Database systems

Introduction and overview

MS in Information Technology
Alua Baurzhanovna

This Lecture

- How to contact me
- Module material
- Reference book
- Lectures and assessment
- Module overview
  - The Relational Model
    - Relational data structures
  - Relational algebra
    - Union, Intersection and Difference
    - Product of Relations
    - Projection, Selection

About me:

- Higher Education :
  - 2009-2013: International Information Technology University (Almaty, Kazakhstan)
  - 2014 – Newcastle University (Newcastle , UK)
  - 2014-2015 – The University of Nottingham (Nottingham , UK)
- Work IITU since 2015

How to contact me

- Before/ after lectures
- In the lab

- **Office 802 is NOT an option!**
- By email
  - alua.ospan@gmail.com

Module material

- dl.iitu.kz (Look for Database)
- Slides for every session will be available
- A number of texts in Library – Database Systems - A Practical Approach to Design, Implementation, and Management, Connolly & Begg (source of some diagrams) – Fundamentals of Database Systems

Course policy

Students are forbidden to:

- submit any tasks after the deadline. Late submissions are graded down **(10% per day)**.
- cheat. Plagiarized papers shall not be graded (ZERO);
- be late for classes. Being tardy three times amounts to one absence;
- retake any tests, unless there is a valid reason for missing them;
- use mobile phones in class;

Students should always

- be appropriately dressed (formal/semi- formal styles are acceptable);
- let the teacher know of any problems arising in connection with their studies

Lectures and Assessments

- Lecture – once a week
- Lab sessions – three times a week
- Assessments for whole semester

Term 1	
Practical Lesson: 6 Quizzes = 30% (each 5 %) Starting from Week 2 to 6	
6 Lab works = 42% (each 7%) **Deadline is the end of every week**	
28 % Mid Term	

Learning and feedback

- **Lectures and lab sessions are extremely important**
  - Not everything I say is in a book!
  - I expect you to attend all sessions and take notes
- **Coursework feedback will be given before the exam**
  - In person during last lab session
  - _ If you will not submit the coursework, you will not be able to pass the module. SORRY)

What is Database

- **“A collection of data arranged for ease and speed of search and retrieval.”**
- – American Heritage Science Dictionary
- **• “A structured set of data held in computer storage”**
- – Oxford English Dictionary
- **• “One or more large structured sets of persistent data, usually associated with software to update and query the data”**
- – Free On-Line Dictionary of Computing

Why we study database?

- Databases are important for computing
- – Many computing applications deal with large amounts of information
- – Database systems give a set of tools for storing, searching and managing this information
- **• Databases are a ‘core topic’ in computer science and IT**
- • Basic concepts and skills with database systems are part of the skill set you will be assumed to have as a CS and IT graduate

Databases are (virtually) everywhere!

- • Library catalogues
- • Medical records
- • Bank accounts
- • Stock market data
- • Personnel systems
- • Product catalogues

- Telephone directories
- Train timetables
- Airline bookings
- Credit card details
- Student records
- Customer histories
- Stock market prices
- and many more...

Example of modern database

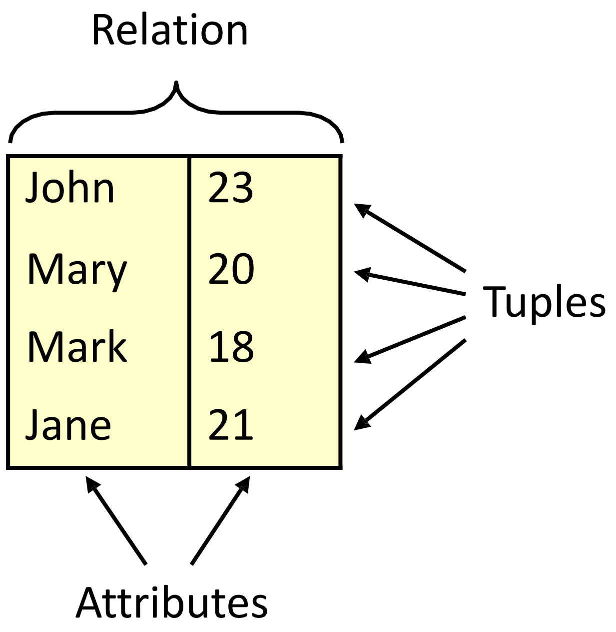
- • Database Management System (DBMS) – The software that implements a database • Examples:
- – Oracle
- – DB2
- – MySQL
- – Ingres
- – PostgreSQL
- – Microsoft SQL Server
- – [MS Access?]

Relational algebra

- first described by E.F. Codd while at IBM, is a family of algebras with a well-founded semantics used for modeling the data stored in **relational databases**, and defining queries on it.

Relational Data Structure

- Data is stored in
- *relations* (tables)
- Relations are made up of *attributes* (columns)
- Data takes the form of
- *tuples* (rows)
  - The order of tuples is not important
    - There must not be
    - duplicate tuples



Relations

- We will use tables to represent relations
- This is an example relation between people and email addresses:

Andrew	aaa@cs.nott.ac.uk
Bill	bbb@cs.nott.ac.uk
Christine	ccc@cs.nott.ac.uk

Relations

- In general, each column has a *domain*, a set from which all possible values for that column can come
- For example, each value in the first column below comes from the set of first names

Andrew	aaa@cs.nott.ac.uk
Bill	bbb@cs.nott.ac.uk
Christine	ccc@cs.nott.ac.uk

Relations

- A mathematical relation is a set of tuples: sequences of values. Each tuple represents a row in the table:

Andrew	aaa@cs.nott.ac.uk	0115 911 1111
Bill	bbb@cs.nott.ac.uk	0115 922 2222
Christine	ccc@cs.nott.ac.uk	0115 933 3333

- {<Andrew, aaa@cs.nott.ac.uk, 01159111111>,
  <Bill, bbb@cs.nott.ac.uk, 01159222222>,
  <Christine, ccc@cs.nott.ac.uk, 01159333333>}

Terminology

- **Degree of a relation**: how long each tuple is, or how many columns the table has
  - In the first example (name, email), the degree of the relation is 2
  - In the second example (name, email, phone) the degree of the relation is 3
  - Degrees of 2, 3, ... are often called Binary, Ternary, etc.
- **Cardinality of a relation**: how many different tuples there are, or how many rows a table has

Mathematical Definition

- The mathematical definition of a relation R of degree n, where values come from domains A_1, ..., A_n:

  $R \subseteq A_1 \times A_2 \times \ldots \times A_n$

  (a relation is a subset of the Cartesian product of domains)

  Cartesian product:

  $A_1 \times A_2 \times \ldots \times A_n =$

  $\{<a_1, a_2, \ldots, a_n>: a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n\}$

Data Manipulation

- Data is represented as relations
- Manipulation of this data (through updates and queries) corresponds to operations on relations
- Relational algebra describes those operations. These take relations as arguments, and produce new relations
- Relational algebra contains two types of operators. Common, set-theoretic operators and those specific to relations

Union

- Standard set-theoretic definition of union:
  $A \cup B = \{x: x \in A \text{ or } x \in B\}$
- For example, $\{a,b,c\} \cup \{a,d,e\} = \{a,b,c,d,e\}$
- For relations, we require the results to be in the form of another relation.
- In order to take a union of relations R and S, R and S must have the same number of columns and corresponding columns must have the same domains

Union-compatible Relations

- Two relations R and S are **union-compatible** if:
  - They have the same number of columns
  - Corresponding columns have the same domains

Example 1: Union-compatible?

Andrew	1970
Bill	1971
Christine	1972

Tom	1980
Sam	1985
Steve	1986

YES!

Same number of columns and matching domains

Example 2: Union-compatible?

Andrew	1970	NG7
Bill	1971	NG16
Christine	1972	NG21

Tom	1980
Sam	1985
Steve	1986

NO!

Different numbers of columns

Example 3: Union-compatible?

Andrew	NG7
Bill	NG16
Christine	NG21

Tom	1980
Sam	1985
Steve	1986

NO!

Corresponding columns have different domains

Unions of Relations

- Let R and S be two union-compatible relations. The Union $R \cup S$ is a relation containing all tuples from both relations:

  $R \cup S = \{x: x \in R \text{ or } x \in S\}$

- Note that union is a partial operation on relations. That is, it is only defined for some (compatible) relations
- This is similar in principle to division of numbers. Division by zero is undefined

Union Example

R

Cheese	1.34
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00

S

Cream	2.00
Soap	1.00

R ∪ S

Cheese	1.34
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00
Cream	2.00

Difference of Relations

- Let R and S be two union-compatible relations. The **difference** R - S is a relation containing all tuples from R that are not in S:

  $R - S = \{x: x \in R \text{ and } x \notin S\}$

- This is also a partial operation on relations

Difference Example

R	
Cheese	1.34
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00

S	
Cream	2.00
Soap	1.00

R - S	
Cheese	1.34
Milk	0.80
Bread	0.60
Eggs	1.20

Intersection of Relations

- Let R and S be two union-compatible relations. The **intersection** $R \cap S$ is a relation containing all tuples that are in both R and S:

  $R \cap S = \{x: x \in R \text{ and } x \in S\}$

- This is also a partial operation on relations

Intersection Example

R	
Cheese	1.34
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00

S	
Cream	2.00
Soap	1.00

R ∩ S	
Soap	1.00

Cartesian Product

- Cartesian product is a total operation on relations.
  - Can be applied to relations of any relative size
- Set-theoretic definition of product:

  $R \times S = \{<x, y>: x \in R, y \in S\}$
- For example, if <Cheese, 1.34> ∈ R and <Soap, 1.00> ∈ S then

  <<Cheese,1.34>,<Soap,1.00>> ∈ R x S

Extended Cartesian Product

- **Extended** Cartesian product flattens the result into a single tuple. For example:

  <Cheese, 1.34, Soap, 1.00>

- This is more useful for relational databases
- For the rest of this module, “product” will mean extended Cartesian product

Extended Cartesian Product of Relations

- Let R be a relation with column domains $\{A_1,\ldots,A_n\}$ and S a relation with column domains $\{B_1,\ldots,B_m\}$. Their extended Cartesian product R x S is a relation:

$$R \times S = \{\langle c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}\rangle : \langle c_1, \ldots, c_n\rangle \in R, \langle c_{n+1}, \ldots, c_{n+m}\rangle \in S\}$$

Product Example

R

Cheese	1.34
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00

S

Cream	2.00
Soap	1.00

R x S

Cheese	1.34	Cream	2.00
Milk	0.80	Cream	2.00
Bread	0.60	Cream	2.00
Eggs	1.20	Cream	2.00
Soap	1.00	Cream	2.00
Cheese	1.34	Soap	1.00
Milk	0.80	Soap	1.00
Bread	0.60	Soap	1.00
Eggs	1.20	Soap	1.00
Soap	1.00	Soap	1.00

Projection

- Sometimes using all columns in a relation is unnecessary
- Let R be a relation with n columns, and X be a set of column identifiers. The projection of R on X is a new relation $\pi_X(R)$ that only has columns in X
- For example, $\pi_{1,2}(R)$ is a table that contains only the 1^{st} and 2^{nd} columns of R
- We can use numbers or names to index columns (naming columns will be discussed in the next lecture)

Projection Example

R

1	2	3
Andrew	aaa@cs.nott.ac.uk	0115 911 1111
Bill	bbb@cs.nott.ac.uk	0115 922 2222
Christine	ccc@cs.nott.ac.uk	0115 933 3333

$\pi_{1,3}(R)$

Andrew	0115 911 1111
Bill	0115 922 2222
Christine	0115 933 3333

Selection

- Sometimes we want to select tuples based on one or more criteria
- Let R be a relation with n columns, and α is a property of tuples
- **Selection from R subject to condition α** is defined as:

$$\sigma_{\alpha}(R) = \{<a_1,\ldots,a_n> \in R: \alpha(a_1,\ldots,a_n)\}$$

Comparison Properties

- We assume that properties are written using {and, or, not} and expressions of the form col(i) Θ col(j), where i, j are column numbers, or col(i) Θ v, where v is a value from domain A_i
- Θ is a comparator which makes sense when applied to values from columns i and j. Often these will be = , ≠, ≤, ≥, $<$,

Meaningful Comparisons

- Comparisons between values can only take place where it makes sense to compare them
  - We can always perform an equivalence test between two values in the same domain
  - In some cases you can compare values from different domains, e.g. if both are strings
- For example, “1975 < 1987” is a meaningful comparison, “Andrew = 1981” is not
- We can only use a comparison in a selection if its result is true or false, never undefined

Selection Example

- $\sigma_{\text{col(3)} < 2002 \text{ and col(2)} = \text{Nolan}}$ (R)

R

Insomnia	Nolan	2002
Magnolia	Anderson	1999
Insomnia	Skjoldbjaerg	1997
Memento	Nolan	2000
Gattaca	Niccol	1997

Selection Example

- $\sigma_{\text{col(3) < 2002 and col(2) = Nolan}}(R)$

R

~~Insomnia~~	~~Nolan~~	~~2002~~
Magnolia	Anderson	1999
Insomnia	Skjoldbjaerg	1997
Memento	Nolan	2000
Gattaca	Niccol	1997

Selection Example

- $\sigma_{\text{col(3) < 2002 and col(2) = Nolan}}$ (R)

R

~~Insomnia~~	~~Nolan~~	~~2002~~
~~Magnolia~~	~~Anderson~~	~~1999~~
~~Insomnia~~	~~Skjoldbjaerg~~	~~1997~~
Memento	Nolan	2000
~~Gattaca~~	~~Niccol~~	~~1997~~

Selection Example

- $\sigma_{\text{col(3) < 2002 and col(2) = Nolan}}(R)$

$\sigma_{\text{col(3) < 2002 and col(2) = Nolan}}(R)$

Memento	Nolan	2000

Other Operations

- Not all SQL queries can be translated into relational algebra operations defined in this lecture
- Extended relational algebra includes counting, joins and other additional operations

Take home messages

1. Relational Model
   - Relations
     - Tuples, attributes, domain
2. Terminology
   - Degree, cardinality
3. Data manipulation
   - Set theoretic operators
   - Operators specific to relations

This Lecture in Exams

What is the result of the following operation?

$\pi_{1,3}(\sigma_{col(2) = col(4)}(R \times S))$, where R and S are:

R

Anne	111111
Bob	222222

S

Chris	111111
Dan	222222

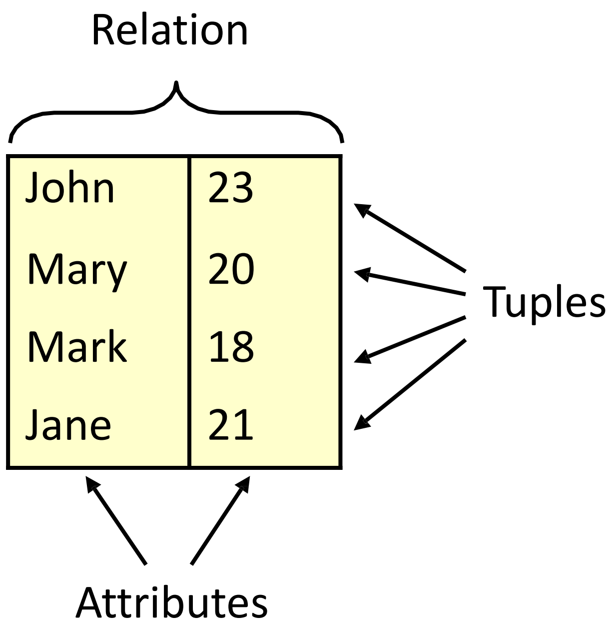
The Relational Model

This lecture

- The Relational Model
  - More on Relations
  - Relational data integrity
    - Candidate, Primary, Foreign Keys

Last lecture

- Data is stored in *relations* (tables)
- Relations are made up of *attributes* (columns)
- Data takes the form of *tuples* (rows)
  - The order of tuples is not important
    - There must not be duplicate tuples



Example from last lecture

What is the result of the following operation?

$\pi_{1,3}(\sigma_{col(2) = col(4)}(R \times S))$, where R and S are:

R

Anne	111111
Bob	222222

S

Chris	111111
Dan	222222

Example from last lecture

$\pi_{1,3}(\sigma_{col(2) = col(4)}(R \times S))$

Start from the inner parenthesis (R x S)

R x S			
Anne	111111	Chris	111111
Bob	222222	Chris	111111
Anne	111111	Dan	222222
Bob	222222	Dan	222222

Example from last lecture

$\pi_{1,3}(\sigma_{col(2) = col(4)}(R \times S))$

Then move outwards, considering the selection

$\sigma_{col(2) = col(4)}(R \times S)$

Anne	111111	Chris	111111
Bob	222222	Dan	222222

Example from last lecture

$\pi_{1,3}(\sigma_{col(2) = col(4)}(R \times S))$

Finally, consider the projection:

$\pi_{1,3}(\sigma_{col(2) = col(4)}(R \times S))$	
Anne	Chris
Bob	Dan

Example from last lecture

$\pi_{1,3}(\sigma_{col(2) = col(4)}(R \times S))$

Start from (R x S)

R x S

Anne	111111	Chris	111111
Bob	222222	Chris	111111
Anne	111111	Dan	222222
Bob	222222	Dan	222222

$\pi_{1,3}(\sigma_{col(2) = col(4)}(S \times R))$

Start from (S x R)

S x R

Chris	111111	Anne	111111
Dan	222222	Anne	111111
Chris	111111	Bob	222222
Dan	222222	Bob	222222

Another example

What about a single table? Can we find a list of pairs of people who share a phone number?

R	
Anne	111111
Chris	222222
Bob	333333
Dan	111111
Max	222222
Sam	444444
Joe	555555

Another example

What about a single table? Can we find a list of pairs of people who share a phone number?

We basically want something like this:

Anne	Dan	
Chris	Max	
Dan	Anne	
Max	Chris	

R X R

Anne	111111	Anne	111111
Chris	222222	Anne	111111
Bob	333333	Anne	111111
Dan	111111	Anne	111111
Max	222222	Anne	111111
Sam	444444	Anne	111111
Joe	555555	Anne	111111
…	…	…	…
Anne	111111	Joe	555555
Chris	222222	Joe	555555
Bob	333333	Joe	555555
Dan	111111	Joe	555555
Max	222222	Joe	555555
Sam	444444	Joe	555555
Joe	555555	Joe	555555

$$\sigma_{\text{col}(2) = \text{col}(4) \text{ and col}(1) \neq \text{col}(3)} \ (R \times R)$$

~~Anne~~	~~111111~~	~~Anne~~	~~111111~~
~~Anne~~	~~111111~~	~~Chris~~	~~222222~~
~~Anne~~	~~111111~~	~~Bob~~	~~333333~~
Anne	111111	Dan	111111
~~Anne~~	~~111111~~	~~Max~~	~~222222~~
~~Anne~~	~~111111~~	~~Sam~~	~~444444~~
~~Anne~~	~~111111~~	~~Joe~~	~~555555~~
…	…	…	…
~~Joe~~	~~555555~~	~~Anne~~	~~111111~~
~~Joe~~	~~555555~~	~~Chris~~	~~222222~~
~~Joe~~	~~555555~~	~~Bob~~	~~333333~~
~~Joe~~	~~555555~~	~~Dan~~	~~111111~~
~~Joe~~	~~555555~~	~~Max~~	~~222222~~
~~Joe~~	~~555555~~	~~Sam~~	~~444444~~
~~Joe~~	~~555555~~	~~Joe~~	~~555555~~

$$\sigma_{col(2) = col(4) \text{ and } col(1) \neq col(3)} (R \times R)$$

Anne	~~111111~~	Dan	~~111111~~
Chris	~~222222~~	Max	~~222222~~
Dan	~~111111~~	Anne	~~111111~~
Max	~~222222~~	Chris	~~222222~~

$$\pi_{1,3}(\sigma_{col(2) = col(4) \text{ and } col(1) \neq col(3)} (R \times R))$$

Anne	Dan
Chris	Max
Dan	Anne
Max	Chris

Another example

What about a single table? Can we find a list of pairs of people who share a phone number?

A: $\pi_{1,3}(\sigma_{col(2) = col(4) \text{ and } col(1) \neq col(3)} (R \times R))$

R

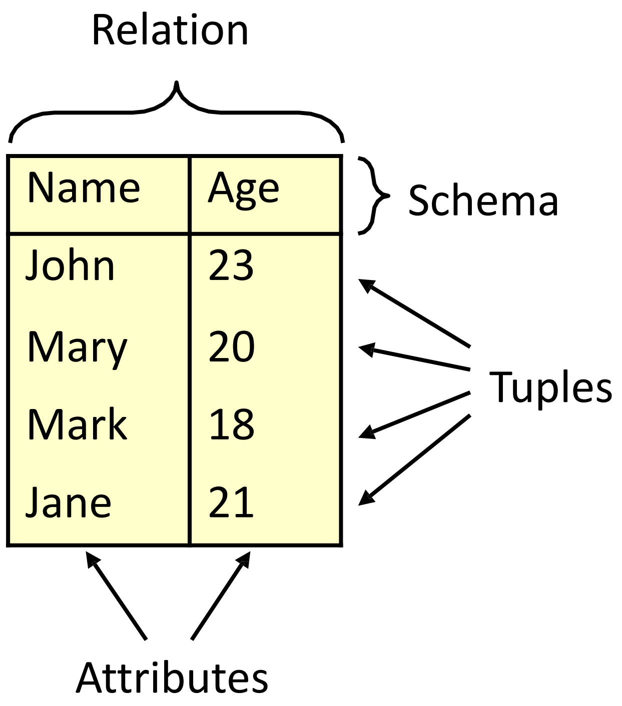
Anne	111111
Chris	222222
Bob	333333
Dan	111111
Max	222222
Sam	444444
Joe	555555

Schemas and Attributes

- Previously, we referenced specific columns in a relation using numbers
  - E.g. $\pi_{1,2}(R)$
- It is often helpful to reference columns using names, which we will have to provide
- *Attributes* are named columns in a relation
- A *schema* defines the attributes for a relation

Relational Data Structure

- Each relation has a *schema* (sometimes called a scheme or heading)
- The schema defines the relation's *attributes* (columns).



Named and Unnamed Tuples

- Tuples specify values for each attribute in a relation
- When writing tuples down, they can be named as sets of pairs, e.g.
  - { (1, John), (2, 23) } or { (2, 23), (1, John) }
  - { (Name, John), (Age, 23) }
- Or unnamed, for convenience, e.g.
  - (John, 23) (equivalent to the above)
- There is no real difference between named and unnamed tuples, but be careful with the ordering of unnamed tuples.

Relational Data Structure

- More formally:
  - A schema is a set of attributes
  - A tuple assigns a value to each attribute in the schema
  - A relation is a set of tuples with the same schema

Name	Age
John	23
Mary	20
Mark	18
Jane	21

{ { (Name, John), (Age, 23) },
{ (Name, Mary), (Age, 20) },
{ (Name, Mark), (Age, 18) },
{ (Name, Jane), (Age, 21) } }

Example Relation

ID	Name	Salary	Department
M139	John Smith	18,000	Marketing
M140	Mary Jones	22,000	Marketing
A368	Jane Brown	22,000	Accounts
P222	Mark Brown	24,000	Personnel
A367	David Jones	20,000	Accounts

Example Relation

ID	Name	Salary	Department
M139	John Smith	18,000	Marketing
M140	Mary Jones	22,000	Marketing
A368	Jane Brown	22,000	Accounts
P222	Mark Brown	24,000	Personnel
A367	David Jones	20,000	Accounts

Schema is { ID, Name, Salary, Department }

Example Relation

Attributes are ID, Name, Salary and Department

The degree of the relation is 4

Schema is { ID, Name, Salary, Department }

ID	Name	Salary	Department
M139	John Smith	18,000	Marketing
M140	Mary Jones	22,000	Marketing
A368	Jane Brown	22,000	Accounts
P222	Mark Brown	24,000	Personnel
A367	David Jones	20,000	Accounts

Example Relation

Attributes are ID, Name, Salary and Department

The degree of the relation is 4

ID	Name	Salary	Department
M139	John Smith	18,000	Marketing
M140	Mary Jones	22,000	Marketing
A368	Jane Brown	22,000	Accounts
P222	Mark Brown	24,000	Personnel
A367	David Jones	20,000	Accounts

Schema is { ID, Name, Salary, Department }

Tuples, e.g.
{ (ID, A368),
(Name, Jane Brown),
(Salary, 22,000),
(Department, Accounts)}

The cardinality of the relation is 5

Relational Data Integrity

- Data integrity controls what data can be in a relation
  - *Domains* restrict the possible values a tuple can assign to each attribute
  - *Candidate* and *Primary Keys* consist of an attribute, or set of attributes, that uniquely identify each tuple that appears in a relation
  - *Foreign Keys* link relations to each other

Attributes and Domains

- A *domain* is given for each attribute
- The domain lists possible values for the attribute
- Each tuple assigns a value to each attribute from *its domain*

- Examples
  - An ‘age’ might have to come from the set of integers between 0 and 150
  - A ‘department’ might come from a list of given strings
  - A ‘notes’ field may allow any string at all

Candidate Keys

- A set of attributes in a relation is a candidate **key** if, and only if:
  - Every tuple has a unique value for that set of attributes: ***uniqueness***
  - No proper subset of the set has the uniqueness property: ***minimality***

ID	First	Last
S139	Alan	Carr
S140	Jo	Brand
S141	Alan	Davies
S142	Jimmy	Carr

Candidate key is {ID}; {First, Last} looks plausible, but people might have the same name

{ID, First}, {ID, Last} and {ID, First, Last} satisfy uniqueness, but are not minimal

{First} and {Last} do not give a unique identifier for each row

Choosing Candidate Keys

- You can’t necessarily infer the candidate keys based solely on the data in your table
  - More often than not, an instance of a relation will only hold a small subset of all the possible values
- You must use knowledge of the real-world to help

Choosing Candidate Keys

What are the candidate keys of the following relation?

