

SQL Data Definition

Database Systems Lecture 5

Munawar, PhD

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 **BOLD 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 – 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
```

SQL Data Definition

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

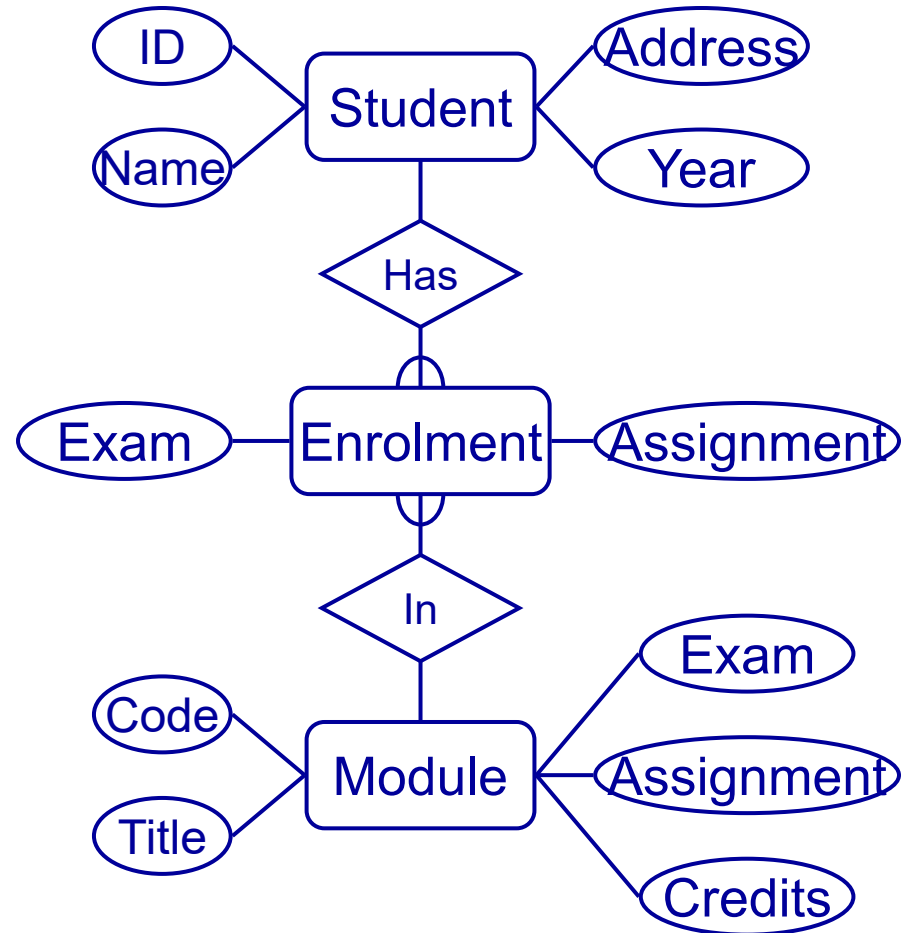
- SQL is based on the relational model
 - It has many of the same ideas
 - Databases that support SQL are often described as relational databases
 - It is not always true to the model
- E/R designs can be implemented in SQL
 - Entities, attributes, and relationships can all be expressed in terms of SQL
 - 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

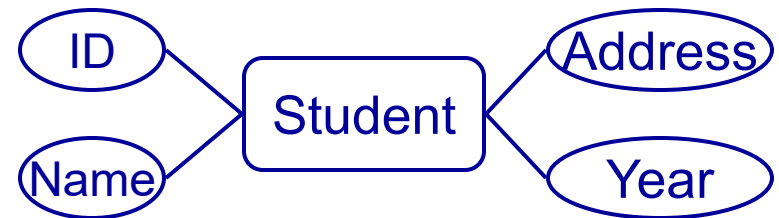
Implementing E/R Designs

- 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

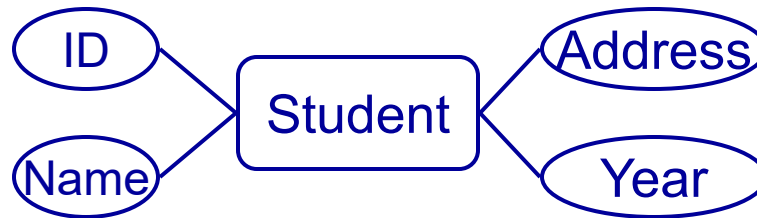
Column Definitions

- 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,  
    stuName VARCHAR(50) NOT NULL,  
    stuAddress VARCHAR(50) ,  
    stuYear INT DEFAULT 1)
```



