SQL DATA DEFINITION

Introduction to Database Systems

Mohammad Tanhaei
Ilam University

IN THIS LECTURE

- SQL
  - The SQL language
  - SQL, the relational model, and E/R diagrams
  - CREATE TABLE
    - Columns
    - Primary Keys
    - Foreign Keys
- For more information
  - Connolly and Begg chapter 6
  - Ullman and Widom 3.2, 6.6.

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
  - SQL-89
  - SQL-92 (SQL2)
  - SQL-99 (SQL3)
- Most modern DBMS use a variety of SQL
  - Most based on SQL2, increasingly SQL3
  - Few (if any) are true to the standard

SQL

- SQL provides
  - A data definition language (**DDL**)
  - A data manipulation language (**DML**)
  - A data control language (**DCL**)
- In addition SQL
  - Can be used from other languages
  - Is often extended to provide common programming constructs (such as if-then tests, loops, variables, etc.)

NOTES

- SQL is (usually) not case-sensitive, but we’ll write SQL keywords in upper case for emphasis
- SQL statements will be written in **`COURIER FONT`**
- Strings in SQL are surrounded by single quotes:

  `'I AM A STRING’`
- Single quotes within a string are doubled:

  `'I''M A STRING’`
- The empty string: `''`

NON-PROCEDURAL PROGRAMMING

- SQL is a **declarative** (non-procedural) language
  - **Procedural** - say exactly what the computer has to do
  - **Non-procedural (imperative)** – describe the required result (not the way to compute it)

- Example: Given a database with tables
  - **Student** with attributes ID, Name, Address
  - **Module** with attributes Code, Title
  - **Enrolment** with attributes ID, Code
- Get a list of students who take the module ‘Database Systems’

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        /* add them to the list    */
      Set E to be the next Enrolment Record
   Set S to be the next Student Record
Return NAMES
```

NON-PROCEDURAL (SQL)