Relations have names

CompanyOffices

officeID	Name	Country	Postcode/Zip	Phone
O1001	Headquarters	England	W1 1AA	0044 20 1545 3241
O1002	R&D Labs	England	W1 1AA	0044 20 1545 4984
O1003	US West	USA	94130	001 415 665981
O1004	US East	USA	10201	001 212 448731
O1005	Telemarketing	England	NE5 2GE	0044 1909 559862
O1006	Telemarketing	USA	84754	001 385 994763

Choosing Candidate Keys

The candidate keys are {OfficeID}, {Phone} and {Name, Postcode/Zip}

officeID	Name	Country	Postcode/Zip	Phone
O1001	Headquarters	England	W1 1AA	0044 20 1545 3241
O1002	R&D Labs	England	W1 1AA	0044 20 1545 4984
O1003	US West	USA	94130	001 415 665981
O1004	US East	USA	10201	001 212 448731
O1005	Telemarketing	England	NE5 2GE	0044 1909 559862
O1006	Telemarketing	USA	84754	001 385 994763

Note: Keys like {Name, Country, Phone} satisfy uniqueness, but not minimality

Primary Keys

- One candidate key is usually chosen to identify tuples in a relation
- This is called the *Primary Key*
- Often a special ID is used as the Primary Key

ID	First	Last
S139	Alan	Carr
S140	Jo	Brand
S141	Alan	Davies
S142	Jimmy	Carr

We might use either {ID} or {First,Last} as the primary key. ID is more convenient as we know it will always be unique. People could have the same name

NULLs and Primary Keys

- Missing information can be represented using NULLs
- A NULL indicates a missing or unknown value
- This will be discussed in a later lecture

- *Entity integrity*

  Primary Keys cannot contain NULL values

Foreign Keys

- Foreign Keys are used to link data in two relations. A set of attributes in the first (referencing) relation is a Foreign Key if its value:
  - Matches a Candidate Key value in a second (referenced) relation
  - Is NULL
- This is called *Referential Integrity*

Foreign Keys Example

Department

DID	DName
13	Marketing
14	Accounts
15	Personnel

Employee

EID	EName	DID
15	John Smith	13
16	Mary Brown	14
17	Mark Jones	13
18	Jane Smith	NULL

{DID} is a Candidate Key for Department – Each entry has a unique value for DID

{DID} is a Foreign Key in Employee – each employee's DID value is either NULL, or matches an entry in the Department relation. This links each Employee to at most one Department

Recursive Foreign Keys Example

Employee

ID	Name	Manager
E1496	John Smith	E1499
E1497	Mary Brown	E1498
E1498	Mark Jones	E1499
E1499	Jane Smith	NULL

{ID} is a Candidate Key for Employee, and {Manager} is a Foreign Key that refers to the same relation. Every tuple's Manager value must match an ID value, or be NULL

Naming Conventions

- Naming conventions
  - A consistent naming convention can help to remind you of the structure
  - Assign each table a unique prefix, so a student name may be stuName,and a module name modName
  - You may even wish to assign a project prefix to the tables you use
- Naming keys
  - Having a unique number as the primary key can be useful
  - If the table prefix is abc, call this abcID
  - A foreign key to this table is then also called abcID

Relational Data Integrity

- *Data integrity* controls what data can be in a relation
- *Domains* restrict the possible values a tuple can assign to each attribute
- *Candidate and Primary Keys* consist of an attribute, or set of attributes, that uniquely identify each tuple that appears in a relation
- Foreign Keys link relations to each other

Referential Integrity

- When relations are updated, referential integrity might be violated
- • This usually occurs when a referenced tuple is updated or deleted
- • There are a number of options when this occurs: • RESTRICT – stop the user from doing it
- • CASCADE – let the changes flow on
- • SET NULL – make referencing values null
- • SET DEFAULT – make referencing values the default for their column

Referential Integrity Example

- • What happens if
-
- • Marketing's DID is changed to 16 in Department?
- • The entry for Accounts is deleted from Department

- • Using RESTRICT, CASCADE and SET NULL

Department

DID	DName
13	Marketing
14	Accounts
15	Personnel

Employee

EID	EName	DID
15	John Smith	13
16	Mary Brown	14
17	Mark Jones	13
18	Jane Smith	NULL

RESTRICT

- • What happens if
-
- • Marketing’s DID is changed to 16 in Department?
- • The entry for Accounts is deleted from Department

Department

DID	DName
13	Marketing
14	Accounts
15	Personnel

Employee

EID	EName	DID
15	John Smith	13
16	Mary Brown	14
17	Mark Jones	13
18	Jane Smith	NULL

RESTRICT

- RESTRICT stops any action that violates integrity
  - You cannot update or delete Marketing or Accounts
  - You *can* change Personnel as it is not referenced

Department

DID	DName
13	Marketing
14	Accounts
15	Personnel

Employee

EID	EName	DID
15	John Smith	13
16	Mary Brown	14
17	Mark Jones	13
18	Jane Smith	NULL

CASCADE

- • What happens if
-
- • Marketing’s DID is changed to 16 in Department?
- • The entry for Accounts is deleted from Department

Department

DID	DName
13	Marketing
14	Accounts
15	Personnel

Employee

EID	EName	DID
15	John Smith	13
16	Mary Brown	14
17	Mark Jones	13
18	Jane Smith	NULL

CASCADE

- CASCADE allows the changes made to flow through
  - If Marketing's DID is changed to 16 in Department, then the DIDs for John Smith and Mark Jones also change
  - If Accounts is deleted then so is Mary Brown

Department

DID	DName
~~13~~ 16	Marketing
~~14~~	~~Accounts~~
15	Personnel

Employee

EID	EName	DID
15	John Smith	~~13~~ 16
~~16~~	~~Mary Brown~~	~~14~~
17	Mark Jones	~~13~~ 16
18	Jane Smith	NULL

SET NULL

- • What happens if
-
- • Marketing's DID is changed to 16 in Department?
- • The entry for Accounts is deleted from Department

- • Using RESTRICT, CASCADE and SET NULL

Department

DID	DName
13	Marketing
14	Accounts
15	Personnel

Employee

EID	EName	DID
15	John Smith	13
16	Mary Brown	14
17	Mark Jones	13
18	Jane Smith	NULL

SET NULL

- SET NULL allows the changes to happen but
  - If Marketing's DID is changed to 16 in Department, then the DIDs for John Smith and Mark Jones is set to NULL
  - If Accounts is deleted then Mary Brown's DID is set to NULL

Department

DID	DName
~~13~~ 16	Marketing
~~14~~	~~Accounts~~
15	Personnel

Employee

EID	EName	DID
15	John Smith	~~13~~ NULL
16	Mary Brown	~~14~~ NULL
17	Mark Jones	~~13~~ NULL
18	Jane Smith	NULL

Naming Example

Student

stuID	stuName

Enrolment

stuID	modID

Module

modID	modName

These attributes are clearly related to the student table

These attributes are foreign keys, related to other tables

These attributes are clearly related to the module table

Entity Relationship Modelling

Last topic

- Foreign Keys reference a Candidate Key in another relation.

Genre

GID	GName
10	Crime
11	Thriller
12	Biography
13	Mystery

BookGenres

GID	BID
10	2
11	2
12	3
13	4
11	4

Database Design

- Before we look at how to create and use a database we'll look at how to design one
- Need to consider
  - What tables, keys, and constraints are needed?
  - What is the database going to be used for?
- Designing your database is important
  - We can create a database design that is independent of DBMS
  - Often results in a more efficient and simpler queries once the database has been created

Entity/Relationship Modelling

- E/R Modelling is used for conceptual design
  - Entities - objects or items of interest
  - Attributes – properties of an entity
  - Relationships - links between entities

- For example, in a University database we might have entities for Students, Modules and Lecturers
  - Students might have attributes such as their ID, Name, and Course
  - Students could have relationships with Modules (enrolment) and Lecturers (tutor/tutee)

Entity/Relationship Diagrams

- E/R Models are often represented as E/R diagrams that
    - Give a conceptual view of the database
    - Are independent of the choice of DBMS
    - Can identify some problems in a design

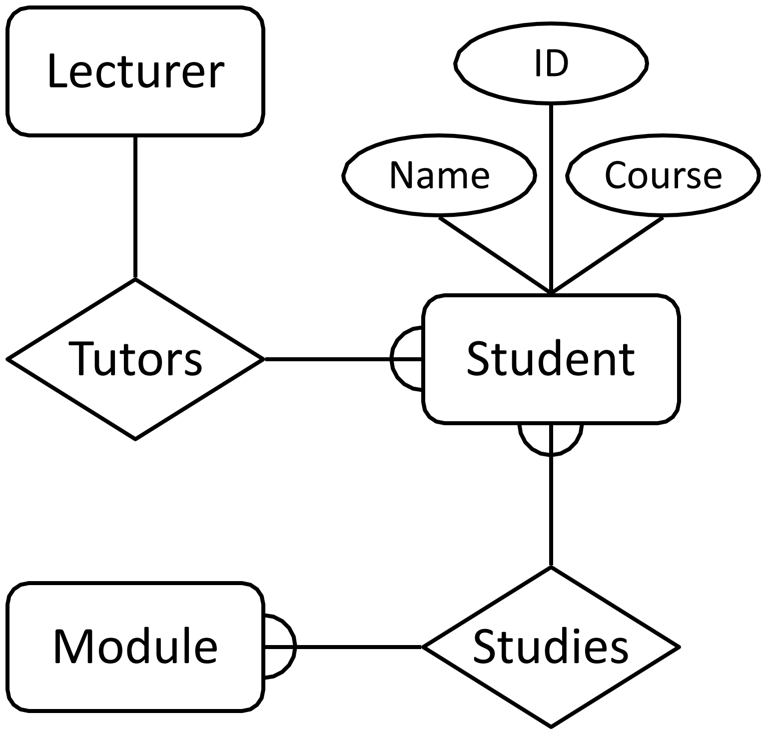
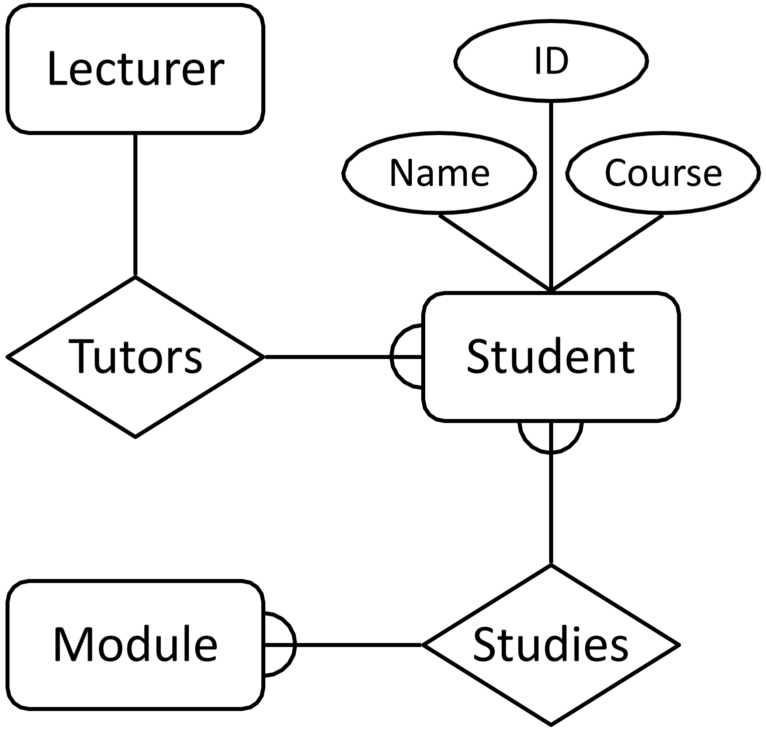


Diagram Conventions

- There are various notations for representing E/R diagrams
- These specify the shape of the various components, and the notation used to represent relationships
- For this introductory module, we will use simplified diagrams

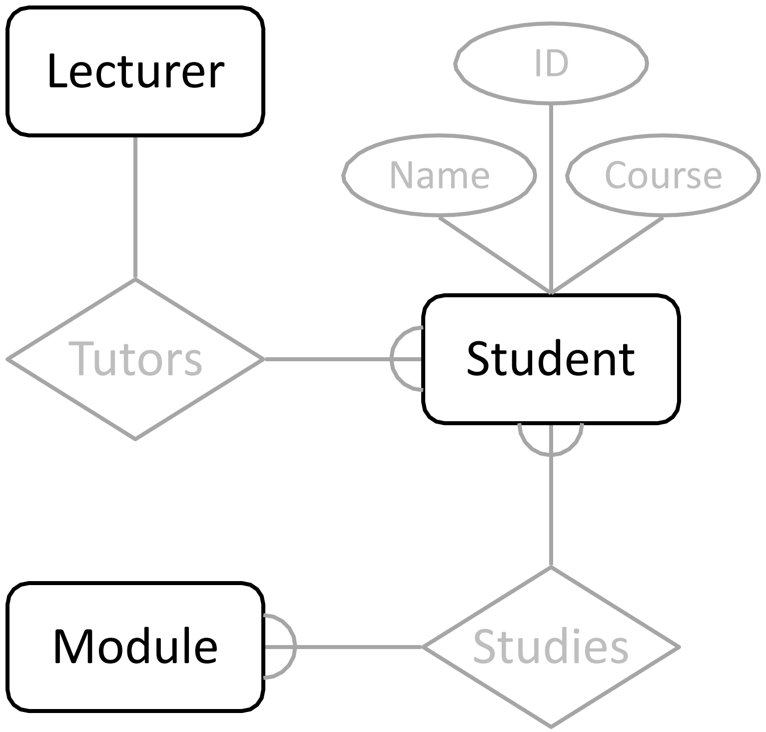


Entities

- Entities represent objects or things of interest
  - Physical things like students, lecturers, employees, products
  - More abstract things like modules, orders, courses, projects
- Entities have
  - A general type or class, such as Lecturer or Module
  - Instances of that particular type. E.g. Boriana Koleva, Steve Bagley are instances of Lecturer
  - Attributes (such as name, email address)

Diagramming Entities

- In E/R Diagrams, we will represent Entities as boxes with rounded corners
- The box is labelled with the name of the class of objects represented by that entity

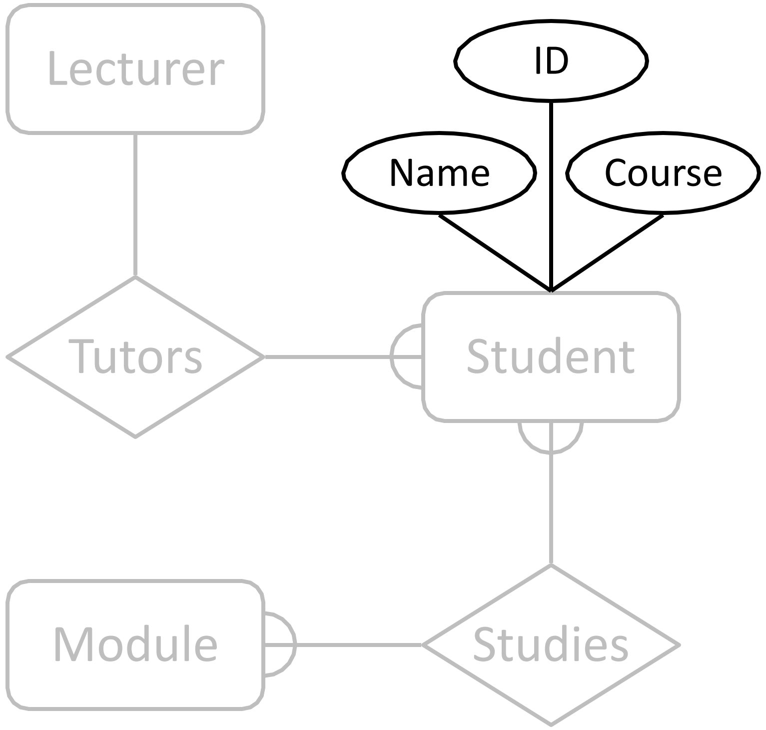


Attributes

- Attributes are facts, aspects, properties, or details about an entity
  - Students have IDs, names, courses, addresses, …
  - Modules have codes, titles, credit weights, levels, …
- Attributes have
  - A name
  - An associated entity
  - Domains of possible values
  - For each instance of the associated entity, a value from the attributes domain

Diagramming Attributes

- In an E/R Diagram attributes are drawn as ovals
- Each attribute is linked to its entity by a line
- The name of the attribute is written in the oval



Relationships

- Relationships are an association between two or more entities
  - Each Student takes several Modules
  - Each Module is taught by a Lecturer
  - Each Employee works for a single Department

- Relationships have
  - A name
  - A set of entities that participate in them
  - A degree - the number of entities that participate (most have degree 2)
  - A cardinality ratio

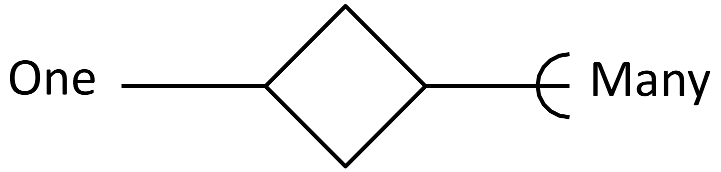
Cardinality Ratios

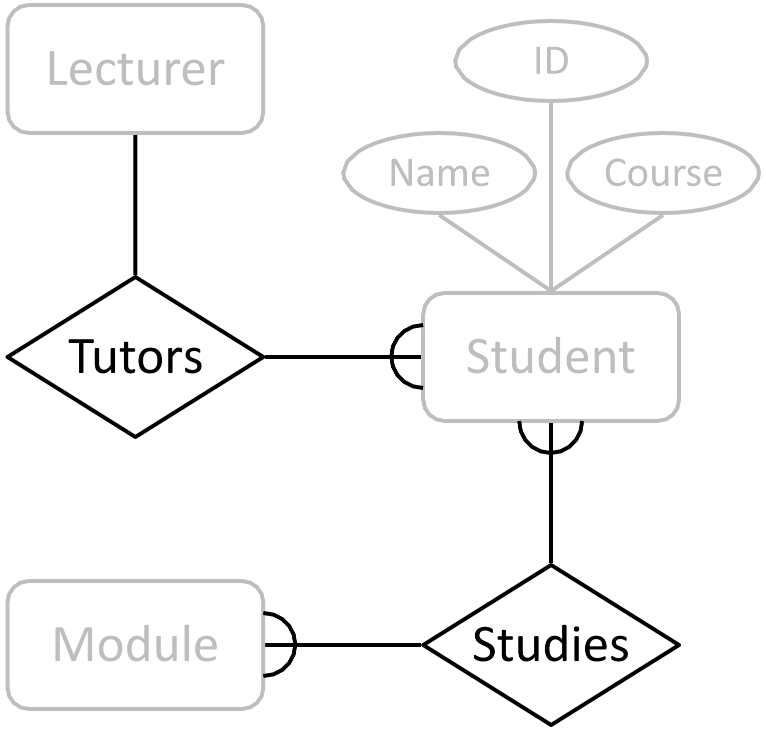
- Each entity in a relationship can participate in zero, one, or more than one instances of that relationship
- We won’t be dealing with optional (zero instances) of relationships
- This leads to 3 types of relationship...

- One to one (1:1)
  - Each lecturer has a unique office & offices are single occupancy
- One to many (1:M)
  - A lecturer may tutor many students, but each student has just one tutor
- Many to many (M:M)
  - Each student takes several modules, and each module is taken by several students

Entity/Relationship Diagrams

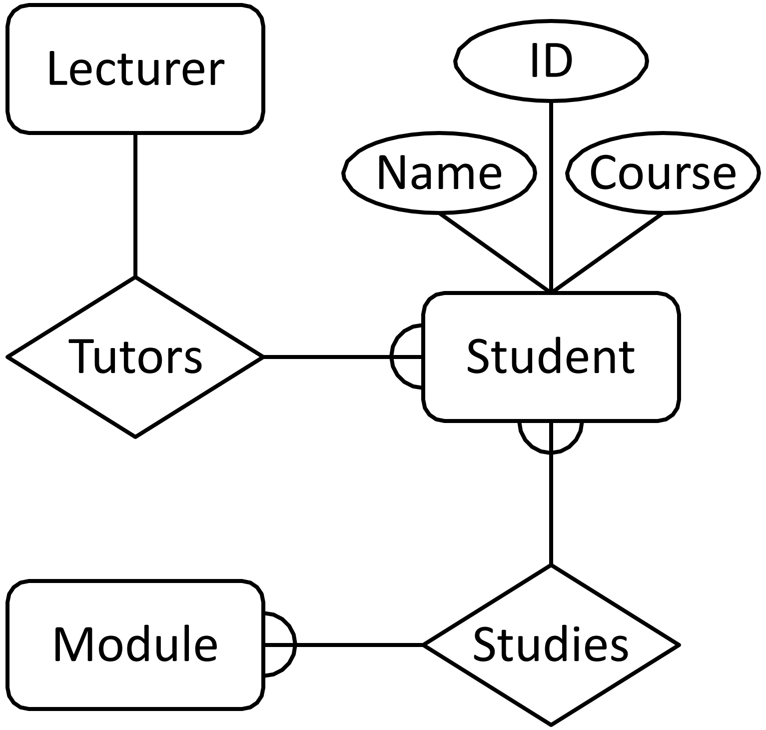
- Relationships are shown as links between two entities
- The name is given in a diamond box
- The ends of the link show cardinality





Entity/Relationship Diagrams

- Final E/R diagram looks like this:



Making E/R Models

- To make an E/R model you need to identify
  - Entities
  - Attributes
  - Relationships
  - Cardinality ratios
- We obtain these from a problem description

- General guidelines
  - Since entities are things or objects they are often nouns in the description
  - Attributes are facts or properties, and so are often nouns also
  - Verbs often describe relationships between entities

Example

- A university consists of a number of departments. Each department of several courses. A number of modules make up each course. Students enrol in a particular course and take modules towards the completion that course. Each module is taught by a lecturer from the appropriate department (several lecturer work in the same department), and each lecturer tutors a group of students. A lecturer can teach more than one module but can work only in one department.

Example - Entities

- A university consists of a number of departments. Each department of several courses. A number of modules make up each course. Students enrol in a particular course and take modules towards the completion that course. Each module is taught by a lecturer from the appropriate department (several lecturer work in the same department), and each lecturer tutors a group of students. A lecturer can teach more than one module but can work only in one department.

- Entities – Department, Course, Module, Student, Lecturer

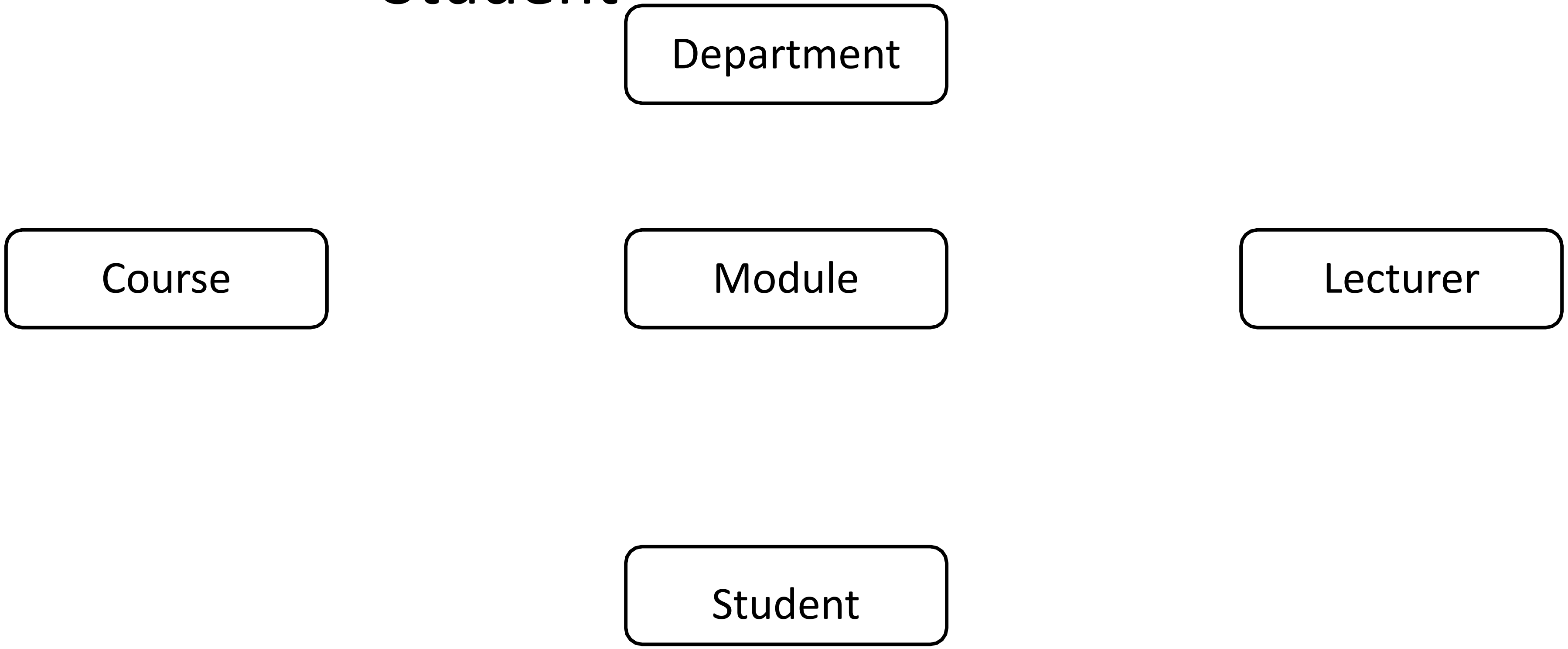
Example - Relationships

A university consists of a number of departments. Each department offers several courses. A number of modules make up each course. Students enrol in a particular course and take modules towards the completion of that course. Each module is taught by a lecturer from the appropriate department (several lecturer work in the same department), and each lecturer tutors a group of students. A lecturer can teach more than one module but can work only in one department.

- Entities – Department, Course, Module, Student, Lecturer
- Relationships – Offers, Make Up, Enrol, Take, Taught By, From The, Tutors

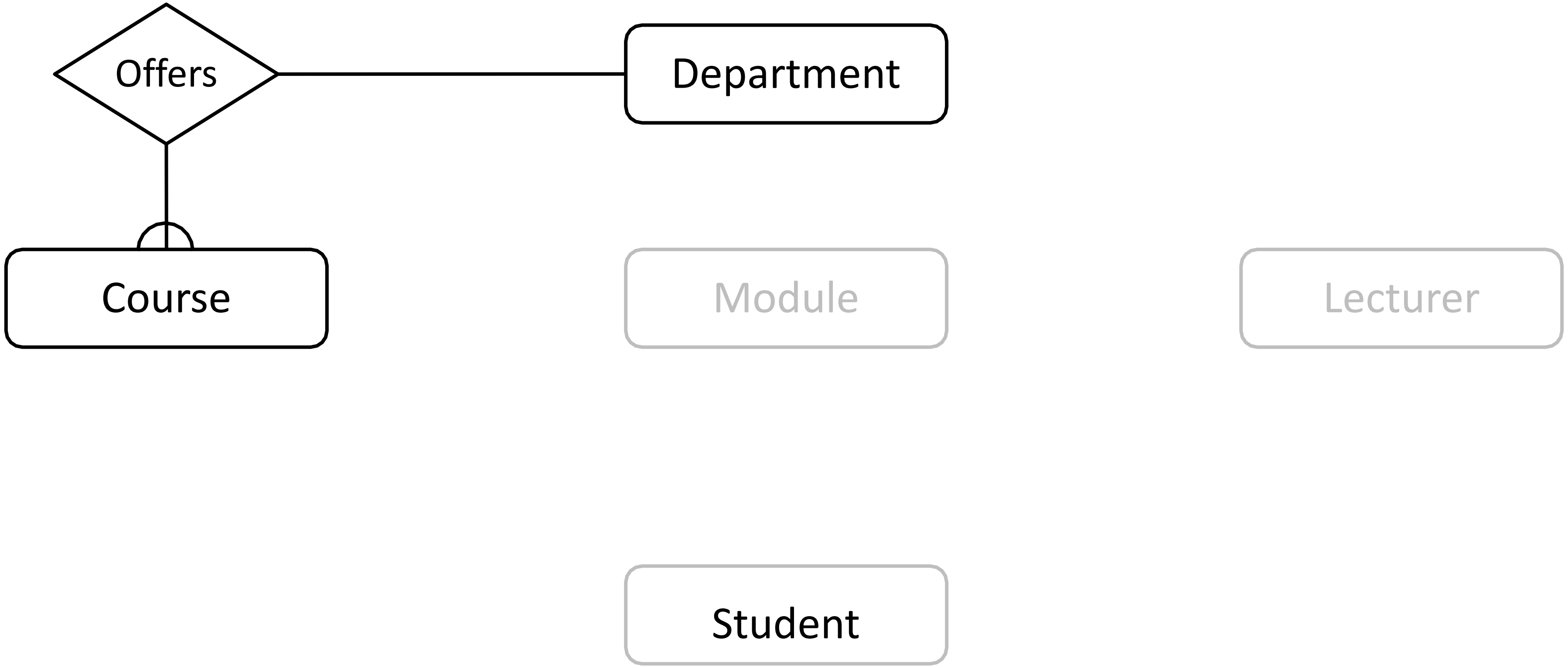
Example – E/R Diagram

Entities: Department, Course, Module, Lecturer,
Student



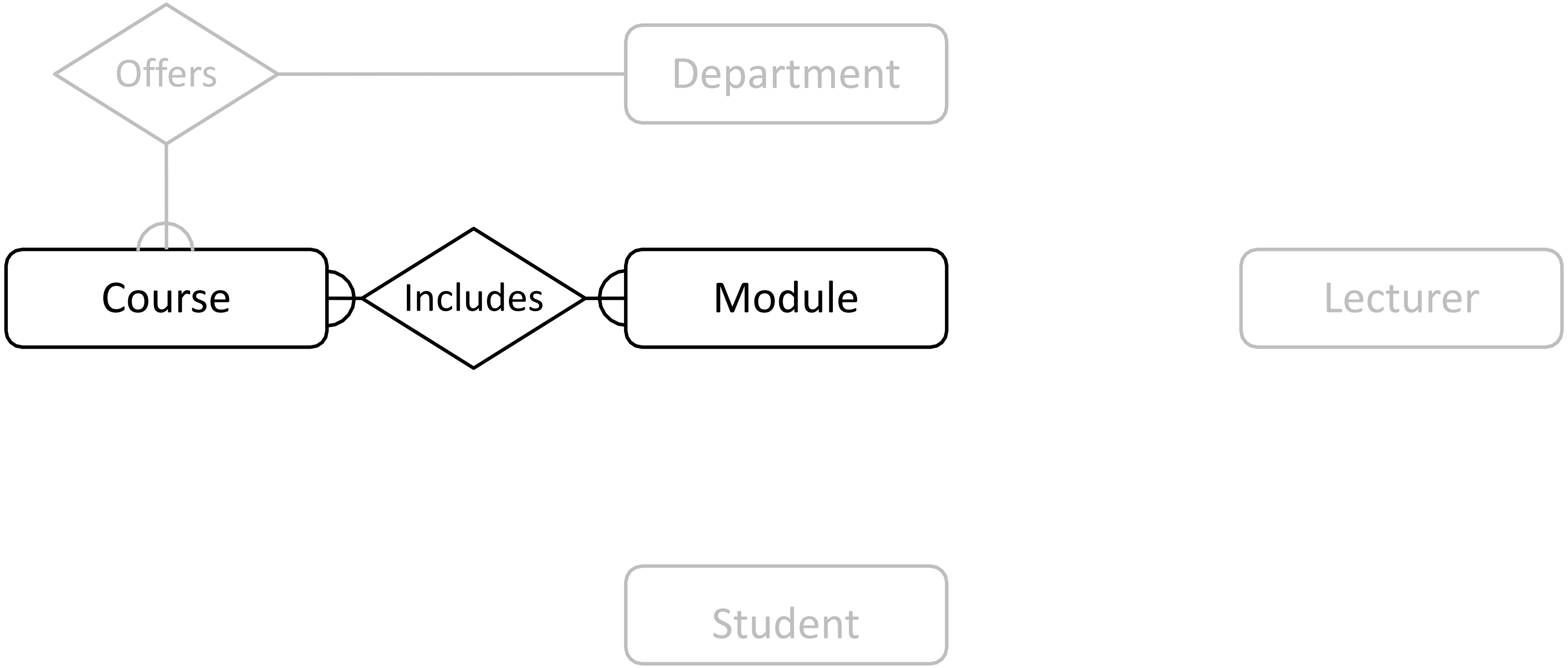
Example – E/R Diagram

Each Department offers several Courses



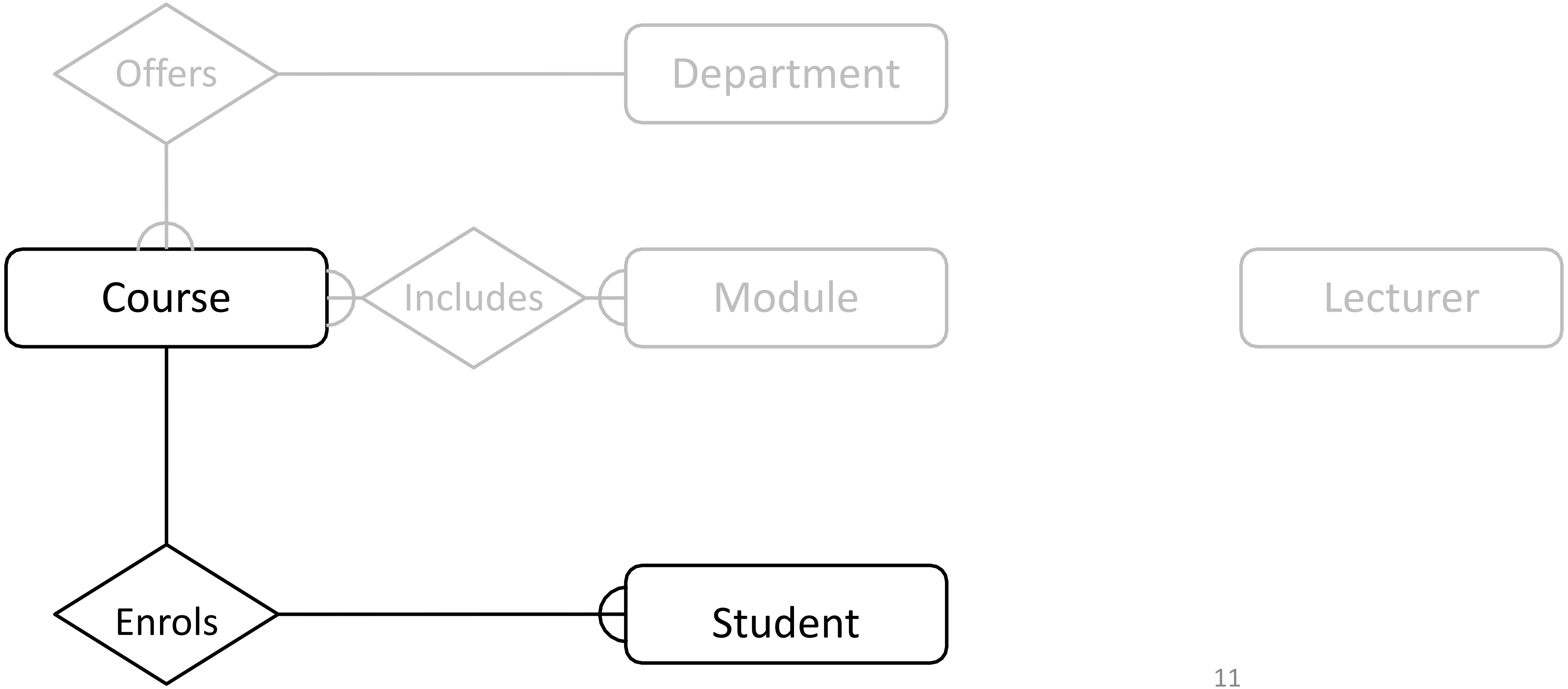
Example – E/R Diagram

A number of modules make up each Course



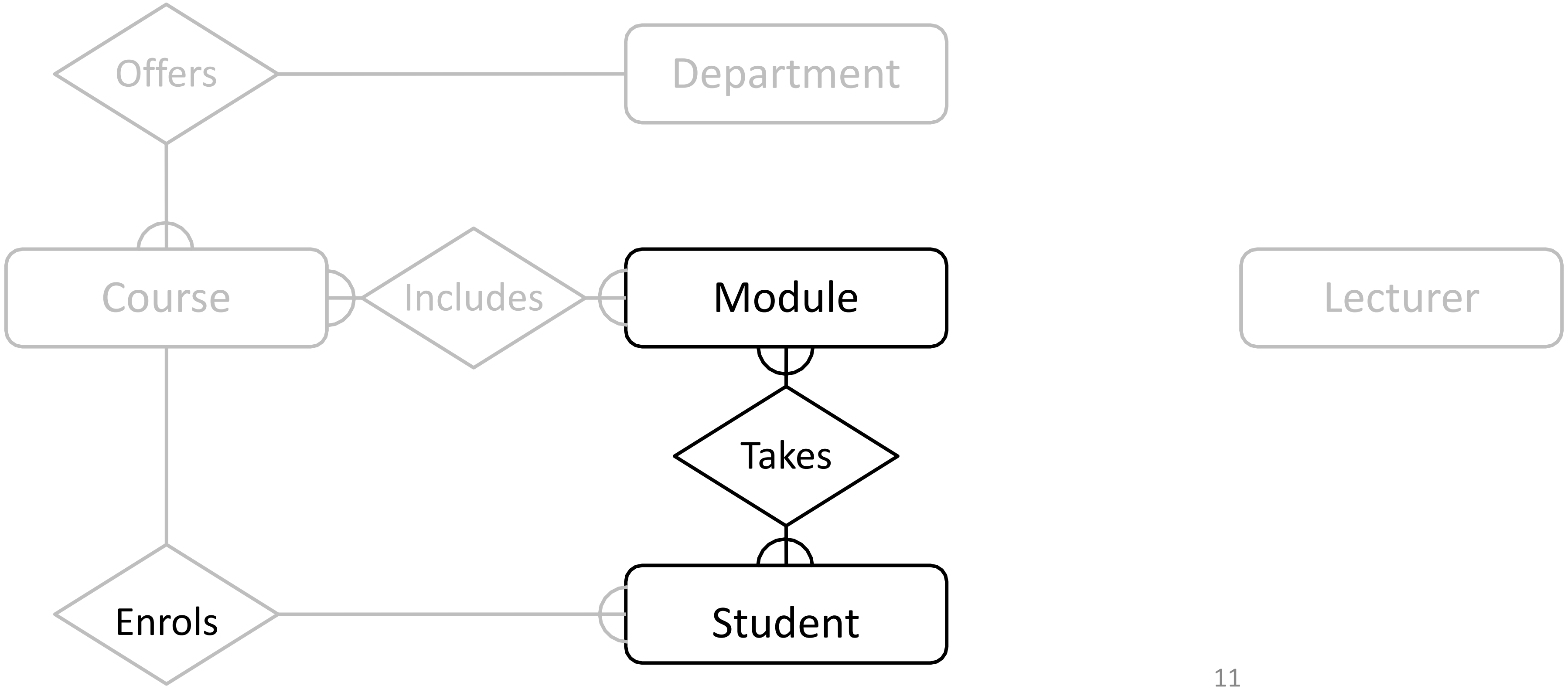
Example – E/R Diagram

Students enrol in a particular course



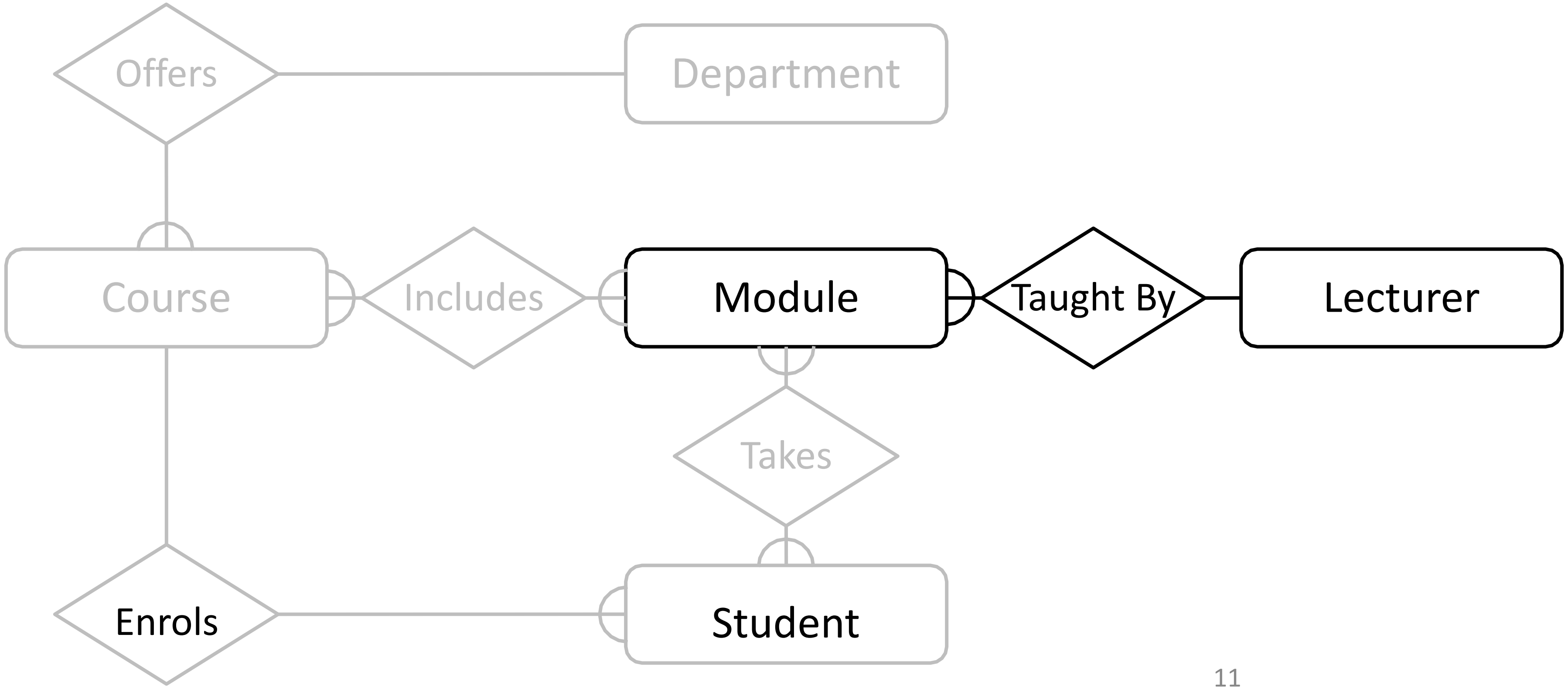
Example – E/R Diagram

Students take several modules



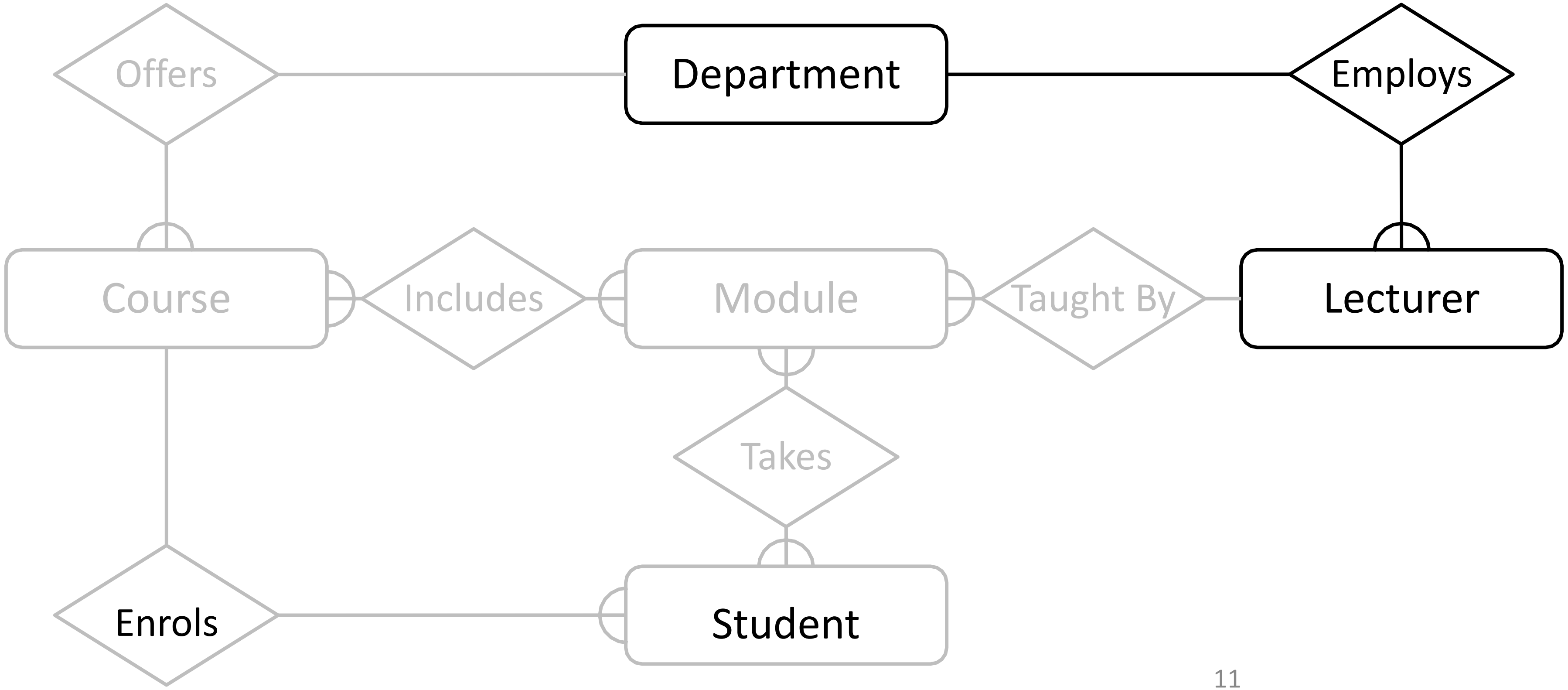
Example – E/R Diagram

Each Module is taught by a Lecturer



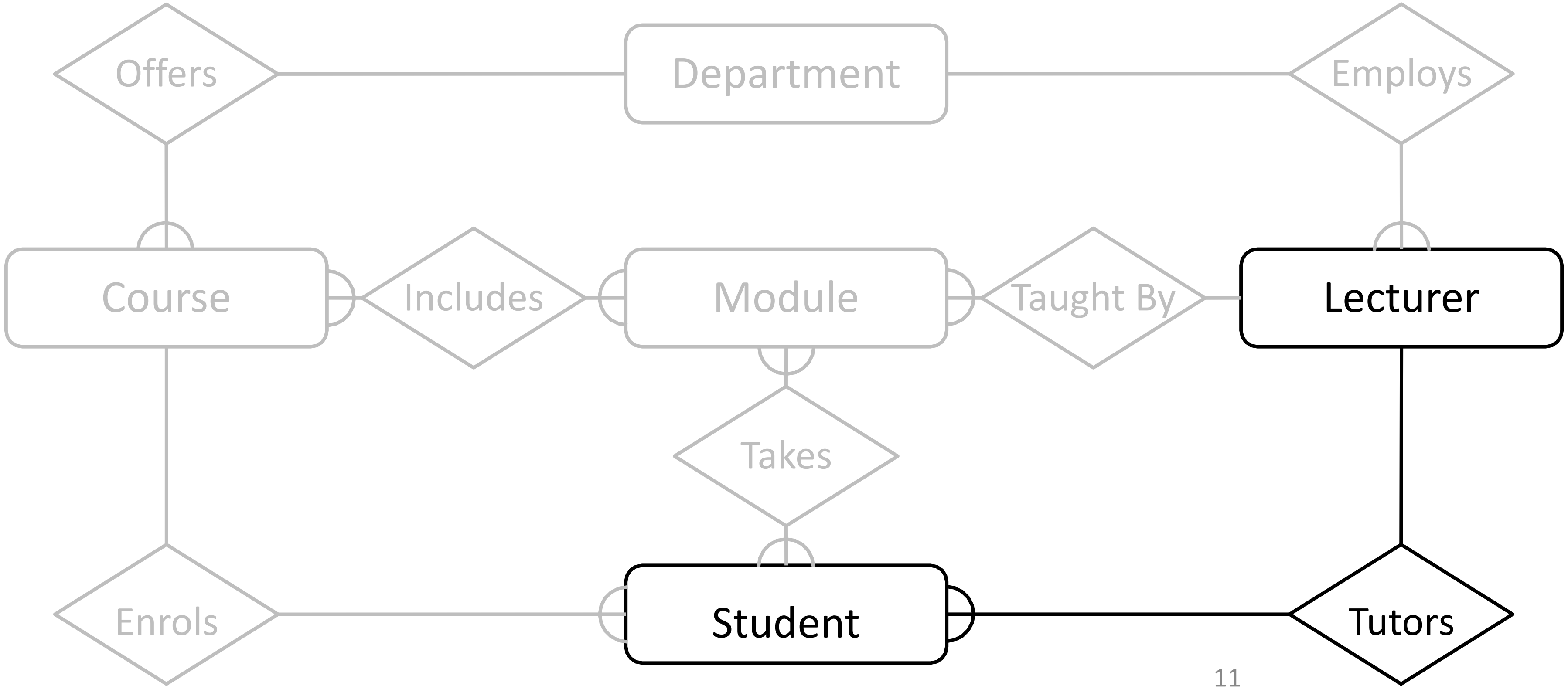
Example – E/R Diagram

Each department employs a number of lecturers



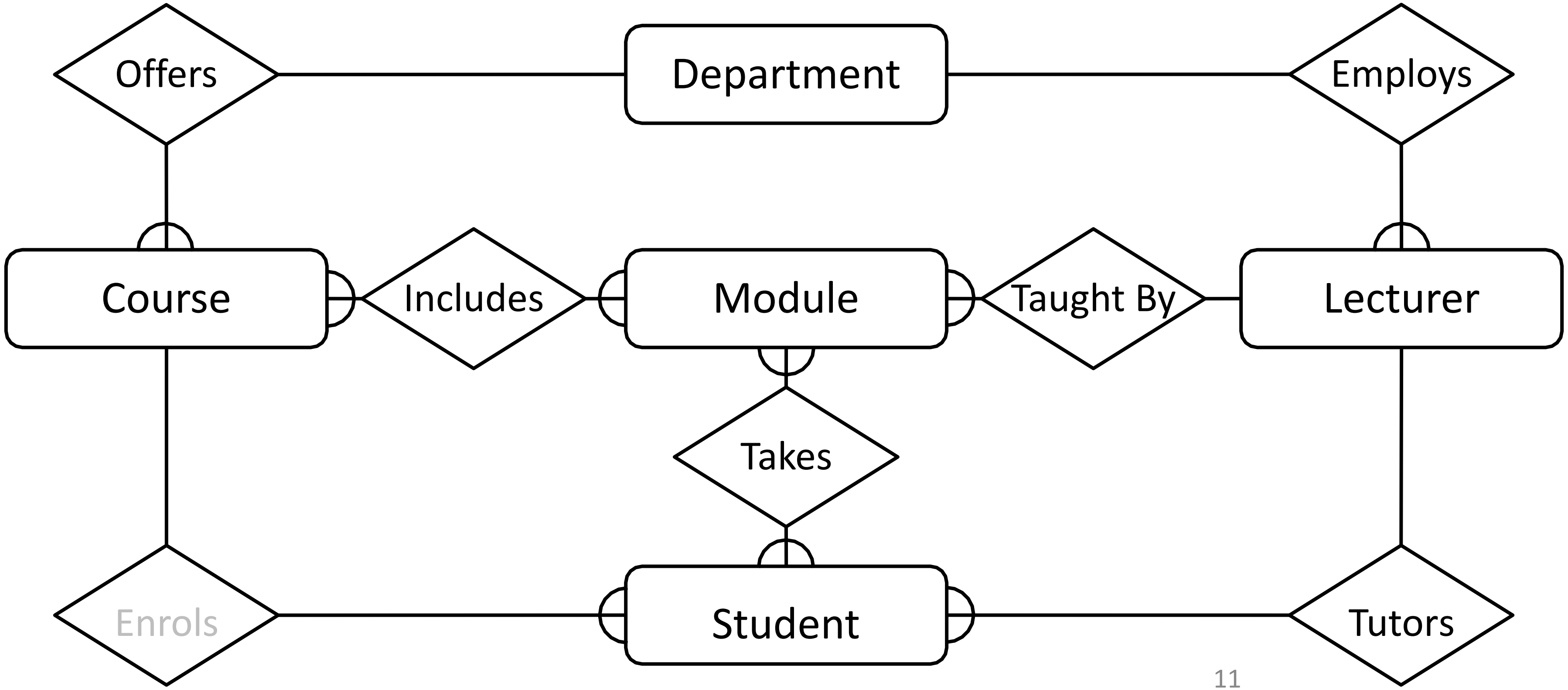
Example – E/R Diagram

Each Lecturer tutors a number of Students



Example – E/R Diagram

The completed diagram. All that remains is to remove M:M relationships



Removing M:M Relationships

- Many to many relationships are difficult to represent in a database:

Student

SID	SName	SMod
1001	Jack Smith	DBS
1001	Jack Smith	PRG
1001	Jack Smith	IAI
1002	Anne Jones	PRG
1002	Anne Jones	IAI
1002	Anne Jones	VIS

Module

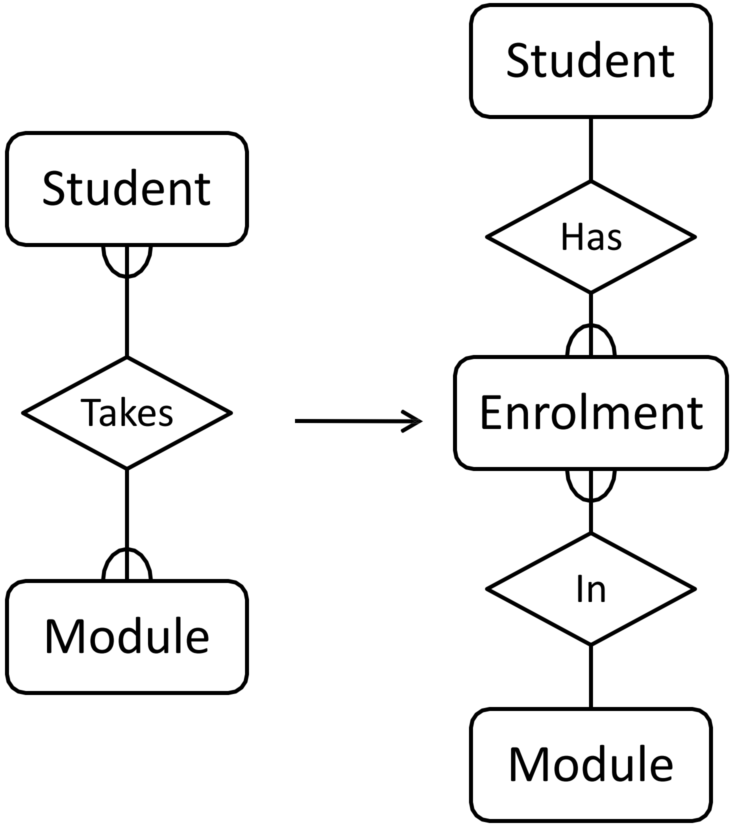
MID	MName
DBS	Database Systems
PRG	Programming
IAI	AI
VIS	Computer Vision

Student

SID	SName	SMods
1001	Jack Smith	DBS, PRG, IAI
1002	Anne Jones	PRG, IAI, VIS

Removing M:M Relationships

- Many to many relationships are difficult to represent in a database
- We can split a many to many relationship into two one to many relationships
- An additional entity is created to represent the M:M relationship



Entities and Attributes

- Sometimes it is hard to tell if something should be an entity or an attribute
  - They both represent objects or facts about the world
  - They are both often represented by nouns in descriptions
- General guidelines
  - Entities can have attributes but attributes have no smaller parts
  - Entities can have relationships between them, but an attribute belongs to a single entity

Example

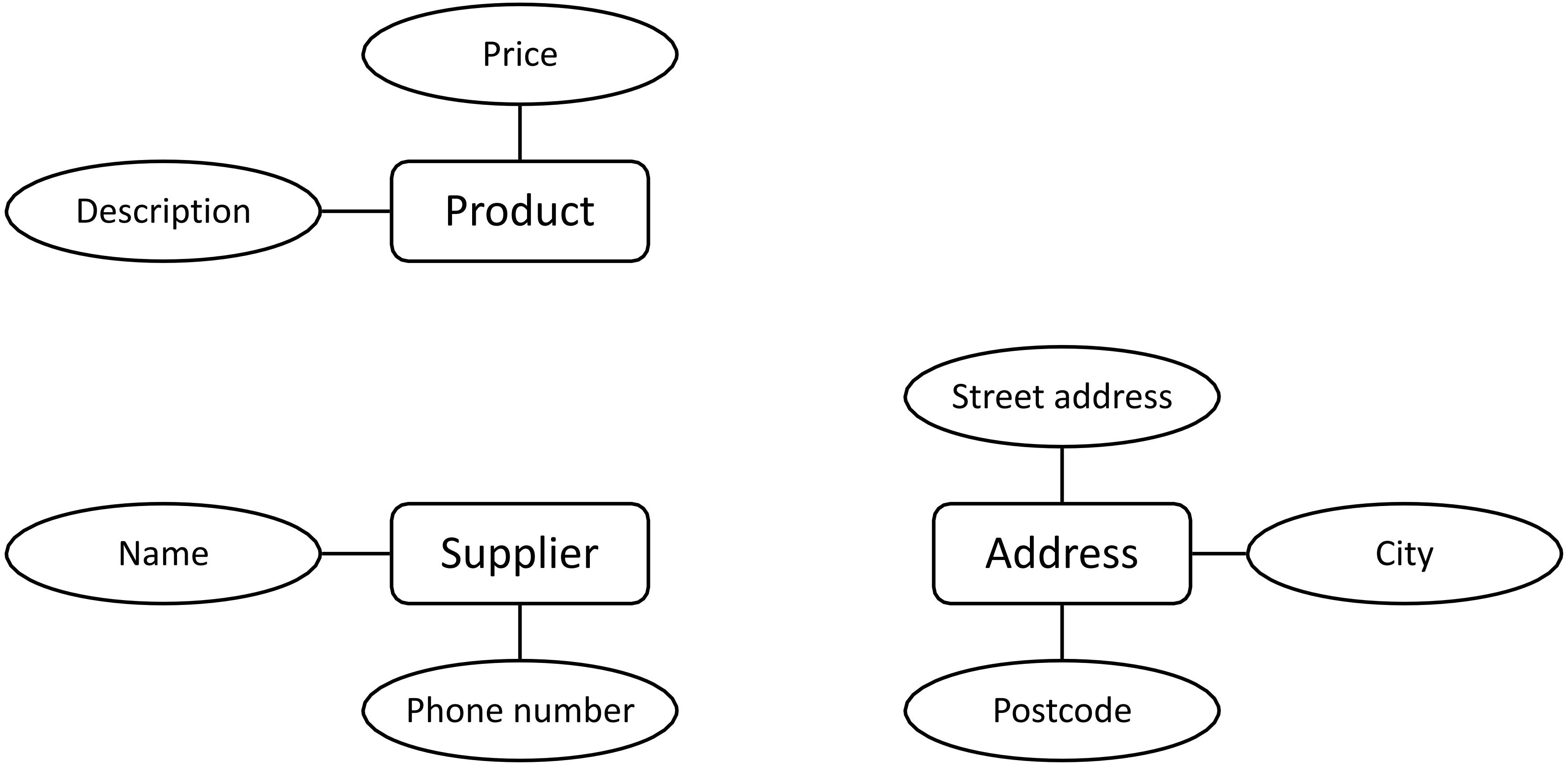
- We want to represent information about products in a database. product has a description, a price and a supplier. Suppliers have addresses, phone numbers, and names. Each address is made up street address, a city, and a postcode.

Example - Entities/Attributes

- Entities or attributes:
    - product
    - description
    - price
    - supplier
    - address
    - phone number
    - name
    - street address
    - city
    - postcode

- Products, suppliers, and addresses all have smaller parts so we make them entities
- The others have no smaller parts and belong to a single entity

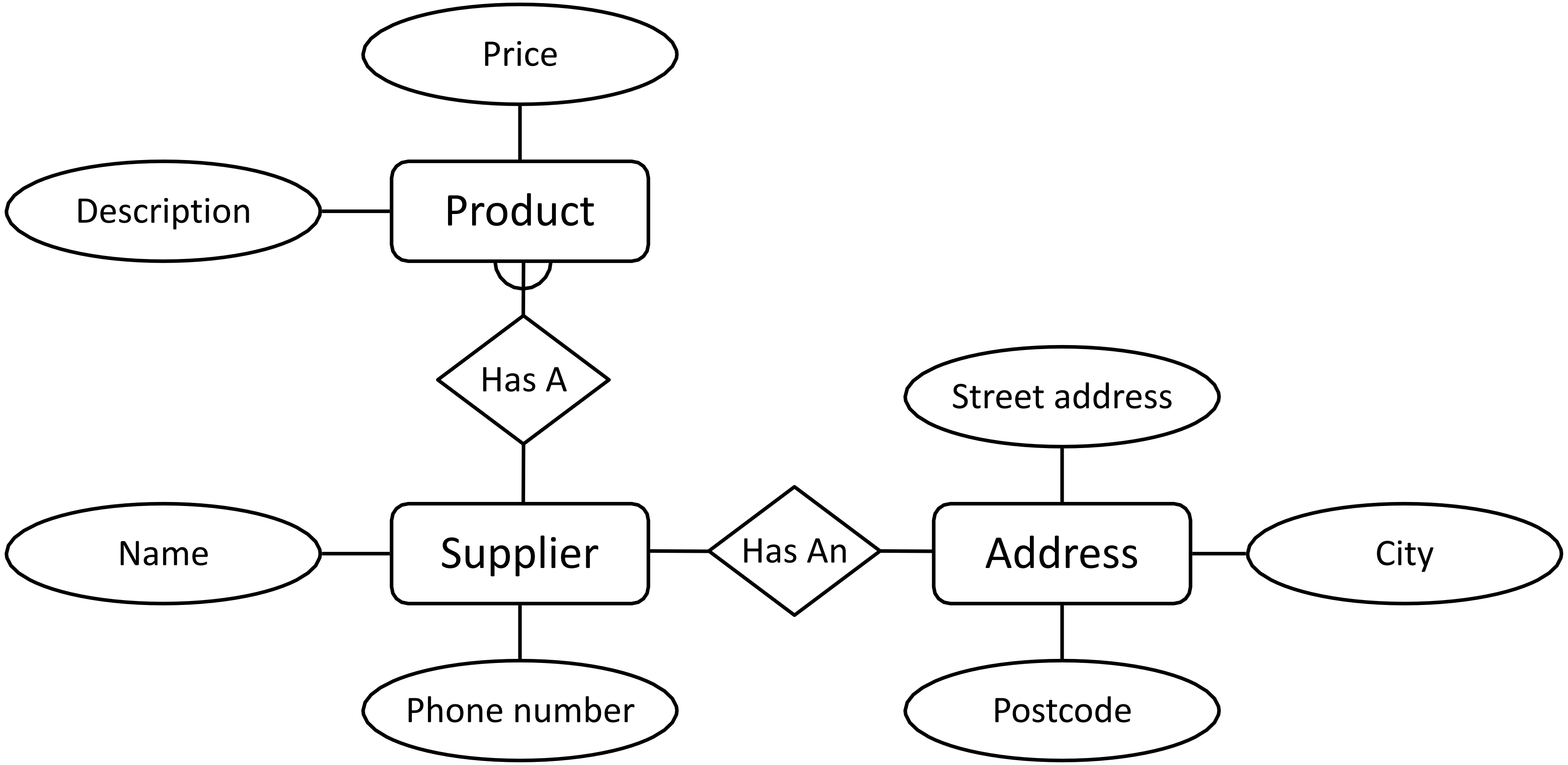
Example - E/R Diagram



Example - Relationships

- Each product has a supplier
  - Each product has a single supplier but there is nothing to stop a supplier supplying many products
  - A many to one relationship
- Each supplier has an address
  - A supplier has a single address
  - It does not seem sensible for two different suppliers to have the same address
  - A one to one relationship

Example - E/R Diagram

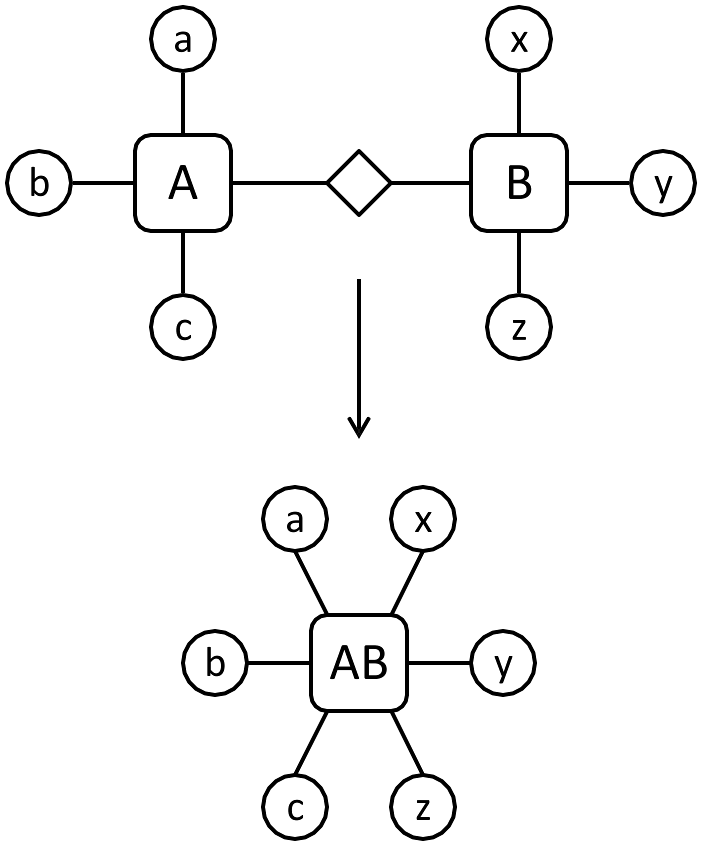


One to One Relationships

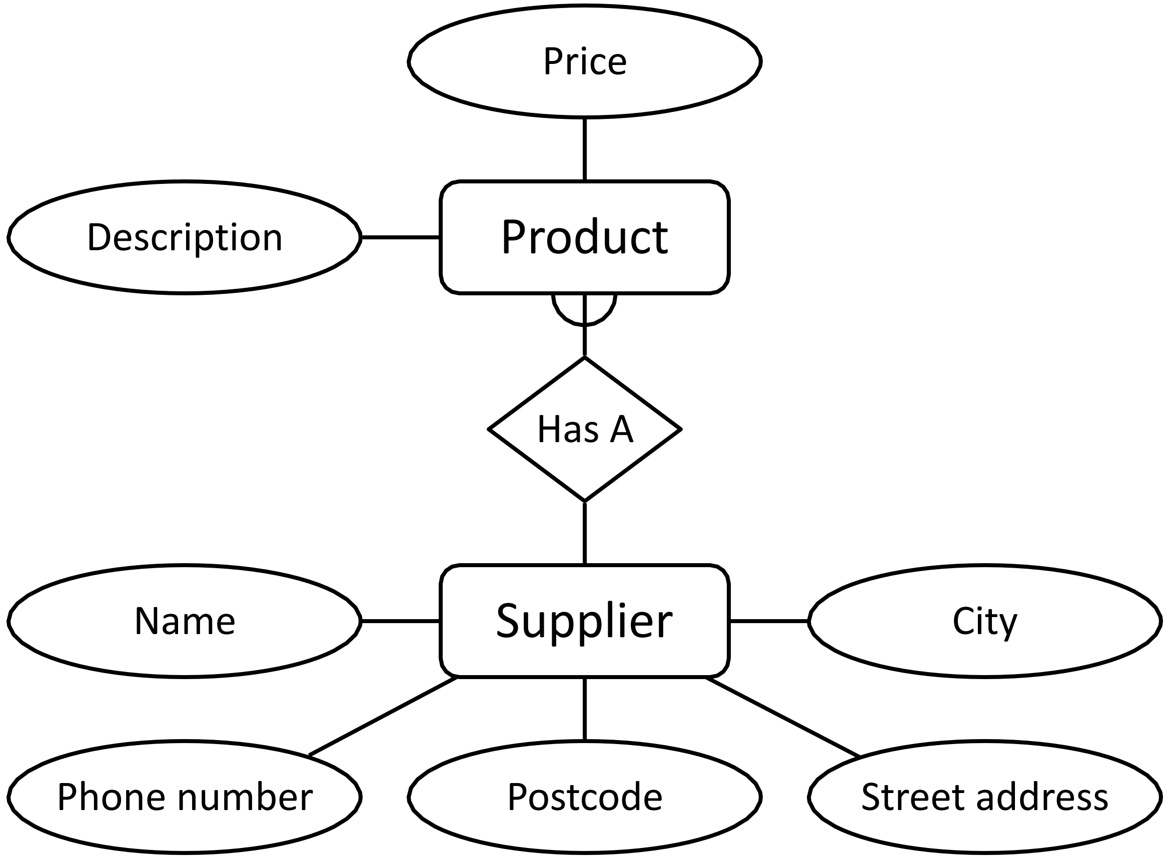
- *Some* relationships between entities, A and B, *might* be redundant if
  - It is a 1:1 relationship between A and B
  - Every A is related to a B and every B is related to an A
- Example - the supplier-address relationship
  - Is one to one
  - Every supplier has an address
  - We don’t need addresses that are not related to a supplier

Redundant Relationships

- We can merge the two entities that take part in a redundant relationship together
  - They become a single entity
  - The new entity has all the attributes of the old ones



Example - E/R Diagram



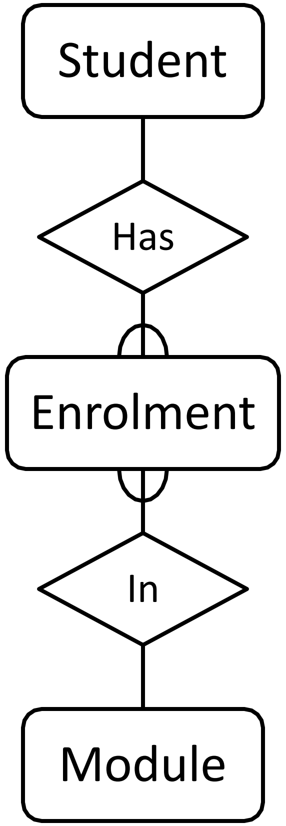
Making E/R Diagrams

- From a description of the requirements identify the
  - Entities
  - Attributes
  - Relationships
  - Cardinality ratios of the relationships

- Draw the E/R diagram
- and then
  - Look at one to one relationships as they might b
redundant
  - Look at many to many relatio
as they will often need to be
into two one to many links, u
intermediate entity

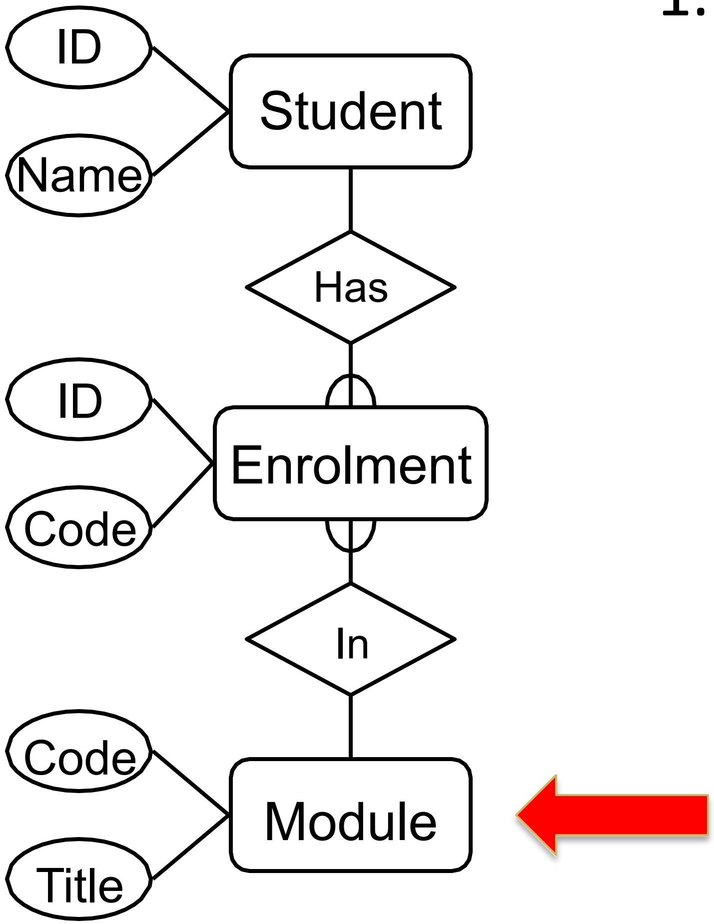
Debugging Designs

- With a bit of practice E/R diagrams can be used to plan queries
  - You can look at the diagram and figure out how to find useful information
  - If you can't find the information you need, you may need to change the design



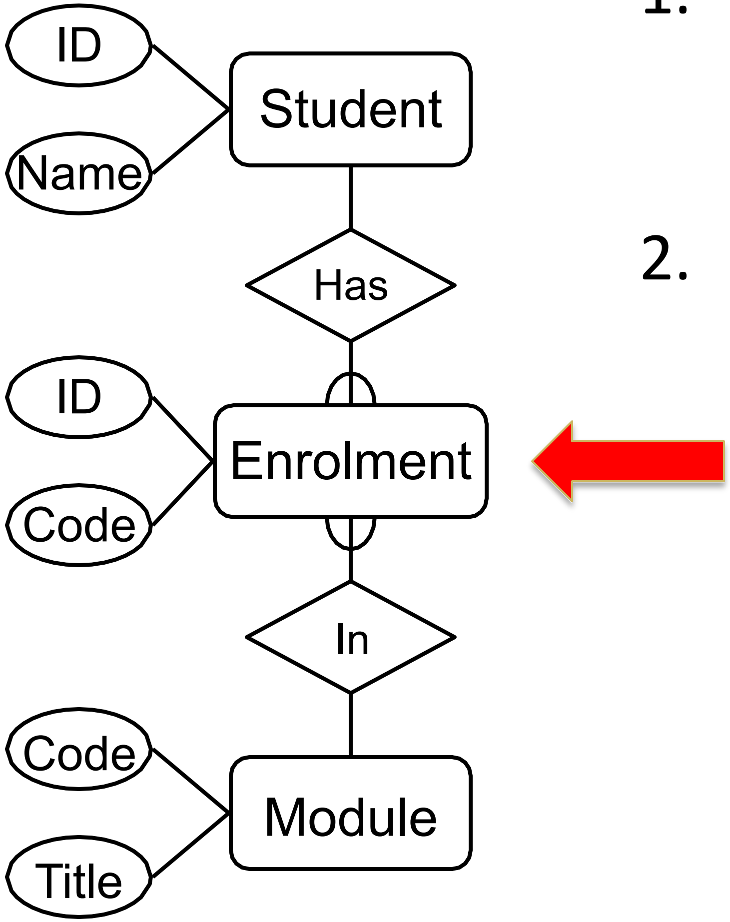
How can you find a list of students who are enrolled in Database systems?

Debugging Designs



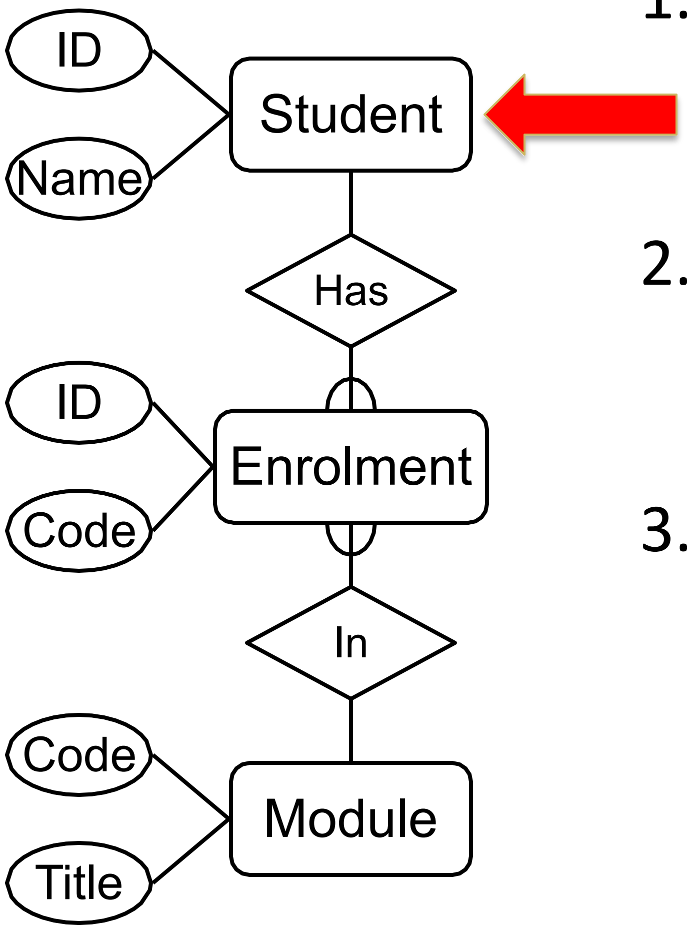
1. Find the instance of Module with the title 'Database Systems'

Debugging Designs



1. Find the instance of Module with the title ‘Database Systems’
2. Find instances of the Enrolment entity with the same Code as the result of (1)

Debugging Designs



1. Find the instance of Module with the title 'Database Systems'
2. Find instances of the Enrolment entity with the same Code as the result of (1)
3. For each instance of Enrolment in the result of (2) find the corresponding student

This Lecture in Exams and Coursework

"A multi-screen cinema wants to create a database for the items that cleaners collect at the end of each film being shown, to improve the recycling operations of the whole cinema and help the environment. The organisation of the database is as follows. Each item that cleaners collect will be given a record in the database. Information stored for a given item consists of an ID number, type of rubbish it represents (plastic, aluminium/can, glass, paper, non-recyclable item), approximate weight, and size (small, medium, big). Items will be collected from different screen rooms (locations). Each location will consist of a unique identifier (screen number), the number of seats available, size of the screen (small, medium, big) and the cleaner assigned. To improve operation, each cleaner will be assigned to one or more locations, but multiple staff cannot be assigned to the same location. Information held on cleaners will include staffID and Name."

BEWARE: Similar to the above but HARDER

This Lecture in Exams and Coursework

Identify the *entities*, *attributes*, *relationships*, and *cardinality ratios* from the description.

Draw an entity-relationship diagram showing the items you identified.

Many-to-many relationships are hard to represent in database tables. Explain the nature of these problems, and describe how they may be overcome.

Take home messages (2)

1. Database Design
   a. Entity Relationship Modelling
   b. Entity Relationship Diagrams
      i. Entities
      ii. Attributes
      iii. Relationships
         – Cardinality Ratios (1:1, 1:M, M:M)

Next Lecture

- SQL
  - The SQL language
  - SQL, the relational model, and E/R diagrams
  - CREATE TABLE
    - Columns
    - Primary Keys
    - Foreign Keys

SQL Data Definition

This Lecture

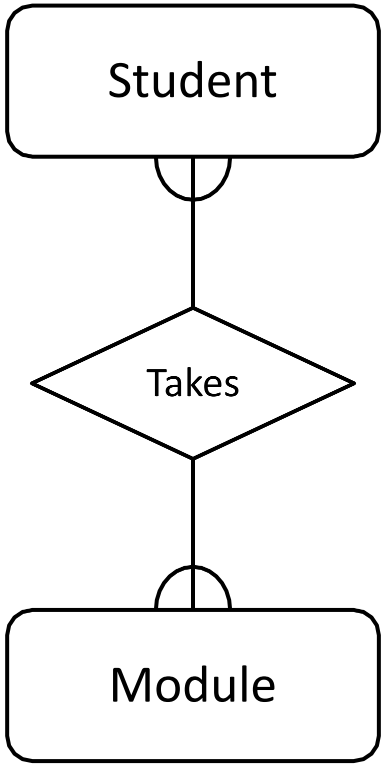
- SQL
  - The SQL language
  - SQL, the relational model, and E/R diagrams
  - CREATE TABLE
    - Columns
    - Primary Keys
    - Foreign Keys
- Further Reading
  - Database Systems, Connolly & Begg, Chapter 7.3
  - The Manga Guide to Databases, Chapter 4

Learning Outcomes

- Introduce the SQL language and its basic commands to create database tables
- Understand how terminology and keywords change throughout the different topics covered in the module
- Familiarise with SQL terms and practice elementary queries

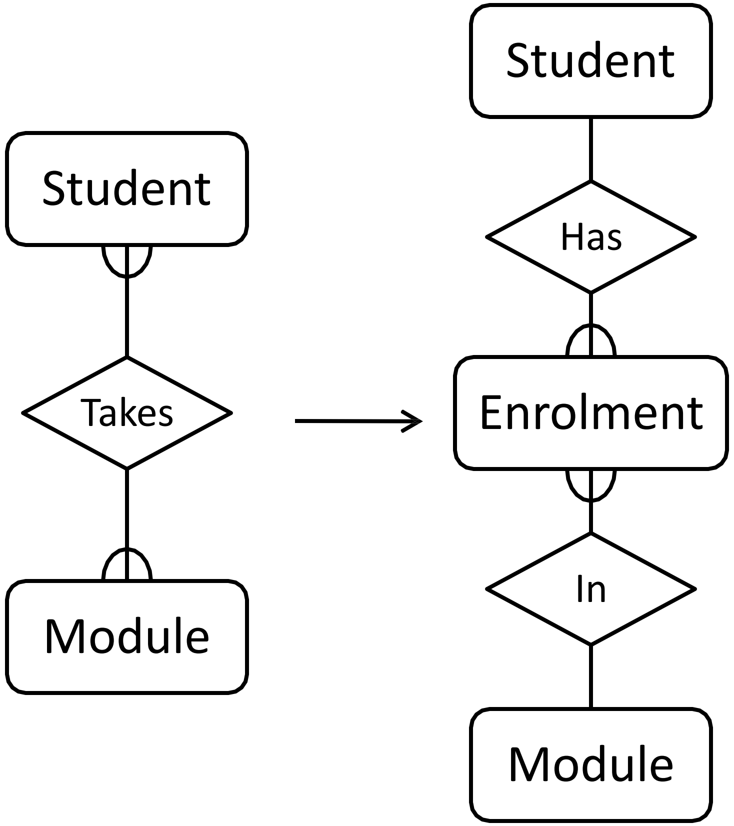
Last Lecture

- Entity Relationship Diagrams
  - Entities
  - Attributes
  - Relationships
- Example
  - Students take many Modules
  - Modules will be taken by many Students



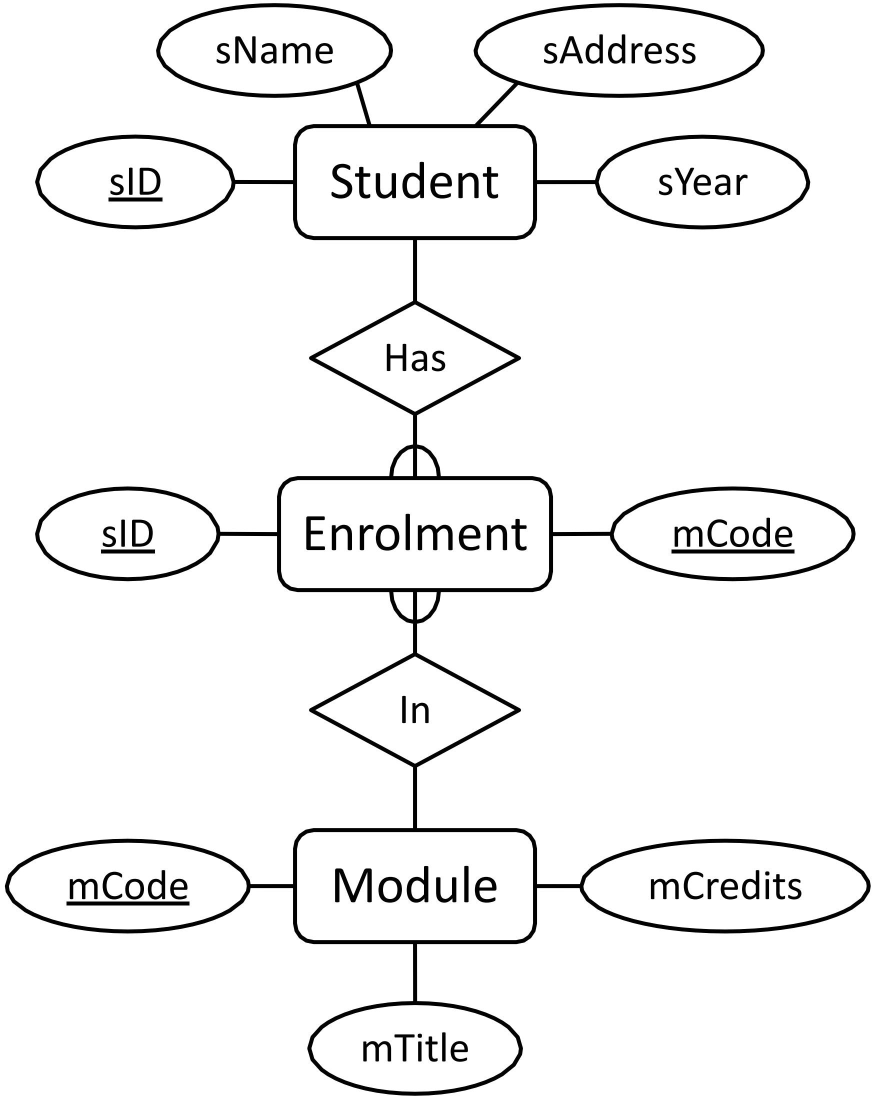
Removing M:M Relationships

- Many to many relationships are difficult to represent in a database
- We can split a many to many relationship into two one to many relationships
- An additional entity is created to represent the M:M relationship



Last Lecture

- Entity Relationship Diagrams (ERD)
  - Entities
  - Attributes
  - Relationships
- Primary keys (PKs)
  - PKs are underlined attributes in ERD



This Lecture

- SQL
  - The SQL language
  - SQL, the relational model, and E/R diagrams
  - CREATE TABLE
    - Columns
    - Primary Keys
    - Foreign Keys
- Further Reading
  - Database Systems, Connolly & Begg, Chapter 7.3
  - The Manga Guide to Databases, Chapter 4

SQL

- Originally 'Sequel' - Structured English query Language, part of an IBM project in the 70's
- Sequel was already taken, so it became SQL - Structured Query Language

- ANSI Standards and a number of revisions
  - SQL-89
  - SQL-92 (SQL2)
  - SQL-99 (SQL3)
  - …
  - SQL:2008 (SQL 2008)
- Most modern DBMS use a variety of SQL
  - Few (if any) are true to the standard

SQL

- SQL is a language based on the relational model
  - Actual implementation is provided by a DBMS
- SQL is everywhere
  - Most companies use it for data storage
  - All of us use it dozens of times per day
  - You will be expected to know it as a software developer

- SQL provides
  - A Data Definition Language (DDL)
  - A Data Manipulation Language (DML)
  - A Data Control Language (DCL)

Provided Languages

- Data Definition Language (DDL)
  - Specify database format
- Data Manipulation Language (DML)
  - Specify and retrieve database contents
- Data Control Language (DCL)
  - Specify access controls (privileges)
- Which are often all one piece of software
  - E.g. SQL

Database Management Systems

- A DBMS is a software system responsible for allowing users access to data
- A DBMS will usually
  - Allow the user to access data using SQL
  - Allow connections from other programming languages
  - Provide additional functionality like concurrency

- There are many DBMSs, some popular ones include:
  - Oracle
  - DB2
  - Microsoft SQL Server
  - Ingres
  - PostgreSQL
  - MySQL
  - Microsoft Access (with SQL Server as storage engine)

SQL Case

- SQL statements will be written in **`COURIER FONT BOLD`**
- SQL keywords are not case-sensitive, but we'll write SQL keywords in upper case for emphasis
- Table names, column names etc. are case sensitive
- For example:

```
SELECT * FROM Student
    WHERE sName = 'James';
```

SQL Strings

- Strings in SQL are surrounded by single quotes:
  - **'I AM A STRING'**
- Single quotes within a string are doubled or escaped using \
  - **A STRING'**

'I''M A STRING'

- **''** is an empty string

'I\'M

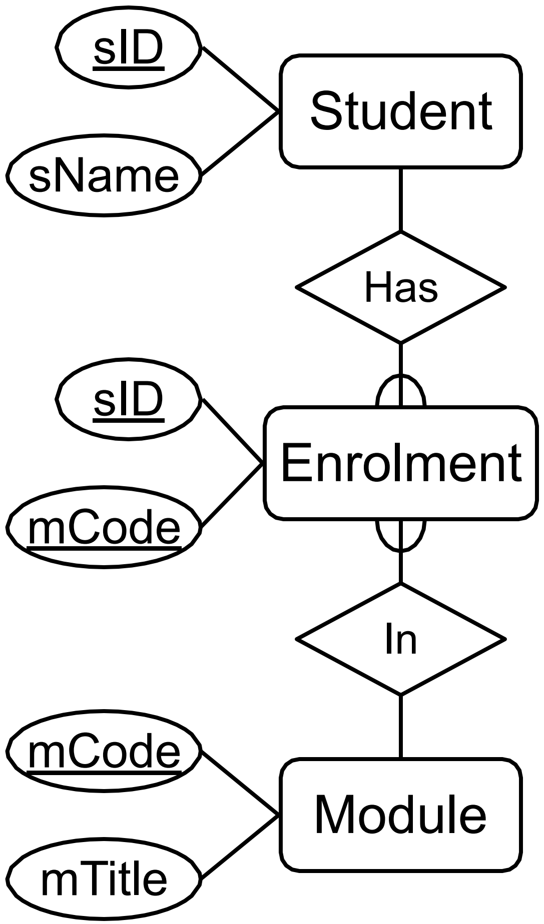
- In MySQL, double quotes also work (this isn't the ANSI standard)

Non-Procedural Programming

- SQL is a declarative (non-procedural) language
  - Procedural – tell the computer what to do using specific successive instructions
  - Non-procedural – describe the required result (not the way to compute it)

- Example: Given a database with tables
  - Student with attributes sID, sName
  - Module with attributes mCode, mTitle
  - Enrolment with attributes sID, mCode
- Get a list of students who take the module 'Database Systems'

Example



1. Find the instance of Module with the title 'Database Systems'
2. Find instances of the Enrolment entity with the same Code as the result of (1)
3. For each instance of Enrolment in the result of (2) find the corresponding student

Procedural Programming

```
Set M to be the first Module Record                  /* Find module code for    */
Code = ''                                            /* 'Database Systems'      */
While (M is not null) and (Code = '')
   If (M.Title = 'Database Systems') Then
      Code = M.Code
   Set M to be the next Module Record
Set NAMES to be empty                                /* A list of student names */
Set S to be the first Student Record
While S is not null                                  /* For each student...     */
   Set E to be the first Enrolment Record
    While E is not null                              /* For each enrolment... */
      If (E.ID = S.ID) And                           /* If this student is    */
         (E.Code = Code) Then                        /* enrolled in DB Systems
                                                         */
         NAMES = NAMES + S.NAME
      Set E to be the next Enrolment Record          /* add them to the list  */
   Set S to be the next Student Record
Return NAMES
```

Non-Procedural (SQL)

```
SELECT sName FROM Student, Enrolment
  WHERE
  (Student.sID = Enrolment.sID)
  AND
  (Enrolment.mCode =
    (SELECT mCode FROM Module WHERE
       mTitle = ‘Database Systems’));
```

Relations, Entities and Tables

- The terminology changes from the Relational Model through to SQL, but usually means the same thing

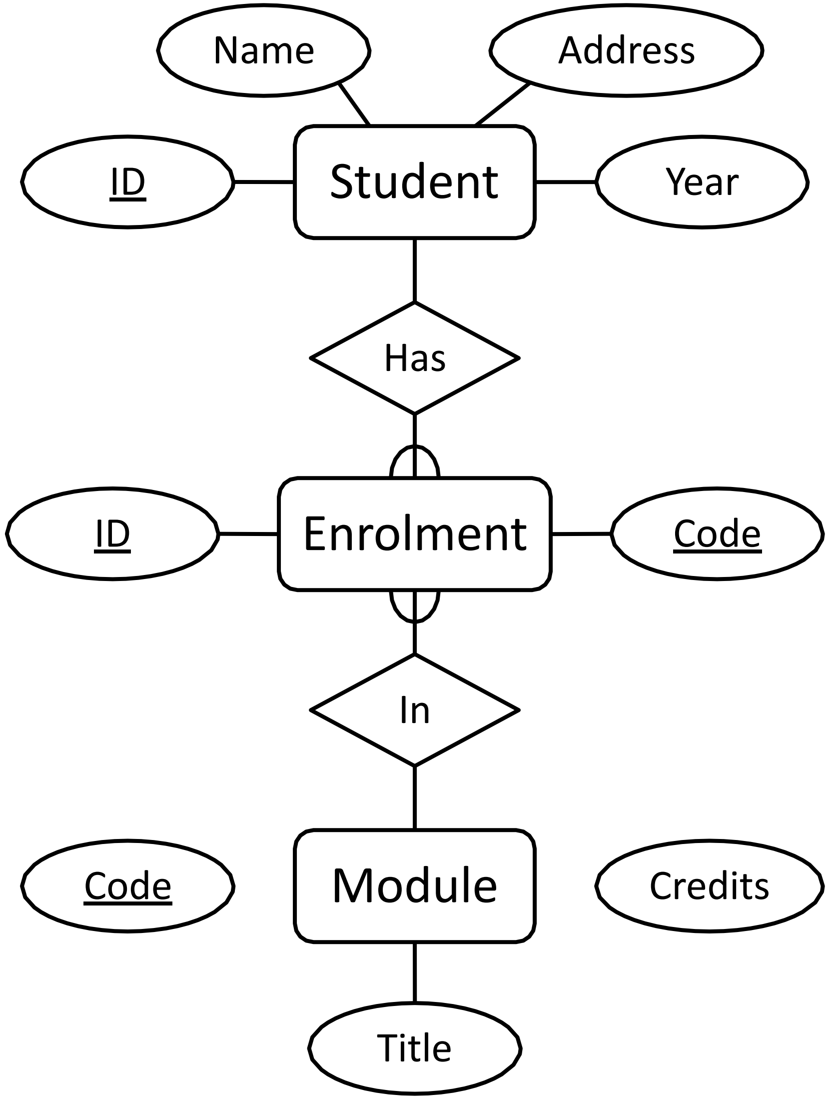
Relations, Entities and Tables

- The terminology changes from the Relational Model through to SQL, but usually means the same thing

Relations	E/R Diagrams	SQL
Relation	Entity	Table
Tuple	Instance	Row
Attribute	Attribute	Column or Field
Foreign Key	M:1 Relationship	Foreign Key
Primary Key	<u>Attribute</u>	Primary Key

Implementing E/R Diagrams

- Given an E/R design
  - The entities become SQL tables
  - Attributes of an entity become columns in the corresponding table
  - We can approximate the domains of the attributes by assigning types to each column
  - Relationships may be represented by **foreign keys**



CREATE DATABASE

- First, we need to create a database

```
CREATE DATABASE database-name;
```

CREATE TABLE (LEFT HERE)

```
CREATE TABLE table-name (
  col-name-1 col-def-1,
  col-name-2 col-def-2,
             :
  col-name-n col-def-n,
  constraint-1,
             :
  constraint-k
);
```

- You supply
  - A name for the table
  - A name and definition / type for each column
  - A list of constraints (e.g. Keys)

Column Definitions

```
col-name col-def
  [NULL | NOT NULL]
  [DEFAULT default_value]
  [NOT NULL | NULL]
  [AUTO_INCREMENT]
  [UNIQUE [KEY] |
  [PRIMARY] KEY]
```

([] *optional*, | *or*)

- Each column has a name and a type
- Most of the rest of the column definition is optional
- There’s more you can add, like storage and index instructions

Types

- There are many types in MySQL, but most are variations of the standard types
- Numeric Types
  - TINYINT, SMALLINT, INT, MEDIUMINT, BIGINT
  - FLOAT, REAL, DOUBLE, DECIMAL
- Dates and Times
  - DATE, TIME, YEAR
- Strings
  - CHAR, VARCHAR
- Others
  - ENUM, BLOB

Types

- We will use a small subset of the possible types:

Type	Description	Example
TINYINT	8 bit integer	-128 to 127
INT	32 bit integer	-2147483648 to 2147483647
CHAR (m)	String of fixed length m	“Hello World ”
VARCHAR (m)	String of maximum length m	“Hello World”
REAL	A double precision number	3.14159
ENUM	A set of specific strings	(‘Cat’, ‘Dog’, ‘Mouse’)
DATE	A Day, Month and Year	‘1981-12-16’ or ‘81-12-16’

Column Definitions

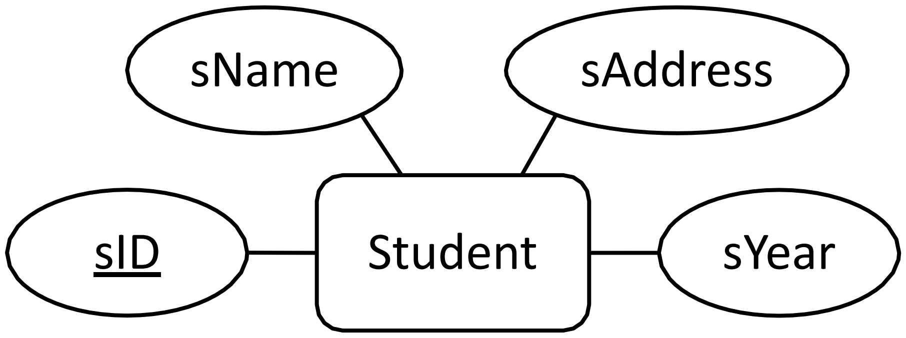
- Columns can be specified as **`NULL`** or **`NOT NULL`**
- **`NOT NULL`** columns cannot have missing values
- **`NULL`** is the default if you do not specify either

- Columns can be given a default value
- You just use the keyword **`DEFAULT`** followed by the value, e.g.:

```
col-name INT DEFAULT 0,
```

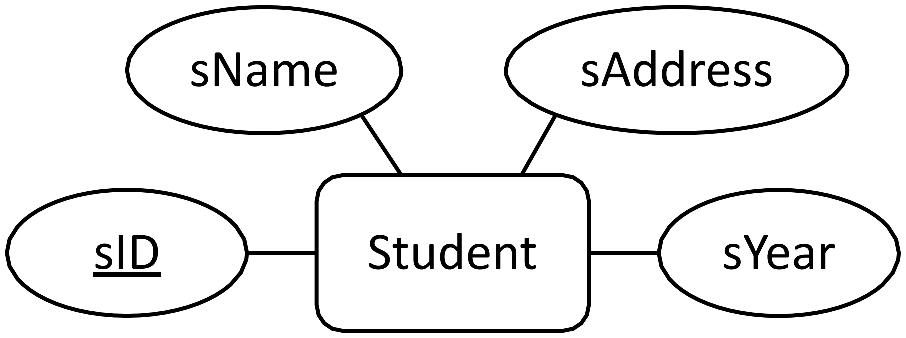
Example

- Write the SQL statement to create a table for Student with the attributes listed below, where the sID number and the Student na cannot be null and, if not otherwise specified, students are in Yea



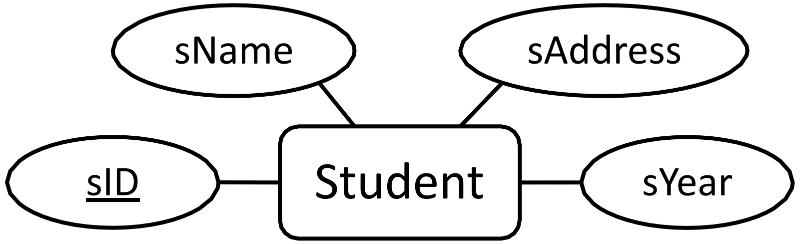
Example

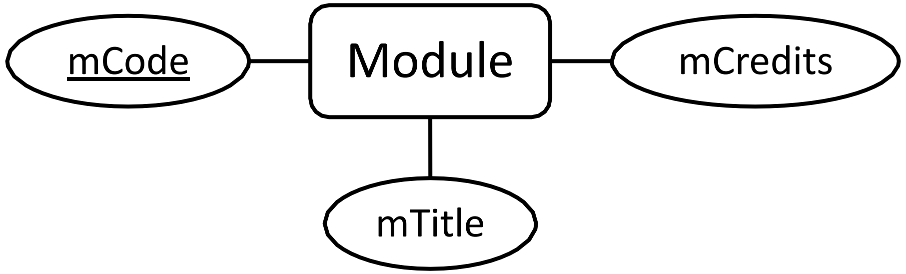
```
CREATE TABLE Student (
  sID INT NOT NULL,
  sName VARCHAR(50) NOT NULL,
  sAddress VARCHAR(255),
  sYear INT DEFAULT 1
);
```



Example

```
CREATE TABLE Student (
  sID INT NOT NULL
   AUTO_INCREMENT,
  sName VARCHAR(50) NOT NULL,
  sAddress VARCHAR(255),
  sYear INT DEFAULT 1
);

CREATE TABLE Module (
  ...
);
```





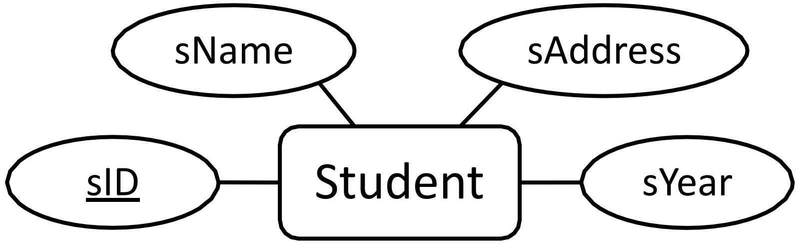
Tips:

-Every module has a 6 characters code (e.g. G64DBS)

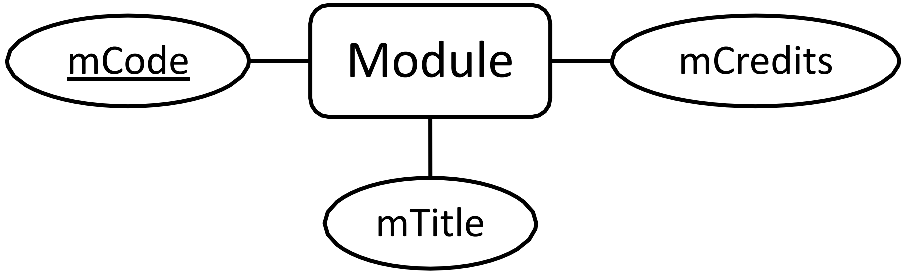
-Every module usually gives 10 credits

Example

```
CREATE TABLE Student (
  sID INT NOT NULL
   AUTO_INCREMENT,
  sName VARCHAR(50) NOT NULL,
  sAddress VARCHAR(255),
  sYear INT DEFAULT 1
);
```




```
CREATE TABLE Module (
  mCode CHAR(6) NOT NULL,


  mCredits TINYINT NOT NULL
   DEFAULT 10,
  mTitle VARCHAR(100) NOT
); NULL
```



Constraints

```
CONSTRAINT
  name
  type
  details
```

- SQL Constraints
  - **PRIMARY KEY**
  - **UNIQUE**
  - **FOREIGN KEY**

- Each constraint is given a name. If you don’t specify a name, one will be generated
- Constraints which refer to single columns can be included in their definition

Primary Keys

- A primary key for each table is defined through a constraint
- **`PRIMARY KEY`** also automatically adds **`UNIQUE`** and **`NOT NULL`** to the relevant column definition

- The details for the Primary Key constraint are the set of relevant columns

```
CONSTRAINT name
   PRIMARY KEY
    (col1, col2, …)
```

Unique Constraints / CKs

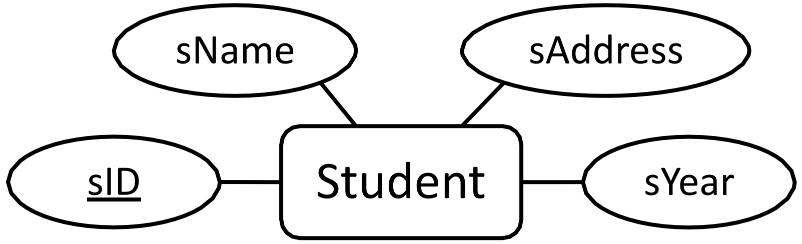
- As well as a single primary key, any set of columns can be specified as **`UNIQUE`**
- This has the effect of making candidate keys in the table

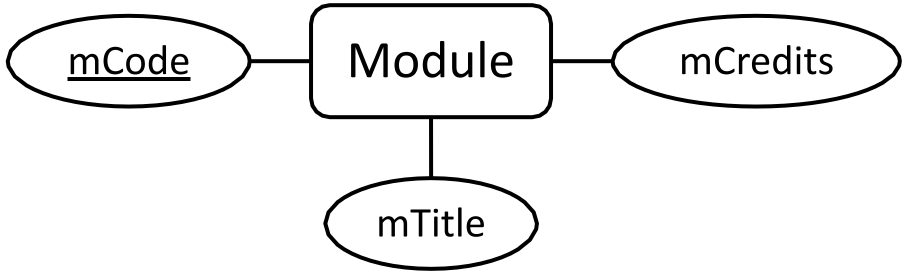
- The details for a unique constraint are a list of columns which make up the candidate key (CK)

```
CONSTRAINT name
  UNIQUE
  (col1, col2, …)
```

Example

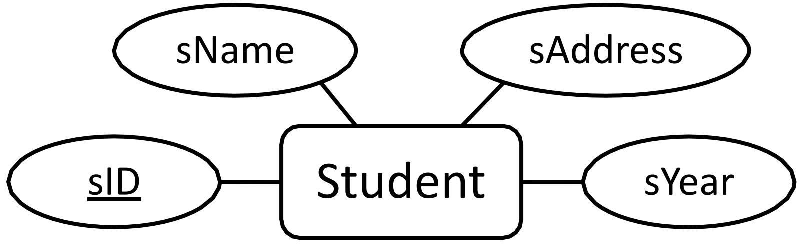
```
CREATE TABLE Student (
  sID INT AUTO_INCREMENT
   PRIMARY KEY,
  sName VARCHAR(50) NOT NULL,
  sAddress VARCHAR(255),
  sYear INT DEFAULT 1
);

CREATE TABLE Module (
  mCode CHAR(6) NOT NULL,
  mCredits TINYINT NOT NULL
   DEFAULT 10,
  mTitle VARCHAR(100) NOT
   NULL,
  ... ADD PRIMARY KEY
);
```

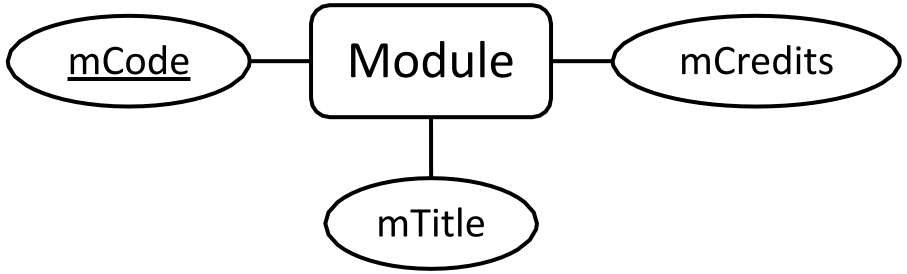




Example

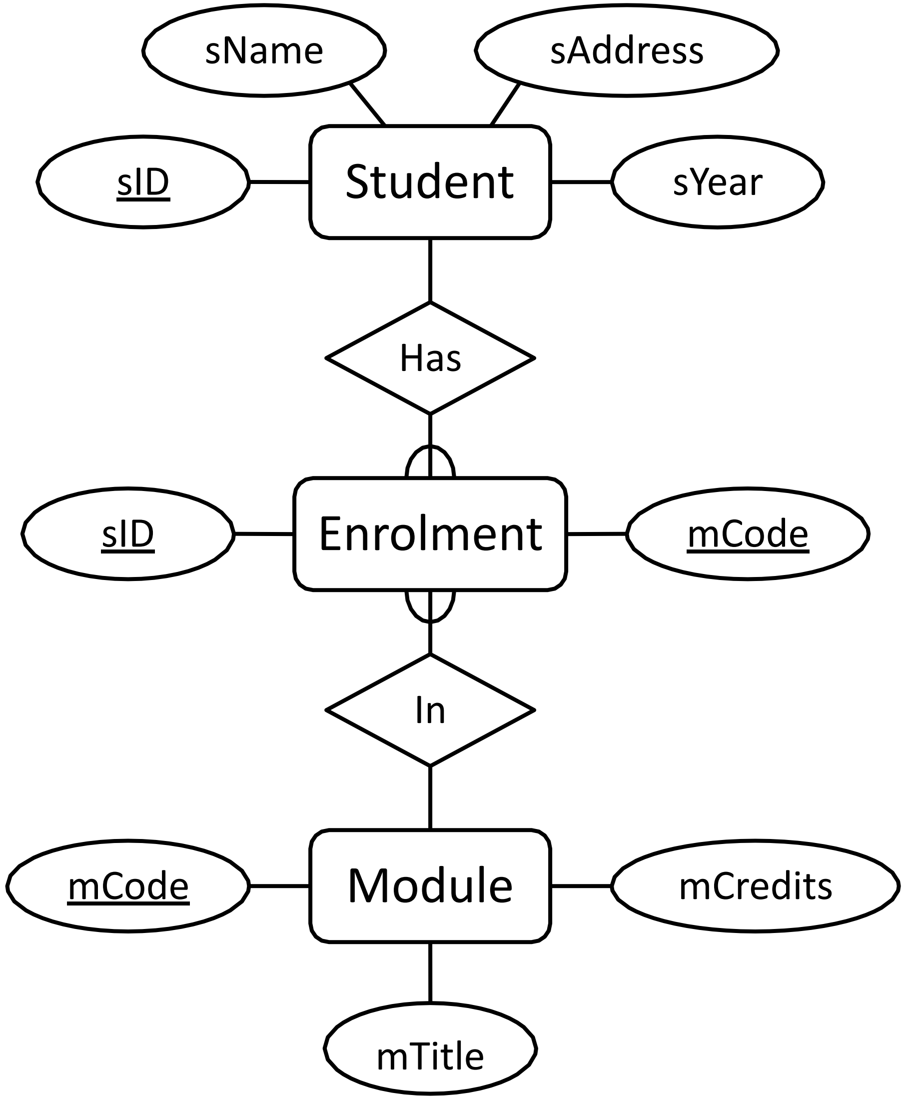
```
CREATE TABLE Student (
  sID INT AUTO_INCREMENT
   PRIMARY KEY,
  sName VARCHAR(50) NOT NULL,
  sAddress VARCHAR(255),
  sYear INT DEFAULT 1
);

CREATE TABLE Module (
  mCode CHAR(6) NOT NULL,
  mCredits TINYINT NOT NULL
   DEFAULT 10,
  mTitle VARCHAR(100) NOT
   NULL,
  CONSTRAINT mod_pk
   PRIMARY KEY (mCode)
);
```





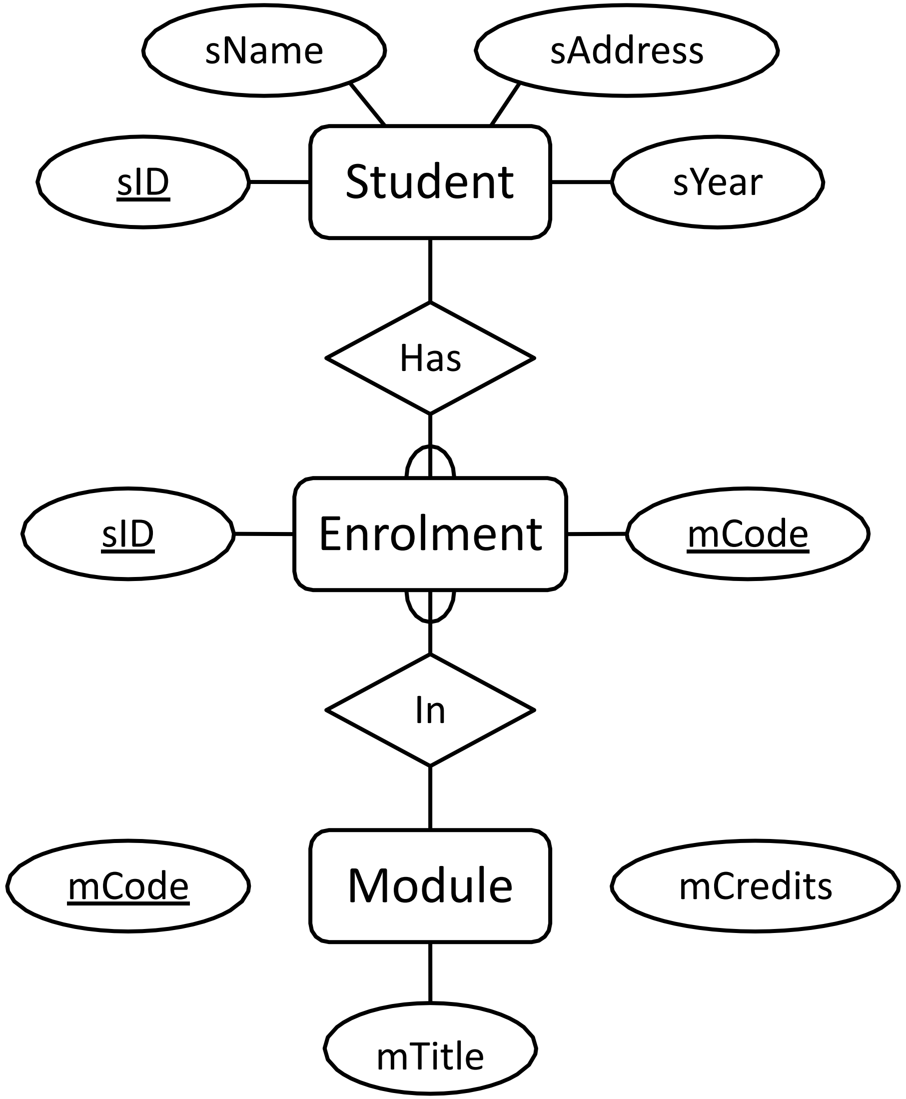
Relationships

- Relationships are represented in SQL using Foreign Keys
  - 1:1 are usually not used, or can be treated as a special case of M:1
  - M:1 are represented as a foreign key from the M-side to the 1
  - M:M are split into two M:1 relationships



Relationships

- The Enrolment table
  - Will have columns for the student ID and module code attributes
  - Will have a foreign key to Student for the ‘has’ relationship
  - Will have a foreign key to Module for the ‘in’ relationship



Foreign Keys

- Foreign Keys are also defined as constraints
- You need to provide
  - The columns which make up the foreign key
  - The referenced table
  - The columns which are referenced by the foreign key
- You can optionally provide reference

```
CONSTRAINT name
 FOREIGN KEY
  (col1, col2, ...)
 REFERENCES
  table-name
  (col1, col2, ...)
 ON UPDATE ref_opt
 ON DELETE ref_opt

ref_opt: RESTRICT |
  CASCADE | SET NULL
  | SET DEFAULT
```

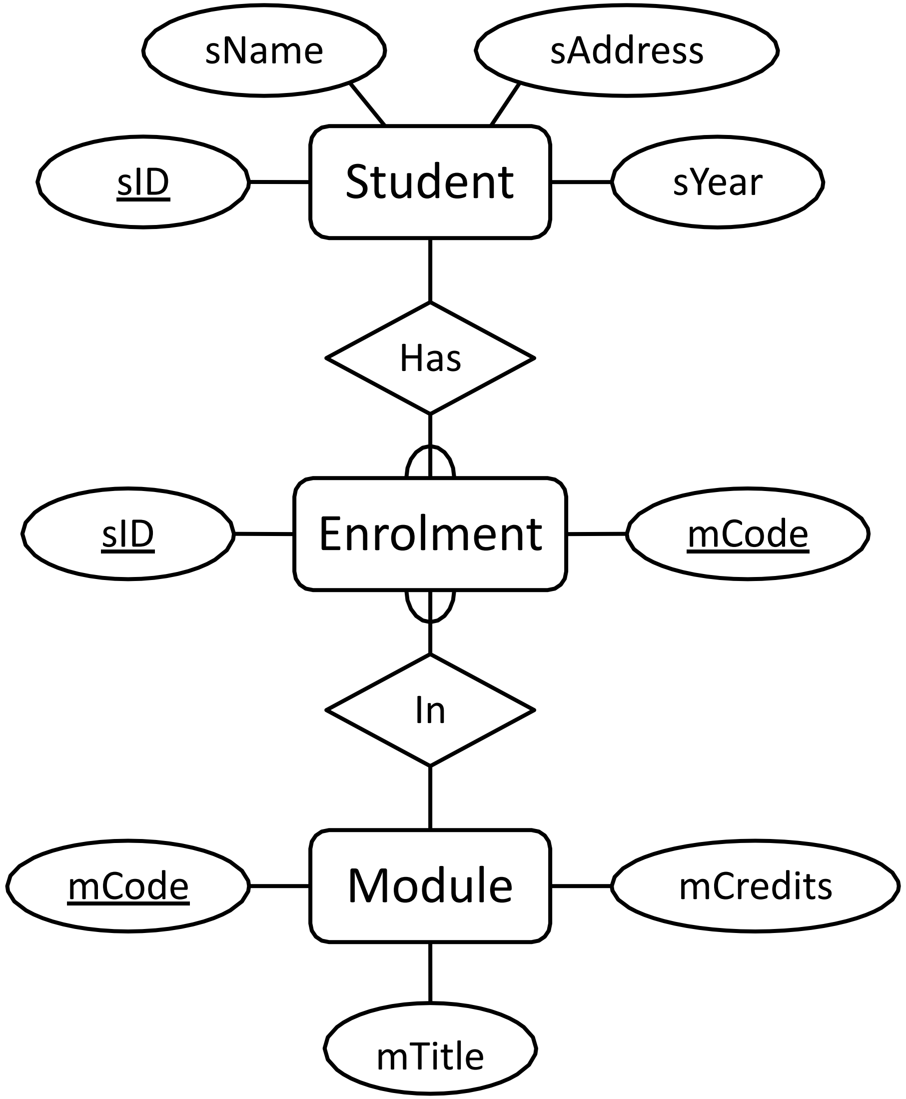
Set Default (Column Definition)

- If you have defined a **`DEFAULT`** value you can use it with referential integrity
- When relations are updated, referential integrity might be violated
- This usually occurs when a referenced tuple is updated or deleted

- There are a number of options when this occurs:
  - RESTRICT – stop the user from doing it
  - CASCADE – let the changes flow on
  - SET NULL – make referencing values null
  - SET DEFAULT – make referencing values the default for their column

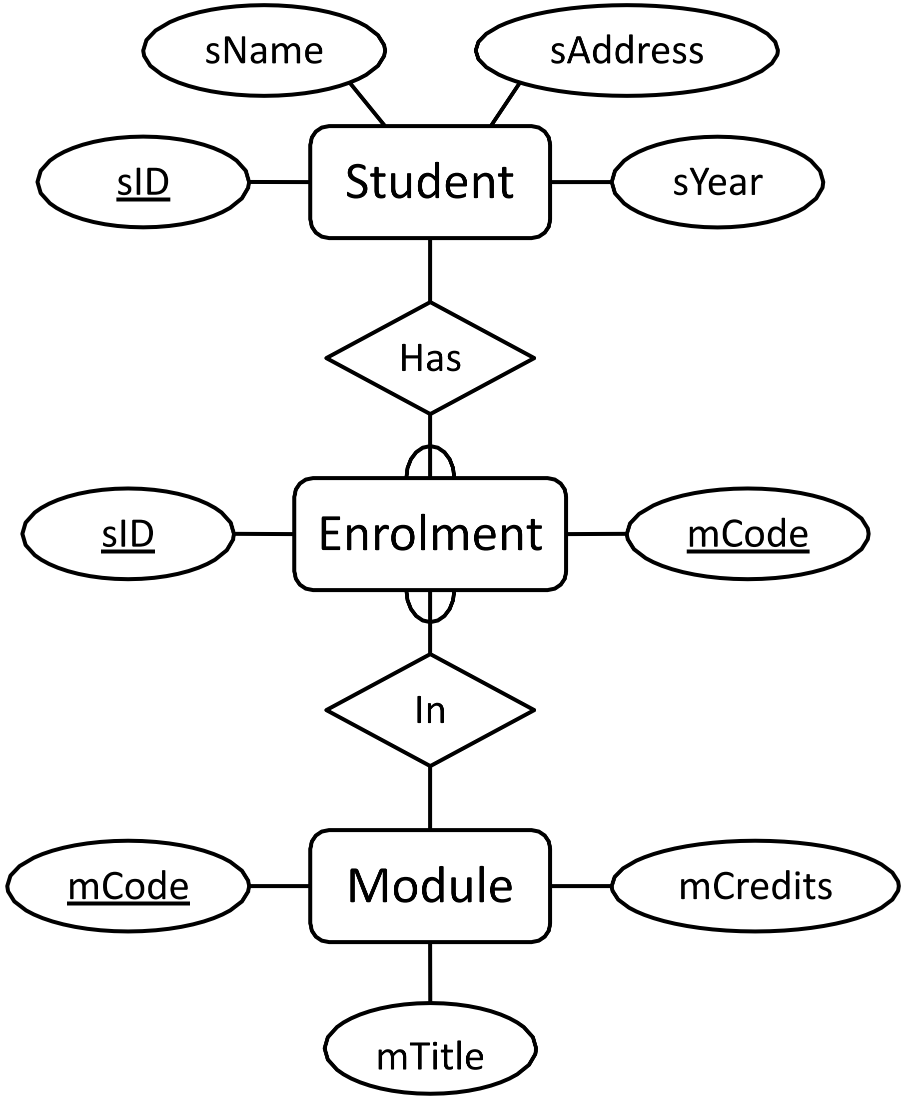
Example

```
CREATE TABLE Enrolment (
  sID INT NOT NULL,
  mCode CHAR(6) NOT NULL,
  ... ADD PRIMARY KEY
  ... AND 2 FOREIGN KEYS
);
```



Example

```
CREATE TABLE Enrolment (
  sID INT NOT NULL,
  mCode CHAR(6) NOT NULL,
  CONSTRAINT en_pk
    PRIMARY KEY (sID, mCode),
  CONSTRAINT en_fk1
    FOREIGN KEY (sID)
    REFERENCES Student (sID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  CONSTRAINT en_fk2
    FOREIGN KEY (mCode)
    REFERENCES Module (mCode)
    ON UPDATE CASCADE
    ON DELETE NO ACTION
);
```



Storage Engines

- In MySQL you can specify the engine used to store files onto disk
- The type of storage engine will have a large effect on the operation of the database
- The engine should be specified when a table is created

- Some available storage
- engines are:
  - **MyISAM** – The default, very fa
    Ignores all foreign key constraints
  - **InnoDB** – Offers transactions a
    foreign keys
  - **Memory** – Stored in RAM (extremely fast)
  - **Others**

InnoDB

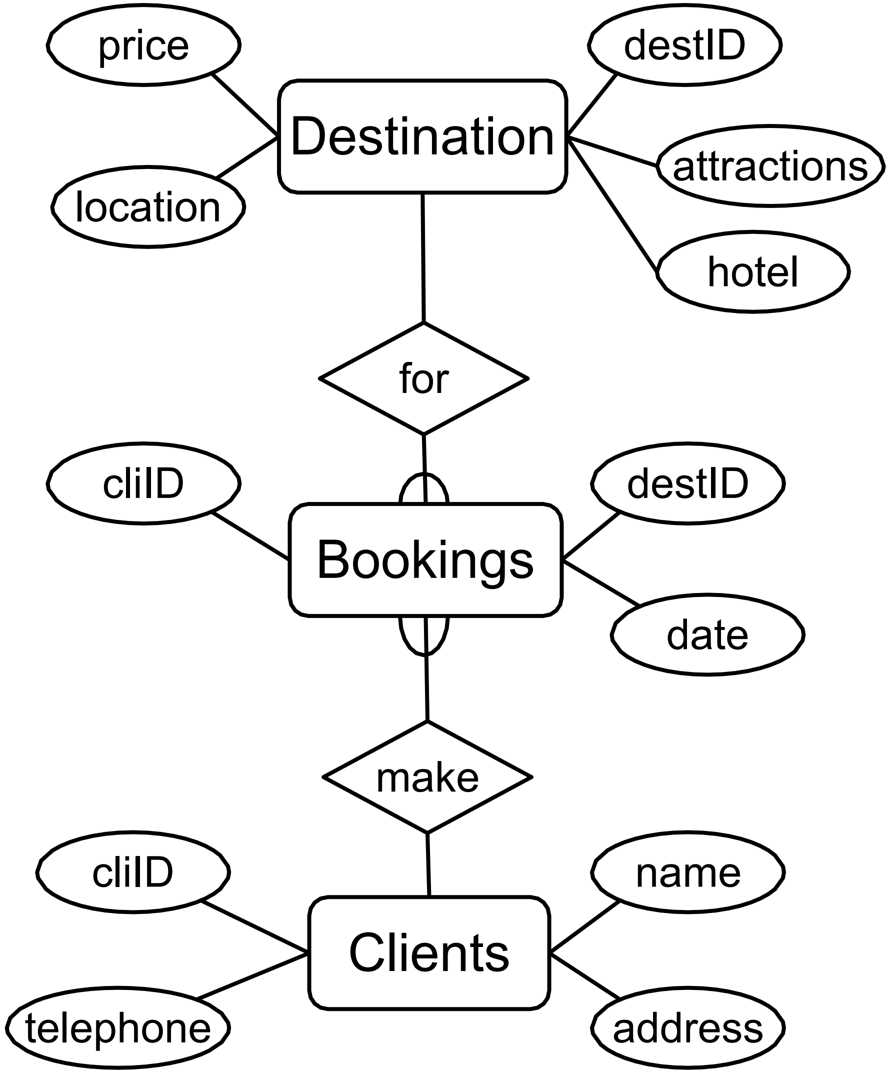
- We will use InnoDB for all tables during this module, for example:

```
CREATE TABLE Student (
  sID INT AUTO_INCREMENT PRIMARY
  KEY,  sName VARCHAR(50) NOT NULL,
  sAddress VARCHAR(255),
  sYear INT  DEFAULT
) ENGINE1
        = InnoDB;
```

Note: All tables in a relationship must be InnoDB for FK constraints to work

Exercise

- Create table in MySQL from the E/R diagram on the right by identifying the:
  - Name of the tables
  - The columns (inc. data types and attributes) for each table
  - Each table's constraints



Solutions (1)

```
CREATE DATABASE Holiday;
use Holiday;
CREATE TABLE Clients(
  cliID INT PRIMARY KEY AUTO_INCREMENT,
  cliName varchar(255) NOT NULL,
  cliAddress varchar(255),
  cliTel INT
) engine=InnoDB;

CREATE TABLE Destination(
  destID INT PRIMARY KEY AUTO_INCREMENT,
  destLocation VARCHAR(255),
  destPrice REAL,
  destHotel VARCHAR(255),
  destAttractions VARCHAR(255)
) ENGINE=InnoDB;
```

Solutions (2)

```
CREATE TABLE Bookings(
  cliID INT NOT NULL,
  destID INT NOT NULL,
  bookDate DATE,
  CONSTRAINT book_pk PRIMARY KEY(cliID,destID),
  CONSTRAINT book_fk1 FOREIGN KEY (cliID)
  REFERENCES Clients (cliID)
  ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT book_fk2 FOREIGN KEY (destID)
  REFERENCES Destination (destID)
  ON UPDATE          ON DELETE CASCADE
  CASCADE
) ENGINE=InnoDB;
```

NoSQL

- SQL is by no means perfect
  - Edgar Codd hated it – It’s actually a pretty poor implementation of the relational model
  - Implementations vary wildly. For example, while Oracle and MySQL both use SQL, there are commands that won’t work on both systems.
  - It’s extremely easy to trigger vast joins or delete large numbers of rows by mistake
- NoSQL is a term used to describe database systems that attempt to avoid SQL and the relational model

This Lecture in Exams

Give the SQL statement(s) required to create a table called Books with the following columns

- bID, an integer that will be the Primary Key
- bTitle, a string of maximum length 64
- bPrice, a double precision value
- gCode, an integer that will be a foreign key to a gCode column in another table Genres

Take home messages

1. SQL - Structured Query Language
2. SQL provide DBMS Languages
3. SQL – Non procedural language
4. We use MySQL as DBMS
5. Create
   a. Database and Tables
   b. Data types / column definition
   c. Constraints (Primary and Foreign keys)

Lab on Thursday

- We’ll start using PCs
- Make sure you know your CS username and password
- **Bring a pen and a piece of paper!!**
  - Automatically generated password will be provided to each of you and will be needed for accessing your database.
  - You can change it, but you will need it first time!