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

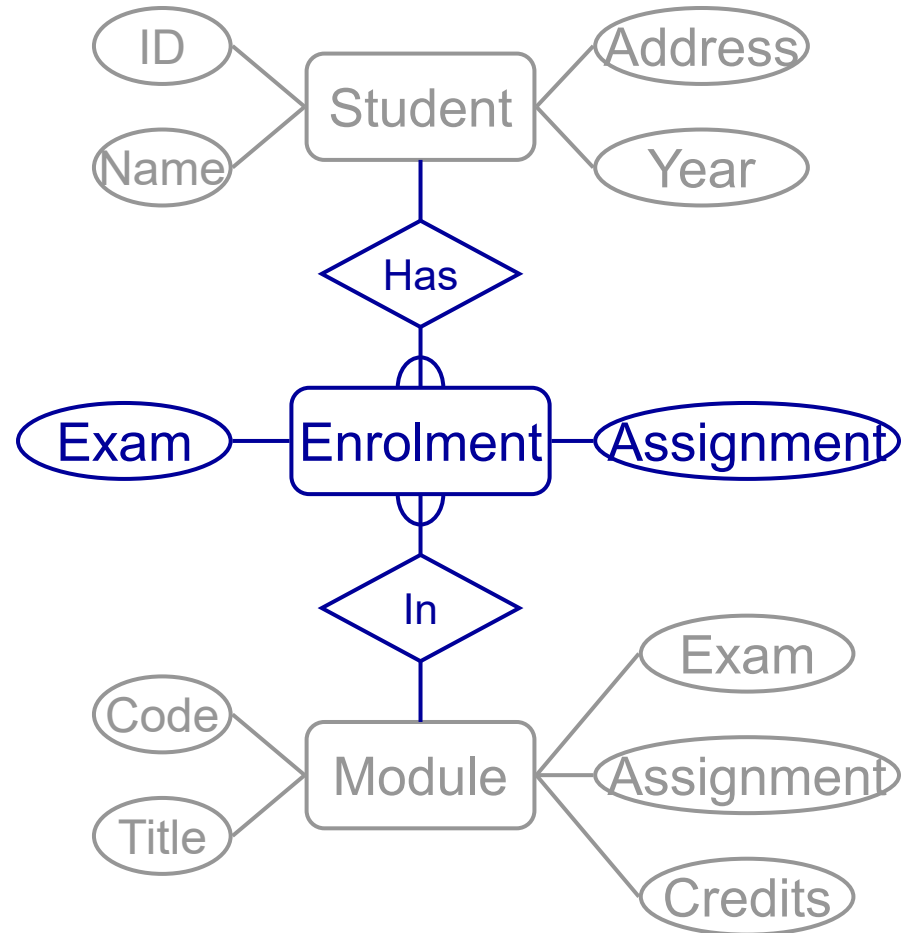
```
CONSTRAINT <name>  
    UNIQUE  
    (col1, col2, ...)
```

Example

```
CREATE TABLE Student (  
    stuID INT NOT NULL,  
    stuName VARCHAR(50) NOT NULL,  
    stuAddress VARCHAR(50) ,  
    stuYear INT DEFAULT 1,  
    CONSTRAINT pkStudent  
        PRIMARY KEY (stuID))
```

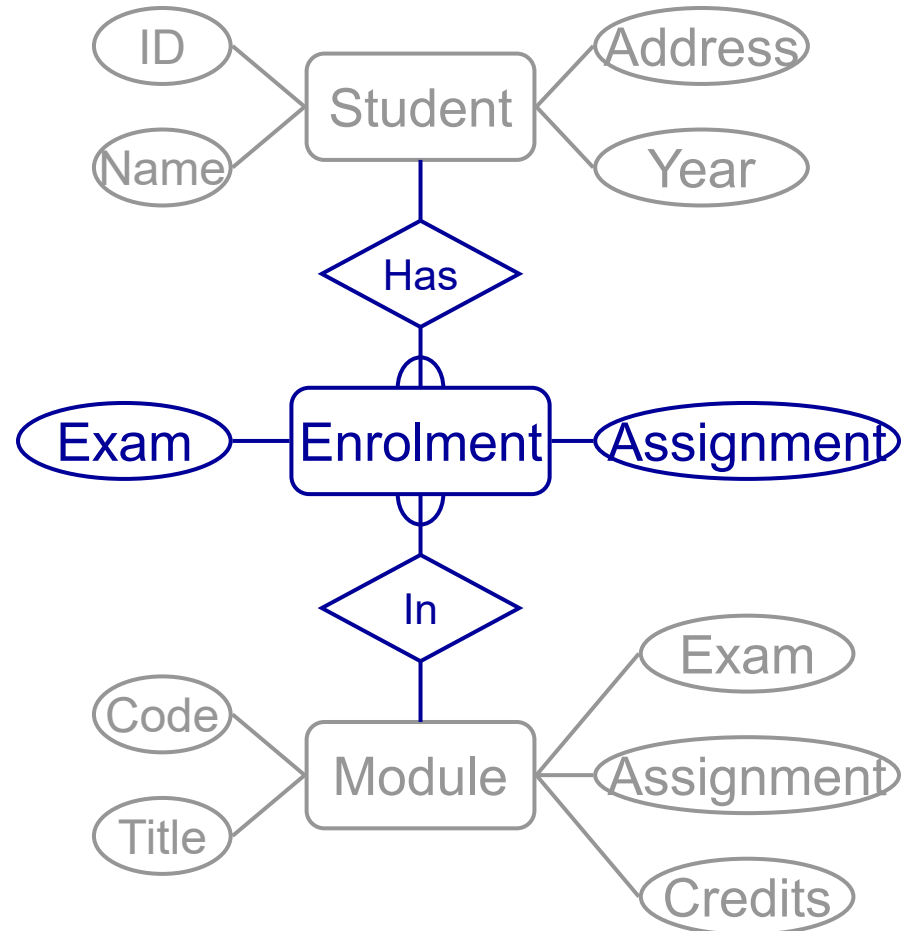
Relationships

- 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,  
    enrAssignment INT,  
    enrExam INT,  
    CONSTRAINT enrPK  
        PRIMARY KEY (stuID, modCode) ,  
    CONSTRAINT enrStu FOREIGN KEY (stuID)  
        REFERENCES Student (stuID) ,  
    CONSTRAINT enrMod FOREIGN KEY (modCode)  
        REFERENCES Module (modCode))
```


Next Lecture

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