```
SELECT Name FROM Student, Enrolment
  WHERE
  (Student.ID = Enrolment.ID)
  AND
  (Enrolment.Code =
    (SELECT Code FROM Module WHERE
       Title = ‘Database Systems’))
```

SQL, THE RELATIONAL MODEL, AND E/R DESIGN

- SQL is based on the relational model
  - It has many of the same ideas
  - Databases that support SQL are often described as relational databases
  - It is not always true to the model

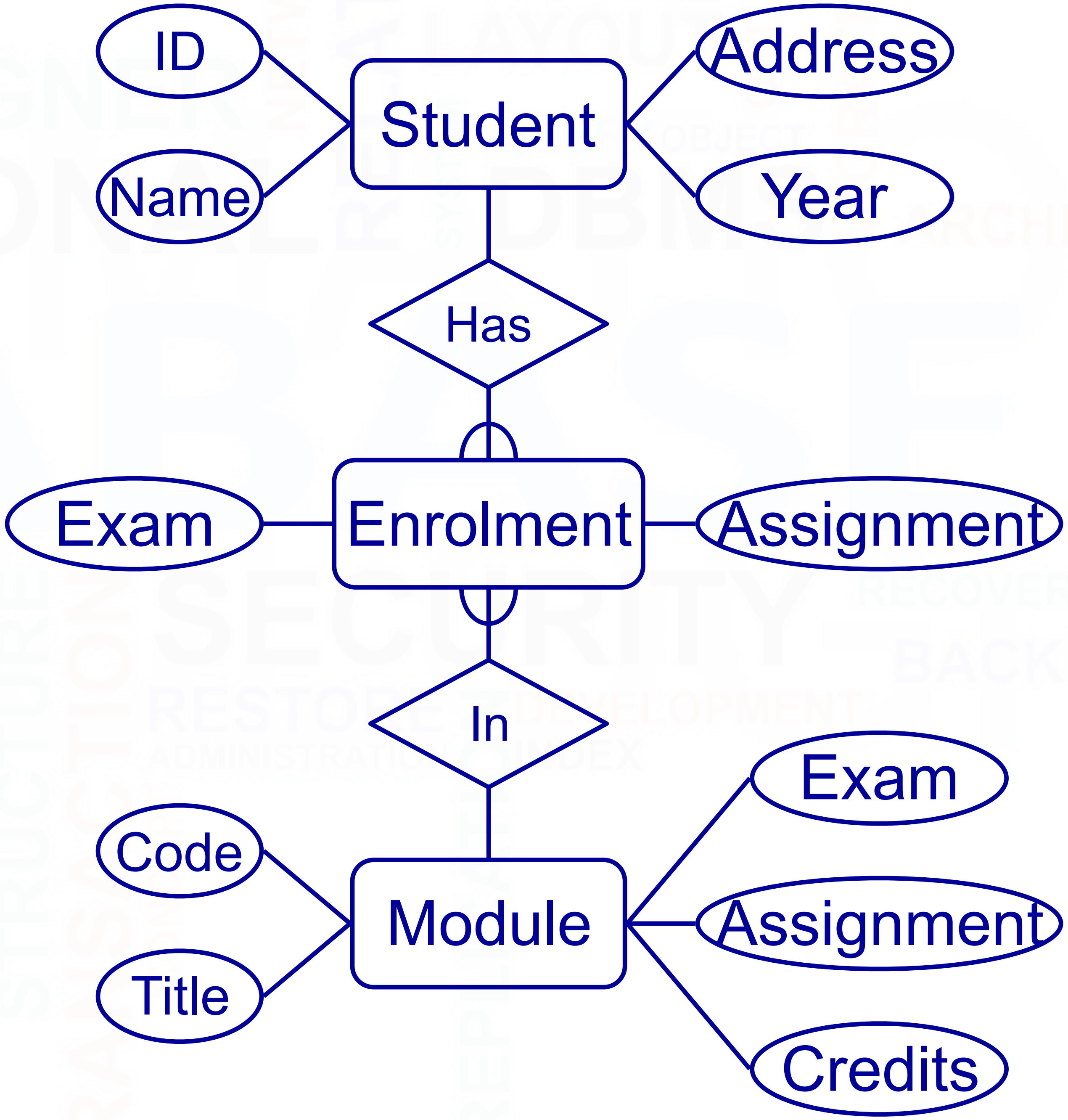
- E/R designs can be implemented in SQL
  - Entities, attributes, and relationships can all be expressed in terms of SQL
  - Many-to-many relationships are a problem, so should be removed

RELATIONS, ENTITIES, TABLES

Relational model	E/R Diagram	SQL
Relation	Entity	Table
Tuple	Instance	Row
Attribute	Attribute	Column or Field
Foreign Key	M:1 Relationship	Foreign Key
Primary Key		Primary Key

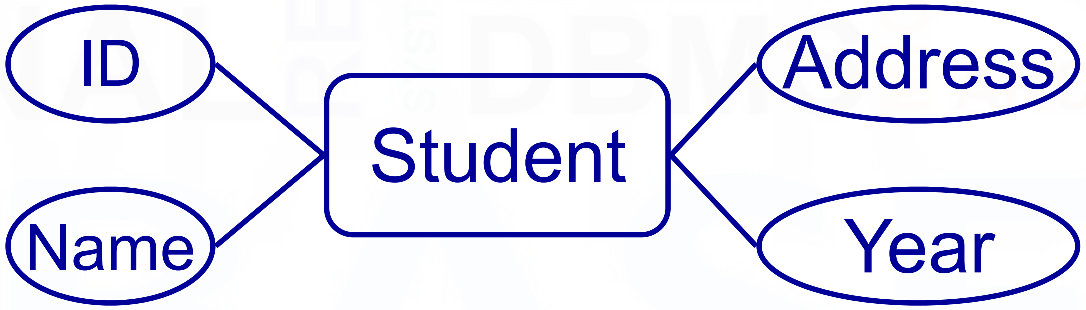
IMPLEMENTING E/R DESIGNS

- Given an E/R design
  - The entities become SQL tables
  - Attributes of an entity become columns in the corresponding table
  - Relationships may be represented by foreign keys



ENTITIES AND ATTRIBUTES

- Each entity becomes a table in the database
  - The name of the table is often the name of the entity
  - The attributes become columns of the table with the same name



- A table called Student
- With columns for ID, Name, Address, and Year

CREATE TABLE

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

- You supply
  - A name for the table
  - A list of column definitions
  - A list of constraints (such as keys)

COLUMN DEFINITIONS

```
<col-name> <type>
  [NULL|NOT NULL]
  [DEFAULT <val>]
  [constraint-1 [,
   constraint-2[,
     ...]]]
```

- Each column has a name and a type
- Common types
  - **INT**
  - **REAL**
  - **CHAR(n)**
  - **VARCHAR(n)**
  - **DATE**
  - …

COLUMN DEFINITIONS

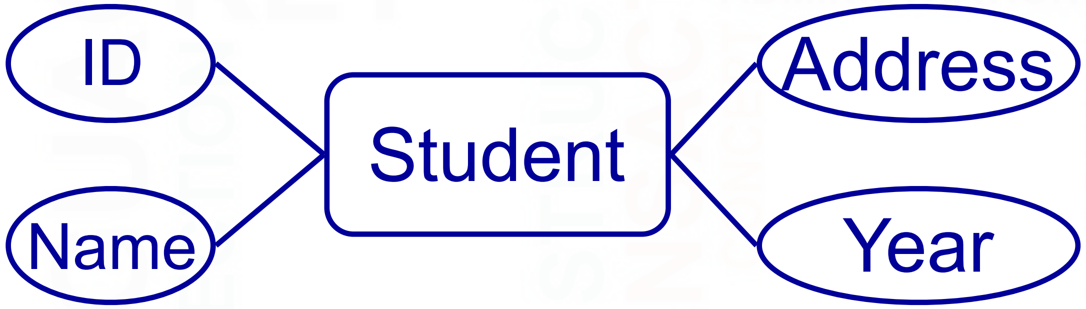
- Columns can be specified as **NULL** or **NOT NULL**
- **NOT NULL** Columns cannot have missing values
- If neither is given then columns are assumed **NULL**

- Columns can be given a default value
- You just use the keyword DEFAULT followed by the value, eg:

```
num INT DEFAULT 0
```

EXAMPLE

```
CREATE TABLE Student (
   stuID INT NOT NULL,
   Name VARCHAR(50) NOT NULL,
   Address VARCHAR(50),
   Year INT DEFAULT 1)
```



ID

Name

Student

Address

Year

CONSTRAINTS

```
CONSTRAINT
   <name>
   <type>
   <details>
```

- Common <type>s
  - PRIMARY KEY
  - UNIQUE
  - FOREIGN KEY
  - INDEX

➤ Each constraint is given a name - Access requires a name, but some others don’t

➤ Constraints which refer to single columns can be included in their definition

PRIMARY KEYS

- Primary Keys are defined through constraints
- A **PRIMARY KEY** constraint also includes a **UNIQUE** constraint and makes the columns involved **NOT NULL**
- The **<details>** for a primary key is a list of columns which make up the key

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

UNIQUE CONSTRAINTS

- 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

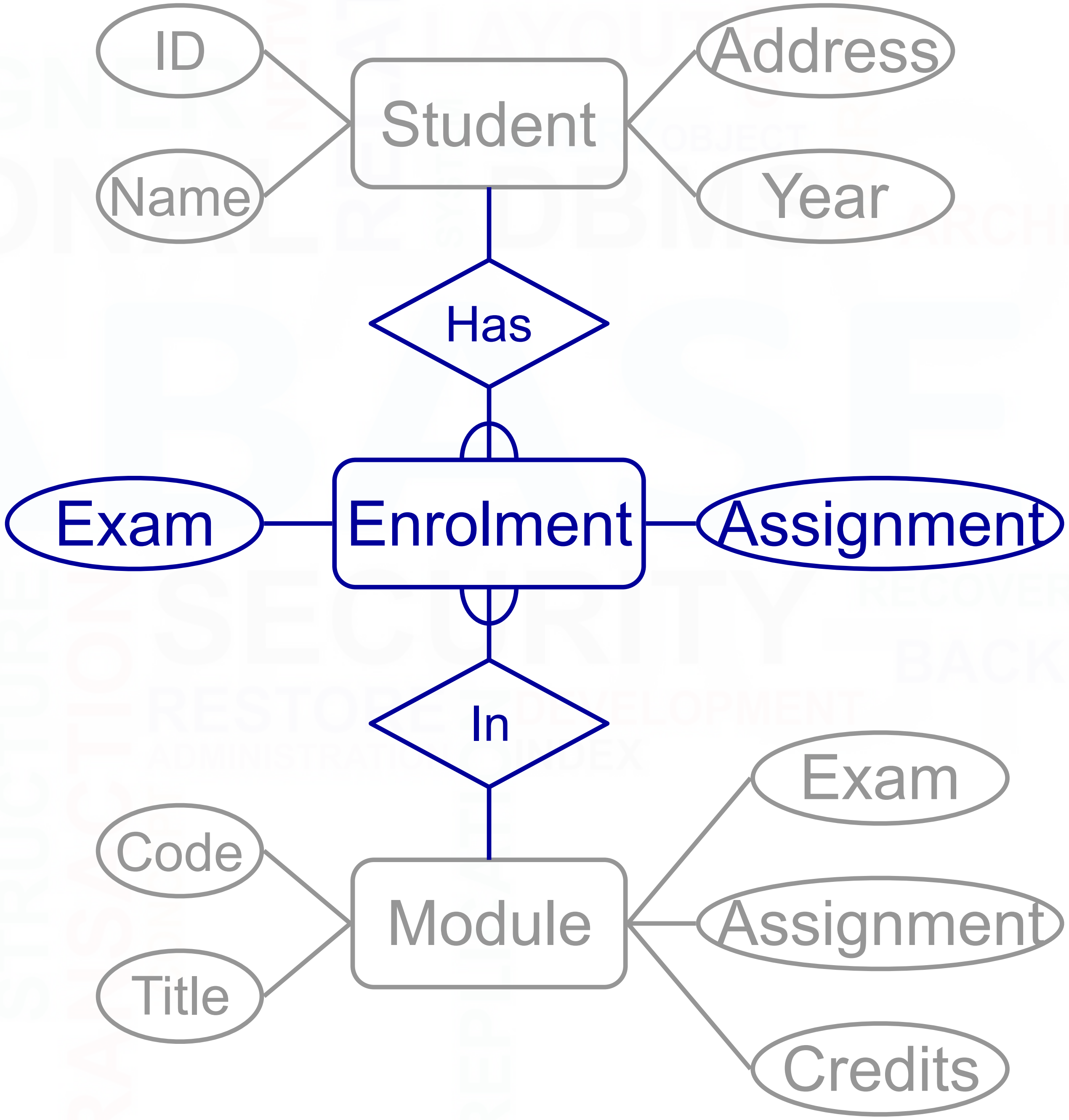
```
CONSTRAINT <name>
   UNIQUE
    (col1, col2, …)
```

EXAMPLE

```
CREATE TABLE Student (
  stuID INT NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Address VARCHAR(50),
  Year INT DEFAULT 1,
  CONSTRAINT pkStudent
    PRIMARY KEY (stuID))
```

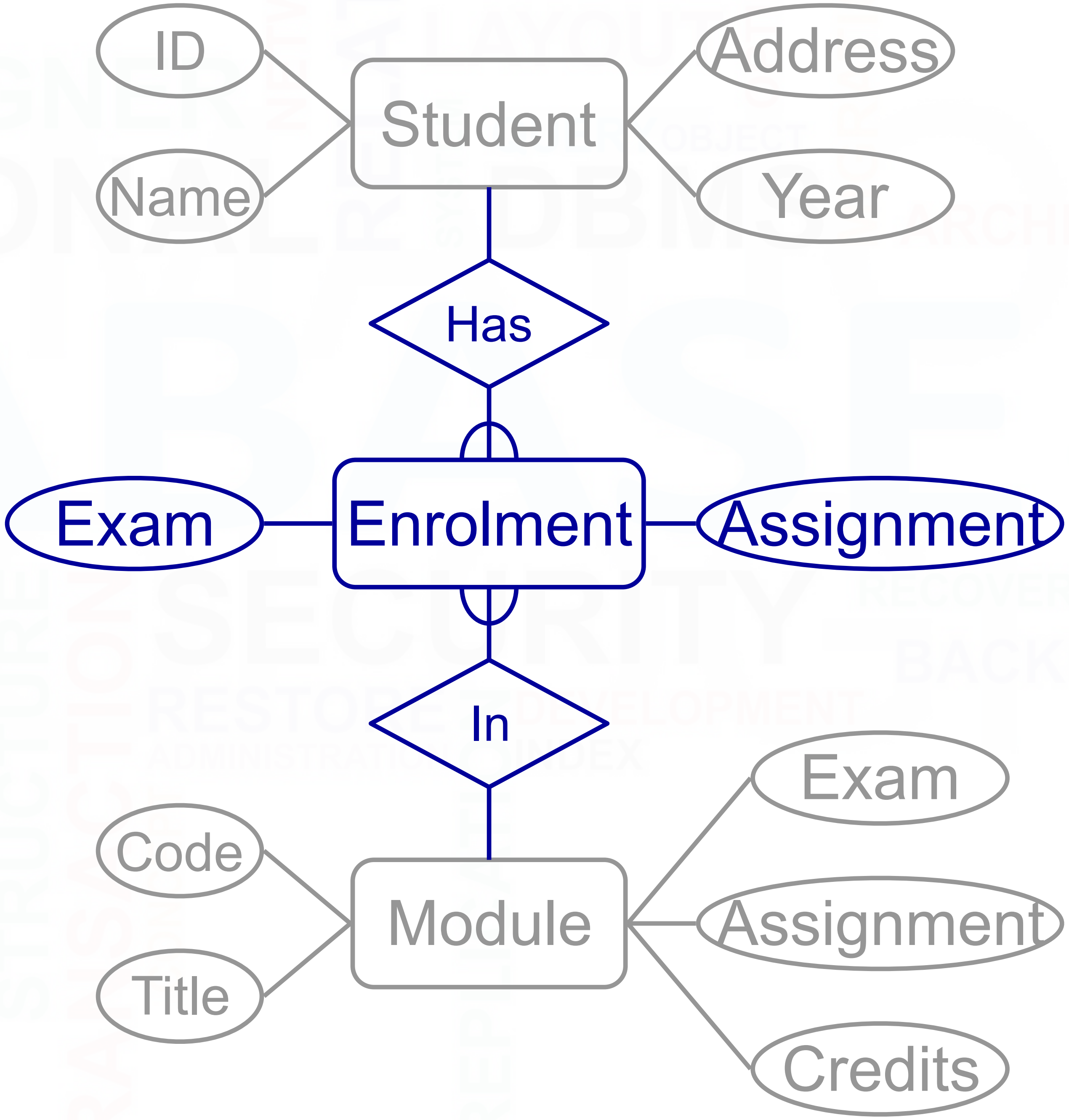
RELATIONSHIPS

- Depends on the type
  - 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



REPRESENTING RELATIONSHIPS

- The Enrolment table
  - Will have columns for the Exam and Assignment 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 give
  - The columns which make up the FK
  - The referenced table
  - The columns which are referenced by the FK

```
CONSTRAINT <name>
   FOREIGN KEY
   (col1,col2,…)
   REFERENCES
   <table>
   [(ref1,ref2,…)]
```

- If the FK references the PK of `<table>` you don't need to list the columns

EXAMPLE

```
CREATE TABLE Enrolment (
  stuID INT NOT NULL,
  modCode CHAR(6) NOT NULL,
  Assignment INT,
  Exam INT,
  CONSTRAINT enrPK
    PRIMARY KEY (stuID, modCode),
  CONSTRAINT enrStu FOREIGN KEY (stuID)
    REFERENCES Student (stuID),
  CONSTRAINT enrMod FOREIGN KEY (modCode)
    REFERENCES Module (modCode))
```

CREATING TABLES

- CREATE TABLE
- Columns
  - Data types
  - [NOT] NULL, DEFAULT values
- Constraints
  - Primary keys
  - Unique columns
  - Foreign keys

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

DELETING TABLES

- To delete a table use

```
DROP TABLE
  [IF EXISTS]
  <name>
```

- Example:

```
DROP TABLE Module
```

- **BE CAREFUL** with any SQL statement with DROP in it
  - You will delete any information in the table as well
  - You won’t normally be asked to confirm
  - There is no easy way to undo the changes

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

ALTERING COLUMNS

- To add or remove columns use

```
ALTER TABLE <table>
 ADD COLUMN <col>

ALTER TABLE <table>
 DROP COLUMN <name>
```

- Examples

```
ALTER TABLE Student
   ADD COLUMN
   Degree VARCHAR(50)

ALTER TABLE Student
   DROP COLUMN Degree
```

ALTERING CONSTRAINTS

- To add or remove columns use

```
ALTER TABLE <table>
   ADD CONSTRAINT
         <definition>

ALTER TABLE <table>
   DROP CONSTRAINT
          <name>
```

- Examples

```
ALTER TABLE Module
   ADD CONSTRAINT
   ck UNIQUE (title)

ALTER TABLE Module
   DROP CONSTRAINT ck
```

END

But take a look at next slide!

NEXT LECTURE

- More SQL
  - INSERT, UPDATE, and DELETE
  - Data dictionary
  - Sequences
- For more information
  - Connolly and Begg chapters 5 and 6
  - Ullman and Widom 6.5