Next Lecture

- More SQL
  - DROP TABLE
  - ALTER TABLE
  - INSERT, UPDATE, and DELETE
  - The Information Schema
- For more information
  - Database Systems, Connolly and Begg, Chapter 6.3
  - The Manga Guide to Databases, Chapter 4

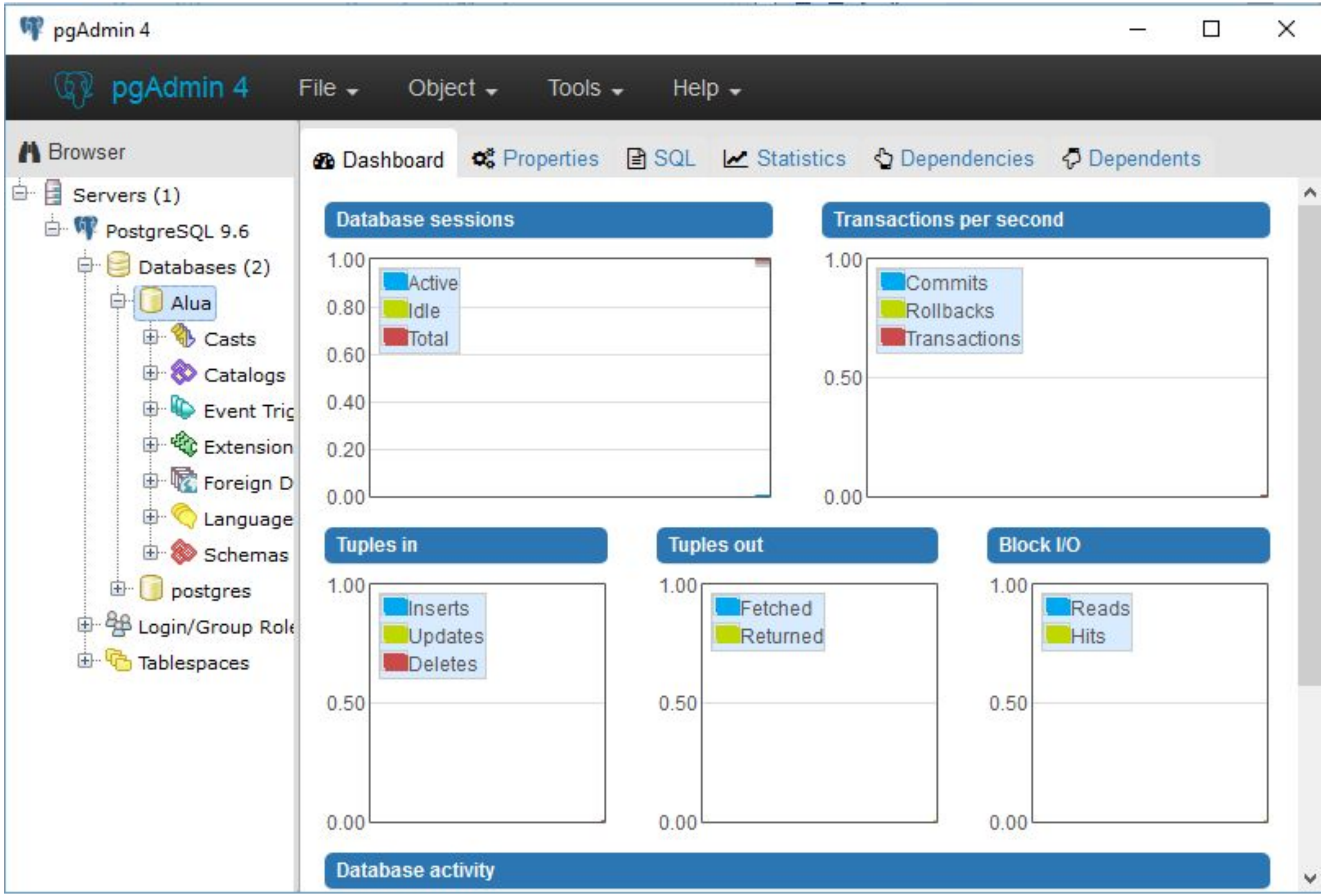
SQL Data Definition II

DBS – Database Systems

Install PostgreSQL on your machine

- Go to http://www.enterprisedb.com/products-services-training/pgdownload#windows
- Select "Download"
- Install PostgreSQL
  - If prompted, select a username and password
  - Please remember your password! You will need it always

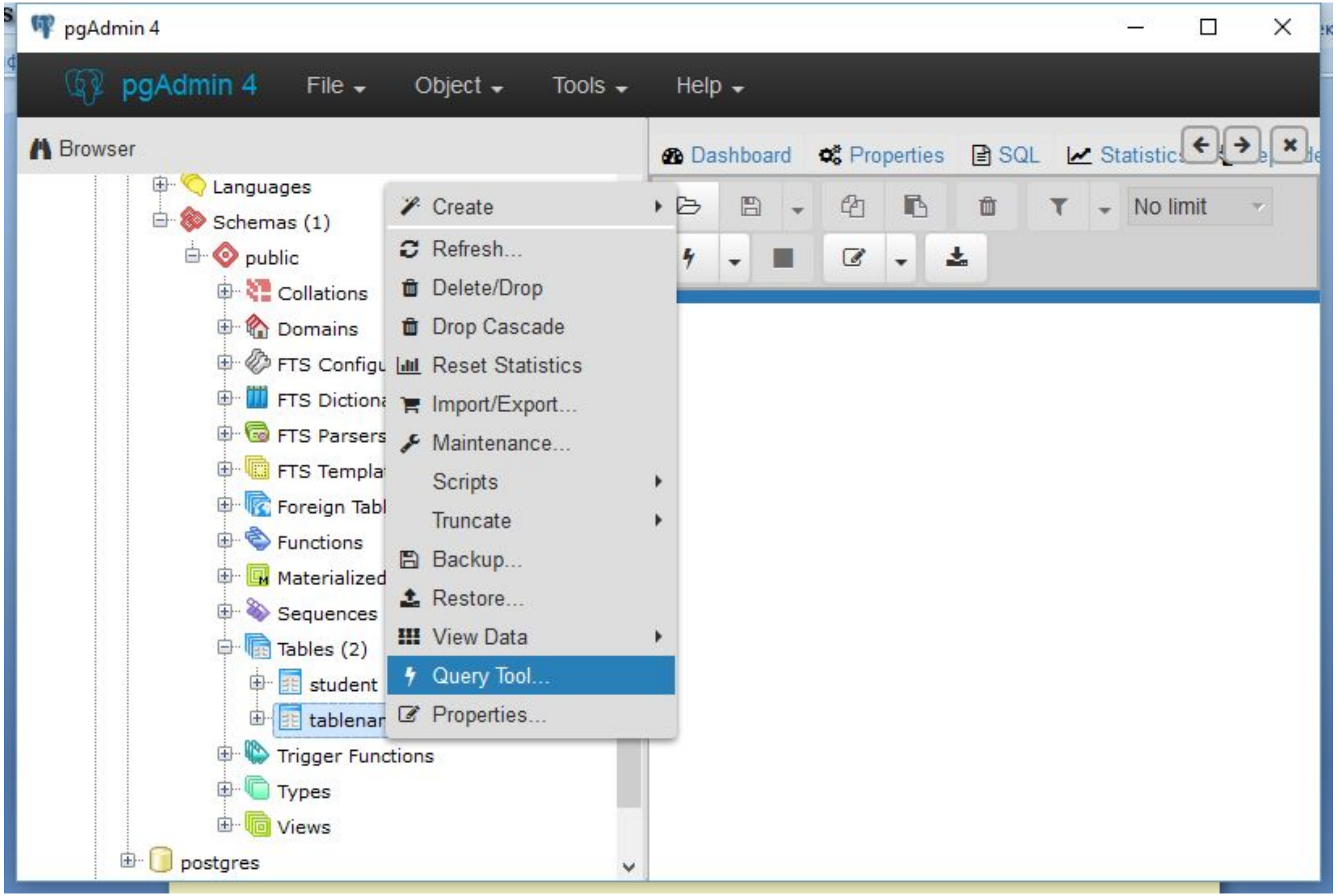
How to get started with Workbench



This Lecture

- More SQL
  - DROP TABLE
  - ALTER TABLE
  - INSERT, UPDATE, and DELETE
  - The Information Schema
- Further Reading
  - Database Systems, Connolly and Begg, Chapter 6.3
  - The Manga Guide to Databases, Chapter 4

How to find Query tool



Notice

- Postgre SQL do NOT save your code,
- Save your SQL code every time
- **Auto_increment PostgreSQL**
  - First, you need to create table
  - CREATE TABLE tablename (
  - colname SERIAL);

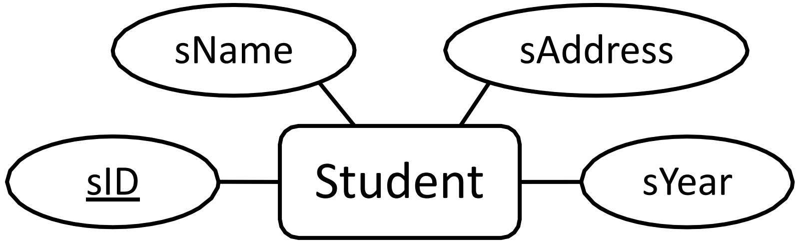
  - Second,
  - CREATE TABLE Student (
  - ID SERIAL PRIMARY KEY,
  - NAME Varchar (50) NOT NULL);

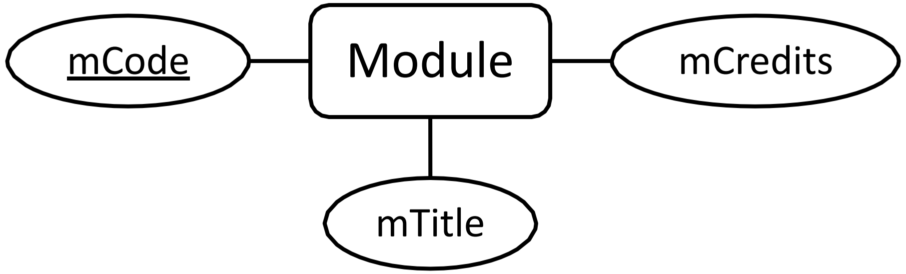
Last Lecture - CREATE TABLE

```
CREATE TABLE table-name (
  col-name-1 col-def-1,
  col-name-2 col-def-2,
            :
  col-name-n col-def-n,
  constraint-1,
            :
  constraint-k
);
```

Last Lecture

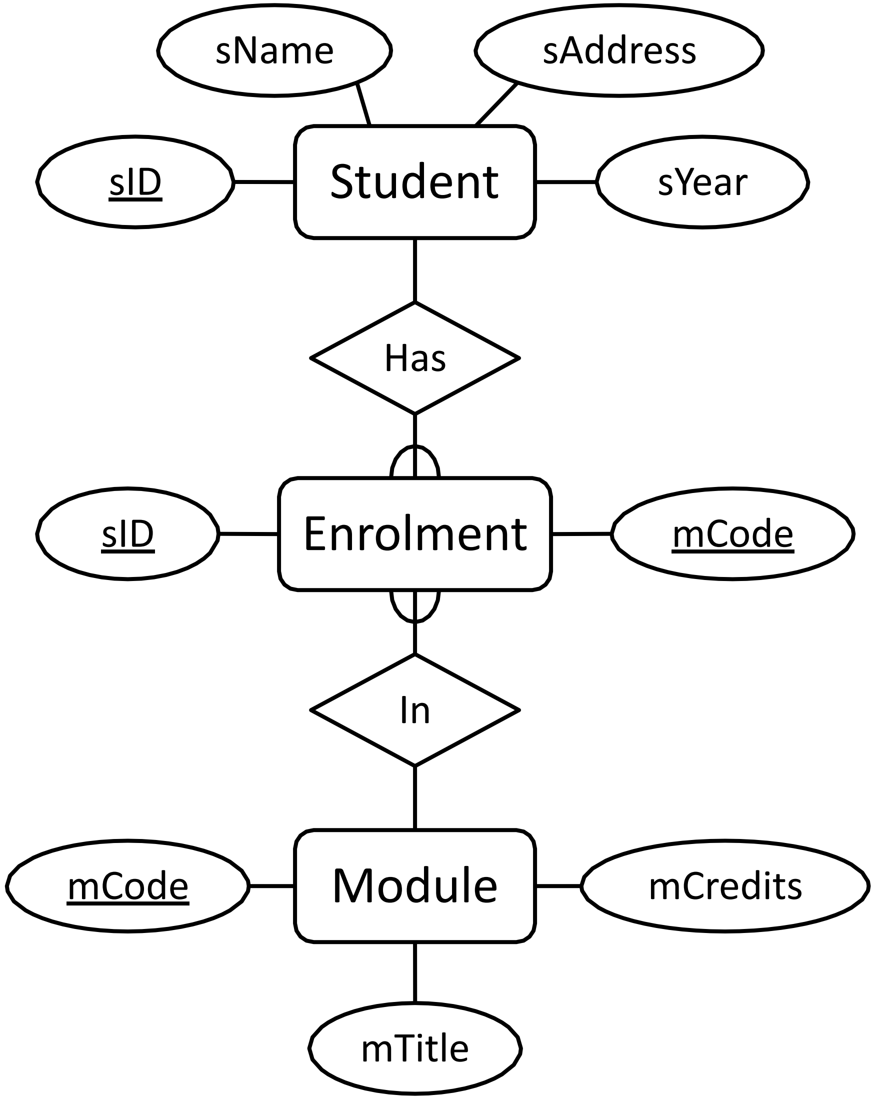
```
CREATE TABLE Student (
  sID INT PRIMARY KEY,
  sName VARCHAR(50) NOT NULL,
  sAddress VARCHAR(255),

  sYear INT DEFAULT 1
) ;

CREATE TABLE Module (
  mCode CHAR(6) NOT NULL,
  mCredits TINYINT NOT NULL
   DEFAULT 10,
  mTitle VARCHAR(100) NOT
   NULL,
  CONSTRAINT pk_mod
  PRIMARY KEY (mCode)
);
```





Last Lecture

```
CREATE TABLE Enrolment (
  sID INT NOT NULL,
  mCode CHAR(6) NOT NULL,
  CONSTRAINT en_pk
    PRIMARY KEY (sID, mCode),
  CONSTRAINT en_fk1
    FOREIGN KEY (sID)
    REFERENCES Student (sID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  CONSTRAINT en_fk2
    FOREIGN KEY (mCode)
    REFERENCES Module (mCode)
    ON UPDATE CASCADE
    ON DELETE NO ACTION
);
```



Another way

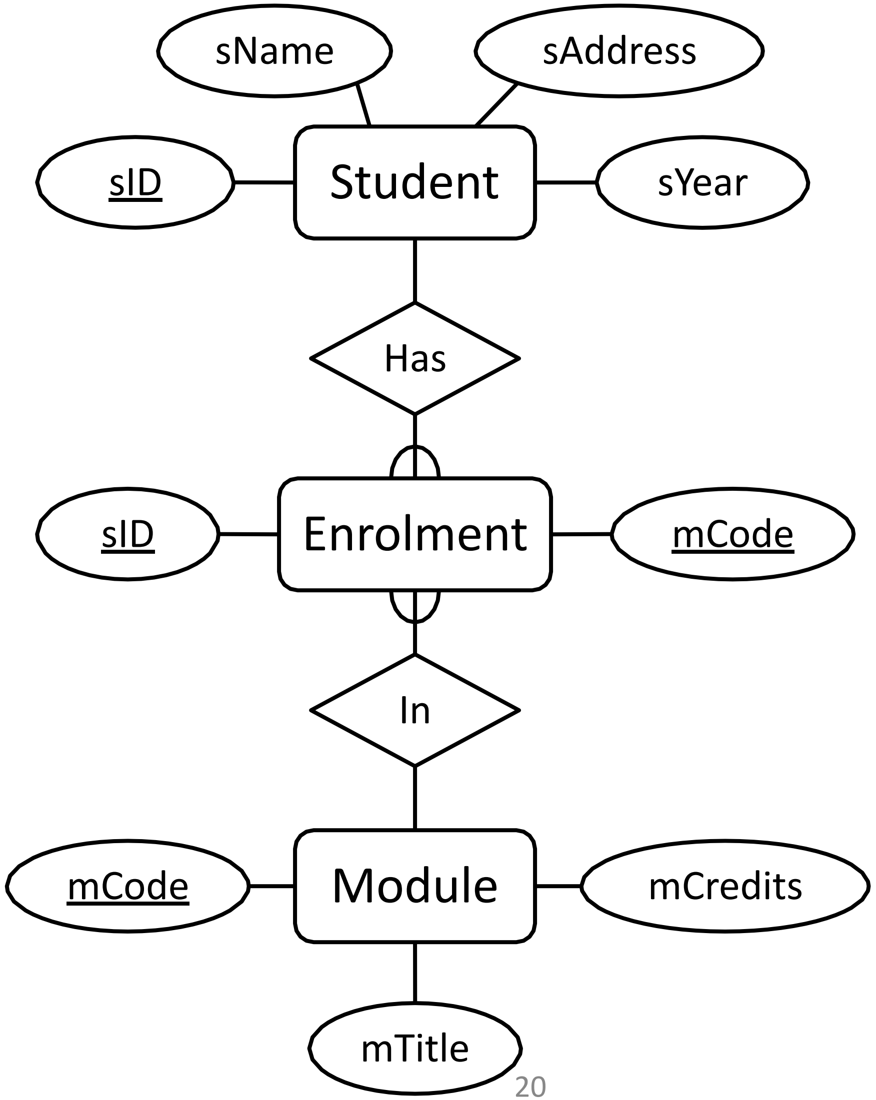
Tables	Attributes
Student	<u>sID</u>, sName, sAddress, sYear
Module	<u>mCode</u>, mTitle, mCredits
Enrolment	<u>sID</u>, <u>mCode</u>

Table (Foreign Keys)	References
Enrolment (sID)	Student (sID)
Enrolment (mCode)	Module (mCode)

Another way

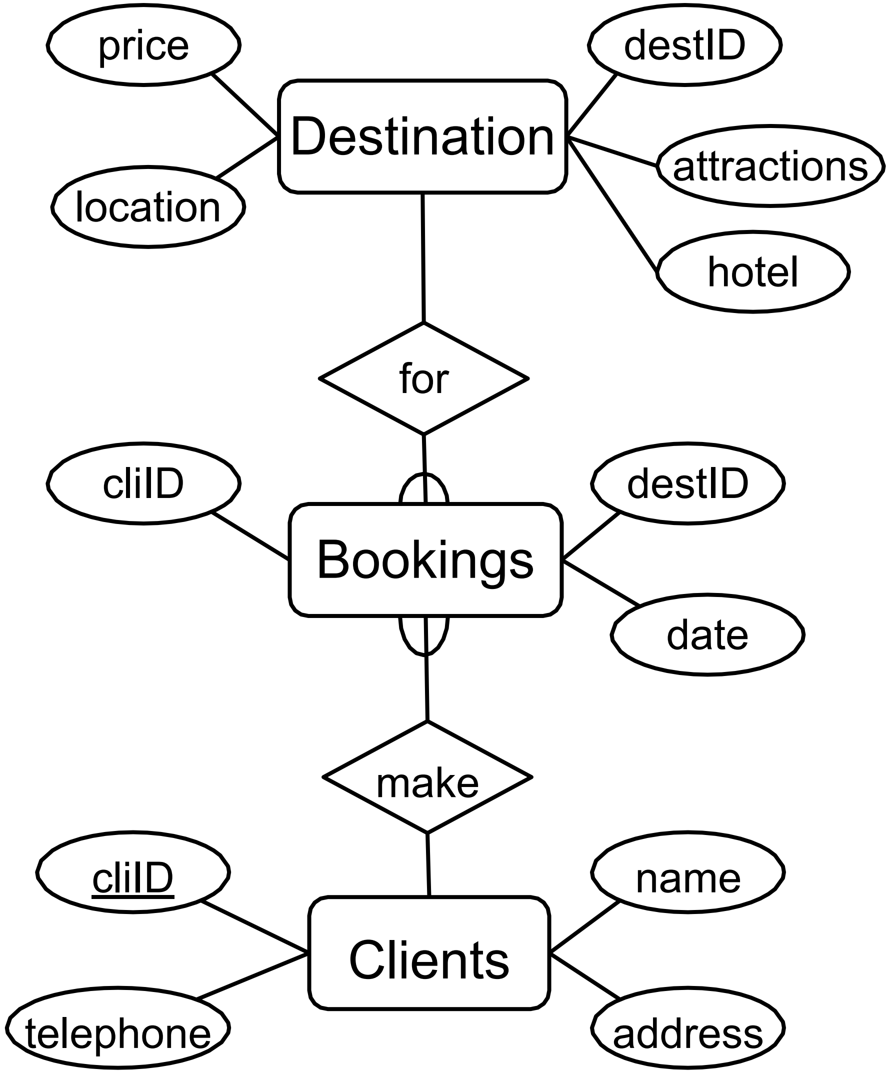
Tables	Attributes
Student	sID, sName, sAddress, sYear
Module	mCode, mTitle, mCredits
Enrolment	sID, mCode

Table (Foreign Keys)	References
Enrolment (sID)	Student (sID)
Enrolment (mCode)	Module (mCode)



Exercise

- Create table in PostgreSQL from the E/R diagram on the right by identifying the:
  - Name of the tables
  - The columns (inc. data types and attributes) for each table
  - Each table's constraints

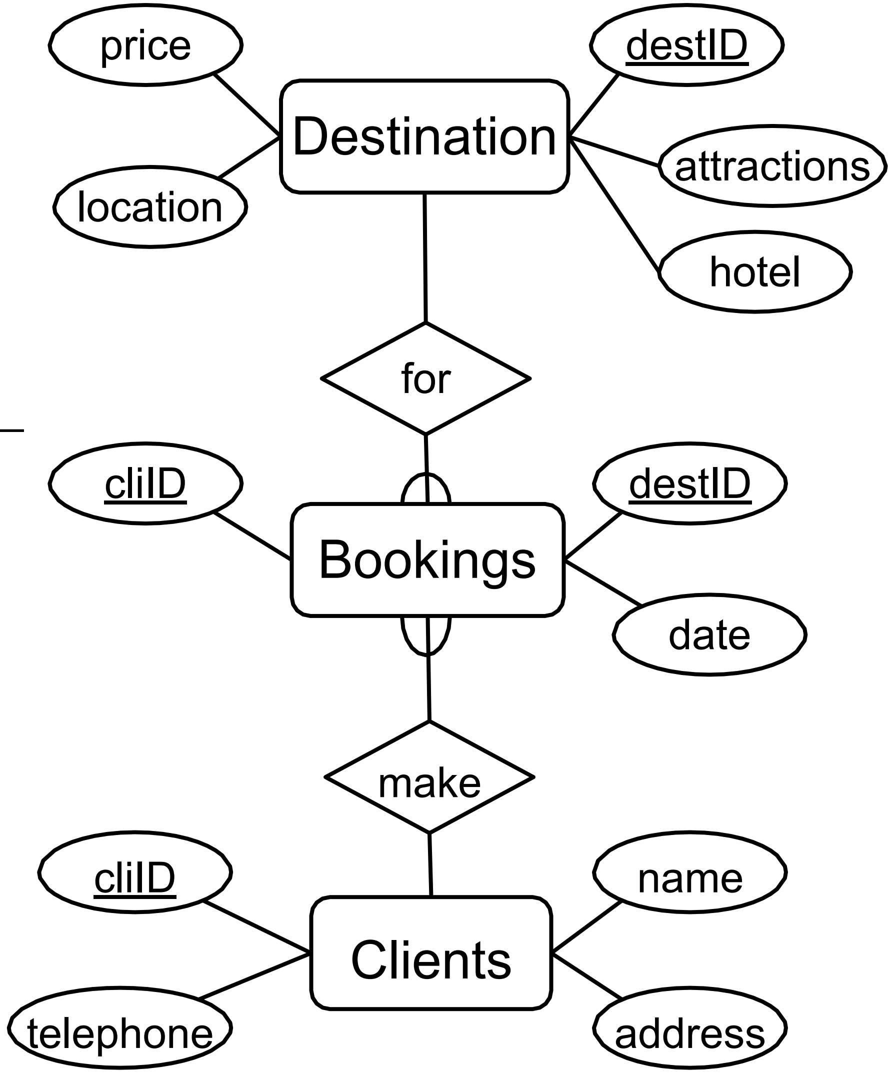


Exercise

- Represent the tables, attributes and relationships from the E/R diagram on the right by completing the following:

Tables	Attributes

Table (Foreign Keys)	References

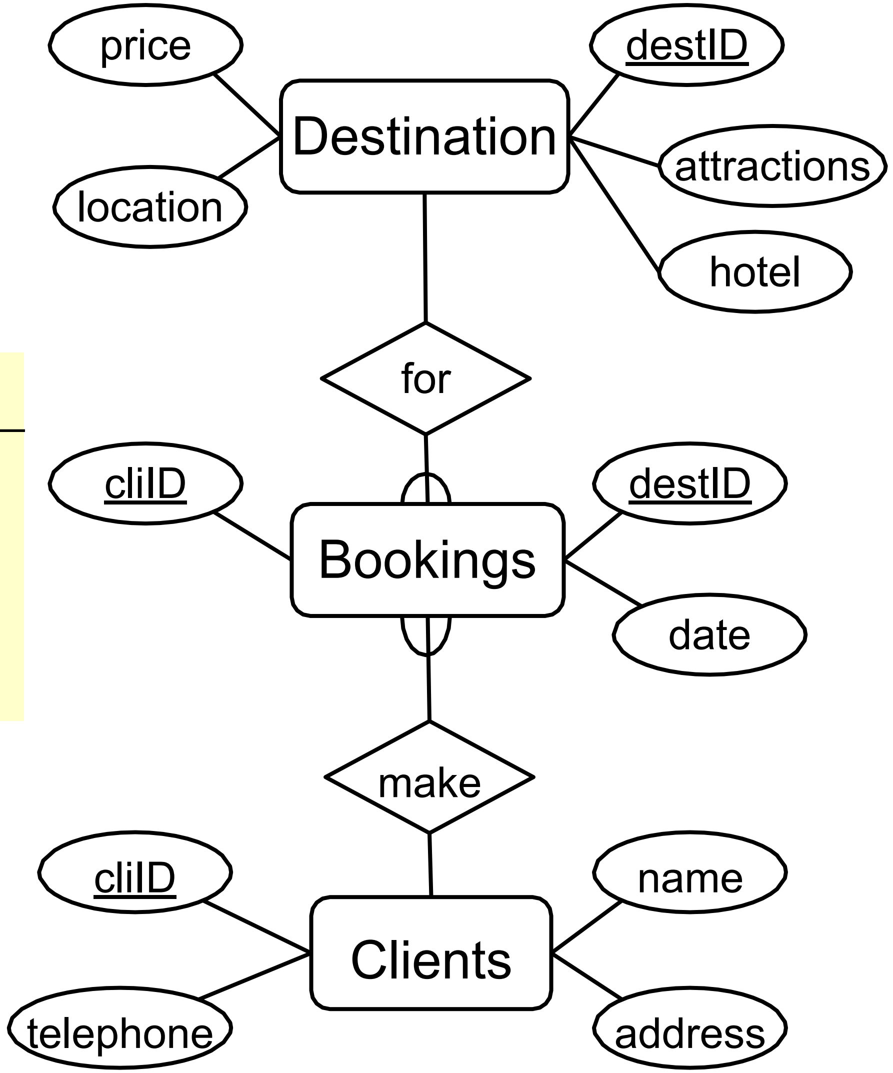


Exercise

- Represent the tables, attributes and relationships from the E/R diagram on the right by completing the following:

Tables	Attributes
Clients	cliID, name, address, telephone
Destination	destID, location, hotel, price, attractions
Bookings	cliID, destID, date

Table (Foreign Keys)	References
Booking (cliID)	Clients (ID)
Booking (destID)	Destination (destID)



Deleting Tables

- You can delete tables with the DROP keyword

```
DROP TABLE
 [IF EXISTS]
 table-name;
```

- For example:

```
DROP TABLE Module;
```

- Be ***extremely careful*** using any SQL statement with DROP in it.
  - All rows in the table will also be deleted
  - You won’t normally be asked to confirm
  - Undoing a DROP is difficult, sometimes impossible

Deleting Tables

- You can delete multiple tables in a list:

```
DROP TABLE
 IF EXISTS
 Module, Student;
```

- Foreign Key constraints will prevent DROPS under the default RESTRICT option
  - To overcome this, either remove the constraint or drop the tables in the correct order (referencing table first)

Changing Tables

- Sometimes you want to change the structure of an existing table
  - One way is to DROP it then rebuild it
  - This is dangerous, so there is the ALTER TABLE command instead
- ALTER TABLE can
  - Add a new column
  - Remove an existing column
  - Add a new constraint
  - Remove an existing constraint
  - Change column name and/or definition

Altering Columns

- To add a column to a table:

```
ALTER TABLE table-name
 ADD COLUMN col-name
 col-def
```

OR

```
ALTER TABLE table-name
 ADD COLUMN col-name
 FIRST | AFTER col2
```

- To remove a column from a table:

```
ALTER TABLE table-name
 DROP COLUMN col-name
```

- For example:

```
ALTER TABLE Student
 ADD COLUMN sDegree
 VARCHAR(64) NOT NULL;
```

```
ALTER TABLE Student
 DROP COLUMN sDegree;
```

Altering Columns

- To change a column's name (and definition):

```
ALTER TABLE table-name
 CHANGE COLUMN
 col-name
 new-col-name
 col-definition
```

- To change the definition of a column only:

```
ALTER TABLE table-name
 MODIFY COLUMN
 col-name
 new-col-definition
```

Note: Changing the type of a column might have unexpected results.
Be careful that the type conversion taking place is appropriate.
E.g. INT → VARCHAR is ok, VARCHAR → INT is problematic.

Altering Columns - constraints

- To add a constraint:

  - **ALTER TABLE table-name ADD CONSTRAINT**
  - **name**
  - **definition**

- For example:

  - **ALTER TABLE Module ADD CONSTRAINT**
  - **ck_module UNIQUE**
  - **(mTitle)**

- To remove a constraint:

  ALTER TABLE table-name

  ...

Altering Columns - constraints

- To add a constraint:

  - **ALTER TABLE table-name ADD CONSTRAINT**
  - **name**
  - **definition**

- To remove a constraint:

  ALTER TABLE table-name DROP CONSTRAINT name

- For example:

  - **ALTER TABLE Module ADD CONSTRAINT**
  - **ck_module UNIQUE**
  - **(mTitle)**

Altering Columns - constraints

- To add a constraint:

  - `ALTER TABLE table-name ADD CONSTRAINT`
  - `name`
  - `definition`

- For example:

  - `ALTER TABLE Module ADD CONSTRAINT`
  - `ck_module UNIQUE`
  - `(mTitle);`

- To remove a constraint:

~~`ALTER TABLE table-name DROP CONSTRAINT name`~~

- That would be too easy!!

```
ALTER TABLE table-name
 DROP INDEX name |
 DROP FOREIGN KEY name |
 DROP PRIMARY KEY
```

| means OR

Example

```
CREATE TABLE Module (
  mCode CHAR(6) NOT NULL,
  mCredits TINYINT NOT NULL
   DEFAULT 10,
  mTitle VARCHAR(100) NOT NULL
);
```

What are the SQL command(s) to add a column lecID to the Module table? Followed by a foreign key constraint to reference the lecID column in a Lecturer table?

Module

mCode	mCredits	mTitle
G64DBS	10	Database Systems
G51PRG	20	Programming
G51IAI	10	Artificial Intelligence
G52ADS	10	Algorithms

Example

To add a lecID column:

```
ALTER TABLE Module
  ADD COLUMN lecID INT NULL | NOT NULL;
```

Module

mCode	mCredits	mTitle	lecID
G64DBS	10	Database Systems	NULL
G51PRG	20	Programming	NULL
G51IAI	10	Artificial Intelligence	NULL
G52ADS	10	Algorithms	NULL

Example

To create a Foreign Key:

- **ALTER TABLE Module**
  - **ADD CONSTRAINT fk_mod_lec**
  - **FOREIGN KEY (lecID) REFERENCES Lecturer (lecID);**

mCode	mCredits	mTitle	lecID
G64DBS	10	Database Systems	NULL
G51PRG	20	Programming	NULL
G51IAI	10	Artificial Intelligence	NULL
G52ADS	10	Algorithms	NULL

Example

Table Lecturer does NOT exist! So we need to create it first

```
CREATE TABLE Lecturer(
  lecID INT PRIMARY KEY,
  lecName VARCHAR(255) NOT NULL);
```

Then we can create the Foreign Key:

```
ALTER TABLE Module
  ADD CONSTRAINT fk_mod_lec
  FOREIGN KEY (lecID) REFERENCES Lecturer (lecID);
```

INSERT, UPDATE, DELETE

- **`INSERT`** - add a row to
  a table

- **`UPDATE`** - change row(s) in a table

- **`DELETE`** -
  remove row(s)
  from a table

- **`UPDATE`** and **`DELETE`** should make use of ‘**`WHERE`** clauses’ to
  specify which rows to change or remove
- ***BE CAREFUL*** with these - an incorrect or absent **`WHERE`** clause can destroy lots of data

INSERT

- Inserts rows into the database with the specified values

```
INSERT INTO
  table-name
  (col1, col2, …)
  VALUES
  (val1, val2, …);
```

- The number of columns and the number of values must be the same
- If you are adding a value to every column, you don't have to list them
- If you don't list columns, be careful of the ordering

INSERT

ID	Name	Salary
1	John	25000

```
INSERT INTO
 Employee  (ID,
 Name, Salary)
 VALUES
 (2, 'Mary', 26000);
```

```
INSERT INTO
 Employee  (Name,
 ID)
 VALUES ('Mary', 2);
```

```
INSERT INTO
 Employee  VALUES
 (2, 'Mary', 26000);
```

INSERT

Employee

ID	Name	Salary
1	John	25000

```
INSERT INTO
 Employee  (ID,
 Name, Salary)
 VALUES
 (2, 'Mary', 26000);
```

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000

```
INSERT INTO
 Employee  (Name,
 ID)
 VALUES ('Mary', 2);
```

Employee

ID	Name	Salary
1	John	25000
2	Mary	

```
INSERT INTO
 Employee  VALUES
 (2, 'Mary', 26000);
```

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000

Last week

```
CREATE TABLE Student (
  sID INT  PRIMARY  KEY,
  sName VARCHAR(50) NOT NULL,
  sAddress VARCHAR(255),
  sYear INT DEFAULT 1
);
```

INSERT

```
INSERT INTO Student
 (sID, sName, sAddress, sYear)
 VALUES
 (1, ‘Smith’, ‘5 Arnold Close’, 1);
```

```
INSERT INTO Student
 (sName, sAddress, sYear)
 VALUES
 (‘Smith’, NULL, 2);
```

```
INSERT INTO Student
 (sName, sAddress)
 VALUES
 (‘Smith’, ‘5 Arnold Close’),
 (‘Brooks’, ‘7 Holly Ave.’);
```

INSERT

```
INSERT INTO Student
 (sID, sName, sAddress, sYear)
 VALUES
 (1, 'Smith', '5 Arnold Close', 1);
```

Student

sID	sName	sAddress	sYear
1	Smith	5 Arnold Close	1

```
INSERT INTO Student
 (sName, sAddress, sYear)
 VALUES
 ('Smith', NULL, 2);
```

Student

sID	sName	sAddress	sYear
1	Smith	NULL	2

```
INSERT INTO Student
 (sName, sAddress)
 VALUES
 ('Smith', '5 Arnold Close'),
 ('Brooks', '7 Holly Ave.');
```

Student

sID	sName	sAddress	sYear
1	Smith	5 Arnold Close	1
2	Brooks	7 Holly Ave.	1

INSERT

However:

```
INSERT INTO Student
 VALUES
 ('Smith', '5 Arnold Close', 1);
```

→ ERROR!

```
INSERT INTO Student
 VALUES
 ('Smith', '5 Arnold Close');
```

→ ERROR!

UPDATE

- Changes values in specified rows based on WHERE conditions

```
UPDATE table-name
SET col1 = val1
   [,col2 = val2…]
 [WHERE
   condition]
```

- All rows where the condition is true have the columns set to the given values
- If no condition is given all rows are changed so BE CAREFUL
- Values are constants or can be computed from columns

UPDATE

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Anne	22000

```
UPDATE Employee
SET Salary = 15000,
    Name = 'Jane'
WHERE ID = 4
```

```
UPDATE
Employee  SET
Salary =       1.05
      Salary *
```

UPDATE

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Anne	22000

```
UPDATE Employee
SET Salary = 15000,
    Name = ‘Jane’
WHERE ID = 4
```

```
UPDATE
Employee  SET
Salary =       1.05
      Salary *
```

UPDATE

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Anne	22000

```
UPDATE Employee
SET Salary = 15000,
    Name = 'Jane'
WHERE ID = 4;
```

```
UPDATE
Employee  SET
Salary =       1.05;
       Salary *
```

UPDATE

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Anne	22000

```
UPDATE Employee
SET Salary = 15000,
    Name = 'Jane'
WHERE ID = 4;
```

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Jane	15000

```
UPDATE
Employee  SET
Salary =          1.05;
       Salary *
```

Employee

ID	Name	Salary
1	John	26250
2	Mary	27300
3	Mark	18900
4	Anne	23100

DELETE

- Removes all rows, or those which satisfy a condition

```
DELETE FROM
  table-name
  [WHERE
   condition]
```

- If no condition is given then ALL rows are deleted - BE CAREFUL
- You might also use **TRUNCATE TABLE** which is like **DELETE FROM** without a **WHERE** but is often quicker

DELETE

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Jane	15000

```
DELETE FROM
  Employee
 WHERE
  Salary > 20000;
```

DELETE

Employee

ID	Name	Salary
1	John	25000
2	Mary	26000
3	Mark	18000
4	Jane	15000

```
DELETE FROM
  Employee
 WHERE
  Salary > 20000;
```

Employee

ID	Name	Salary
3	Mark	18000
4	Jane	15000

```
DELETE FROM Employee;
```

Employee

ID	Name	Salary

SQL SELECT

- SELECT is the type of query you will use most often.
  - Queries one or more tables and returns the result as a table
  - Lots of options, which will be covered over the next few lectures
  - Usually queries can be achieved in a number of ways

Simple SELECT

```
SELECT columns
  FROM table-name;
```

`columns` can be

- A single column
- A comma-separated list of columns
- `*` for ‘all columns’

Sample SELECTs

```
SELECT * FROM Student;
```

Student

sID	sName	sAddress	sYear
1	Smith	5 Arnold Close	2
2	Brooks	7 Holly Avenue	2
3	Anderson	15 Main Street	3
4	Evans	Flat 1a, High Street	2
5	Harrison	Newark Hall	1
6	Jones	Southwell Hall	1

Sample SELECTs

```
SELECT sName FROM Student;
```

Sample SELECTs

```
SELECT sName FROM Student;
```

sName
Smith
Brooks
Anderson
Evans
Harrison
Jones

Sample SELECTs

```
SELECT sName, sAddress
 FROM Student;
```

Sample SELECTs

```
SELECT sName, sAddress
 FROM Student;
```

sName	sAddress
Smith	5 Arnold Close
Brooks	7 Holly Avenue
Anderson	15 Main Street
Evans	Flat 1a, High Street
Harrison	Newark Hall
Jones	Southwell Hall

Sample SELECTs

$\pi_{\text{sName, sAddress}}(\text{Student})$

sName	sAddress
Smith	5 Arnold Close
Brooks	7 Holly Avenue
Anderson	15 Main Street
Evans	Flat 1a, High Street
Harrison	Newark Hall
Jones	Southwell Hall

Being Careful

- When using DELETE and UPDATE
  - You need to be careful to have the right WHERE clause
  - You can check it by running a SELECT statement with the same WHERE clause first

Before running

```
DELETE FROM Student
    WHERE sYear = 3;
```

run

```
SELECT * FROM
 Student
WHERE sYear = 3;
```

Listing Tables

- To list all of your tables using SHOW:

```
SHOW tables;
```

Next Lecture

- SQL SELECT
    - WHERE Clauses
    - SELECT from multiple tables
    - JOINs
- Further reading
    - Database Systems, Connolly and Begg, Chapter 6
    - The Manga Guide to Databases, Chapter 4

SQL SELECT

Database Systems

This Lecture

- SQL SELECT
  - WHERE Clauses
  - SELECT from multiple tables
  - JOINs
- Further reading
  - Database Systems, Connolly & Begg, Chapter 6
  - The Manga Guide to Databases, Chapter 4

SQL SELECT Overview

```
SELECT
  [DISTINCT | ALL] column-list
  FROM table-names
  [WHERE condition]
  [ORDER BY column-list]
  [GROUP BY column-list]
  [HAVING condition]
```

([] *optional*, | *or*)

Example Tables

Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	Jones
S107	John	Brown

Course

Code	Title
DBS	Database Systems
PR1	Programming 1
PR2	Programming 2
IAI	Introduction to AI

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

DISTINCT and ALL

- Sometimes you end up with duplicate entries
- Using DISTINCT removes duplicates
- Using ALL retains duplicates
- ALL is used as a default if neither is supplied
- These will work over multiple columns

```
SELECT ALL Last
   FROM Student;
```

Last
Smith
Jones
Brown
Jones
Brown

```
SELECT DISTINCT Last
   FROM Student;
```

Last
Smith
Jones
Brown

WHERE Clauses

- In most cases returning all the rows is not necessary
  - A WHERE clause restricts rows that are returned
  - It takes the form of a condition – only rows that satisfy the condition are returned

- Example conditions:
  - `Mark < 40`
  - `First = ‘John’`
  - `First <> ‘John’`
  - `First = Last`
  - `(First = ‘John’) AND (Last = ‘Smith’)`
  - `(Mark < 40) OR (Mark > 70)`

WHERE Examples

```
SELECT * FROM Grade
  WHERE Mark >= 60;
```

```
SELECT DISTINCT ID
  FROM Grade
  WHERE Mark >= 60;
```

WHERE Examples

```
SELECT * FROM Grade
WHERE Mark >= 60;
```

ID	Code	Mark
S103	DBS	72
S104	PR1	68
S104	IAI	65
S107	PR1	76
S107	PR2	60

```
SELECT DISTINCT ID
FROM Grade
WHERE Mark >= 60;
```

ID
S103
S104
S107

WHERE Examples

- Given the table:

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

- Write an SQL query to find a list of the ID numbers and Marks for students who have passed (scored 50% or more) in IAI

ID	Mark
S103	58
S104	65

Solution

```
SELECT ID, Mark FROM Grade
   WHERE (Code = ‘IAI’)
   AND (Mark >= 50);
```

WHERE Examples

- Given the table:

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

- Write an SQL query to find a list of the ID numbers and Marks for students who have passed with Merit (Marks in [60, 69])

ID	Mark
S104	68
S104	65
S107	60

Solution

```
SELECT ID, Mark FROM Grade
    WHERE (Mark >=60
    AND Mark < 70);
```

Solution (only in MySQL!)

```
SELECT ID, Mark FROM Grade WHERE
    Mark BETWEEN 60 AND 69;
```

WHERE Examples

- Given the table:

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

- Write an SQL query to find a list of the students IDs for both the IAI and PR2 modules

ID
S103
S104
S106
S107
S107

WHERE Examples

- Given the table:

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

- Write an SQL query to find a list of the students IDs for both the IAI and PR2 modules

ID
S103
S104
S106
S107
S107

Solution

```
SELECT ID FROM Grade
    WHERE (Code = 'IAI'
    OR Code = 'PR2');
```

SELECT from Multiple Tables

- Often you need to combine information from two or more tables
- You can produce the effect of a Cartesian product using:

```
SELECT * FROM Table1,
  Table2
```

- If the tables have columns with the same name, ambiguity will result
- This can be resolved by referencing columns with the table name:

```
TableName.ColumnName
```

SELECT from Multiple Tables

```
SELECT
            Last, Mark
  First,
FROM
  Student, Grade
WHERE
  (Student.ID = Grade.ID)
  AND (Mark >= 40);
```

Student

ID	First	Last
S103	John	Smith
S104	Mar	
S105	Jane	
S106	Mar	
S107	John	

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

SELECT from Multiple Tables

```
SELECT ... FROM Student, Grade WHERE ...
```

ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S103	John	Smith	S104	PR1	68
S103	John	Smith	S104	IAI	65
S103	John	Smith	S106	PR2	43
S103	John	Smith	S107	PR1	76
S103	John	Smith	S107	PR2	60
S103	John	Smith	S107	IAI	35
S104	John	Smith	S103	DBS	72
S104	Mary	Jones	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65

SELECT from Multiple Tables

```
SELECT ... FROM Student, Grade
  WHERE (Student.ID = Grade.ID) AND ...
```

ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65
S106	Mark	Jones	S106	PR2	43
S107	John	Brown	S107	PR1	76
S107	John	Brown	S107	PR2	60
S107	John	Brown	S107	IAI	35

SELECT from Multiple Tables

```
SELECT ... FROM Student, Grade
  WHERE (Student.ID = Grade.ID) AND (Mark >= 40)
```

ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65
S106	Mark	Jones	S106	PR2	43
S107	John	Brown	S107	PR1	76
S107	John	Brown	S107	PR2	60

SELECT from Multiple Tables

```
SELECT First, Last, Mark FROM Student, Grade
  WHERE (Student.ID = Grade.ID) AND (Mark >= 40)
```

First	Last	Mark
John	Smith	72
John	Smith	58
Mary	Jones	68
Mary	Jones	65
Mark	Jones	43
John	Brown	76
John	Brown	60

SELECT from Multiple Tables

- When selecting from multiple tables, it is almost always best to use a **WHERE** clause to find common values

```
SELECT *
From
   Student, Grade,
   Course
WHERE
   Student.ID =
   Grade.ID
   AND
   Course.Code =
   Grade.Code
```

SELECT from Multiple Tables

Student			Grade			Course	
ID	**First**	**Last**	**ID**	**Code**	**Mark**	**Code**	**Title**
S103	John	Smith	S103	DBS	72	DBS	Database Systems
S103	John	Smith	S103	IAI	58	IAI	Introduction to AI
S104	Mary	Jones	S104	PR1	68	PR1	Programming 1
S104	Mary	Jones	S104	IAI	65	IAI	Introduction to AI
S106	Mark	Jones	S106	PR2	43	PR2	Programming 2
S107	John	Brown	S107	PR1	76	PR1	Programming 1
S107	John	Brown	S107	PR2	60	PR2	Programming 2

Student.ID = Grade.ID

Grade.Code = Course.Code

Examples

Student

sID	sName	sAddress	sYear
1	Smith	5 Arnold Close	2
2	Brooks	7 Holly Avenue	2
3	Anderson	15 Main Street	3
4	Evans	Flat 1a, High Street	2
5	Harrison	Newark Hall	1
6	Jones	Southwell Hall	1

Module

mCode	mCredits	mTitle
G51DBS	10	Database Systems
G51PRG	20	Programming
G51IAI	10	Artificial Intelligence
G52ADS	10	Algorithms

Enrolment

sID	mCode
1	G52ADS
2	G52ADS
5	G51DBS
5	G51PRG
5	G51IAI
4	G52ADS
6	G51PRG
6	G51IAI

Examples

- Write SQL statements to do the following:
  1. Produce a list of all student names and all their enrolments (module codes)
  2. Find a list of students who are enrolled on the G52ADS module
  3. Find a list of module titles being taken by the student named “Harrison”
  4. Find a list of module codes and titles for all modules currently being taken by first year students

Solutions

1.
```
SELECT sName, mCode FROM Student, Enrolment
WHERE Student.sID = Enrolment.sID;
```

2.
```
SELECT sID, sName FROM Student, Enrolment
WHERE Student.sID = Enrolment.sID and mCode= 'G52ADS';
```

3.
```
SELECT mTitle FROM Module, Student, Enrolment
WHERE (Module.mCode = Enrolment.mCode) AND
(Student.sID = Enrolment.sID) AND
Student.sName = "Harrison";
```

4.
```
SELECT Module.mCode, mTitle FROM Enrolment,
Module, Student WHERE
(Module.mCode = Enrolment.mCode) AND
(Student.sID = Enrolment.sID) AND sYear = 1;
```

Next Lecture

- More SQL SELECT
  - Aliases
  - 'Self-Joins'
  - Subqueries
  - IN, EXISTS, ANY, ALL
  - LIKE
- Further reading
  - Database Systems, Connolly & Begg, Chapter 6
  - The Manga Guide to Databases, Chapter 4