or *partial key*) of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.
- We underline the discriminator of a weak entity set with a dashed line.
- We put the identifying relationship of a weak entity in a double diamond.
- Primary key for *section* – (*course_id*, *sec_id*, *semester*, *year*)



- Note: the primary key of the strong entity set is not explicitly stored with the weak entity set, since it is implicit in the identifying relationship.
- If *course_id* were explicitly stored, *section* could be made a strong entity, but then the relationship between *section* and *course* would be duplicated by an implicit relationship defined by the attribute *course_id* common to *course* and *section*

E-R Diagram for a University Enterprise: