SQL Data Definition

G51DBS Database Systems

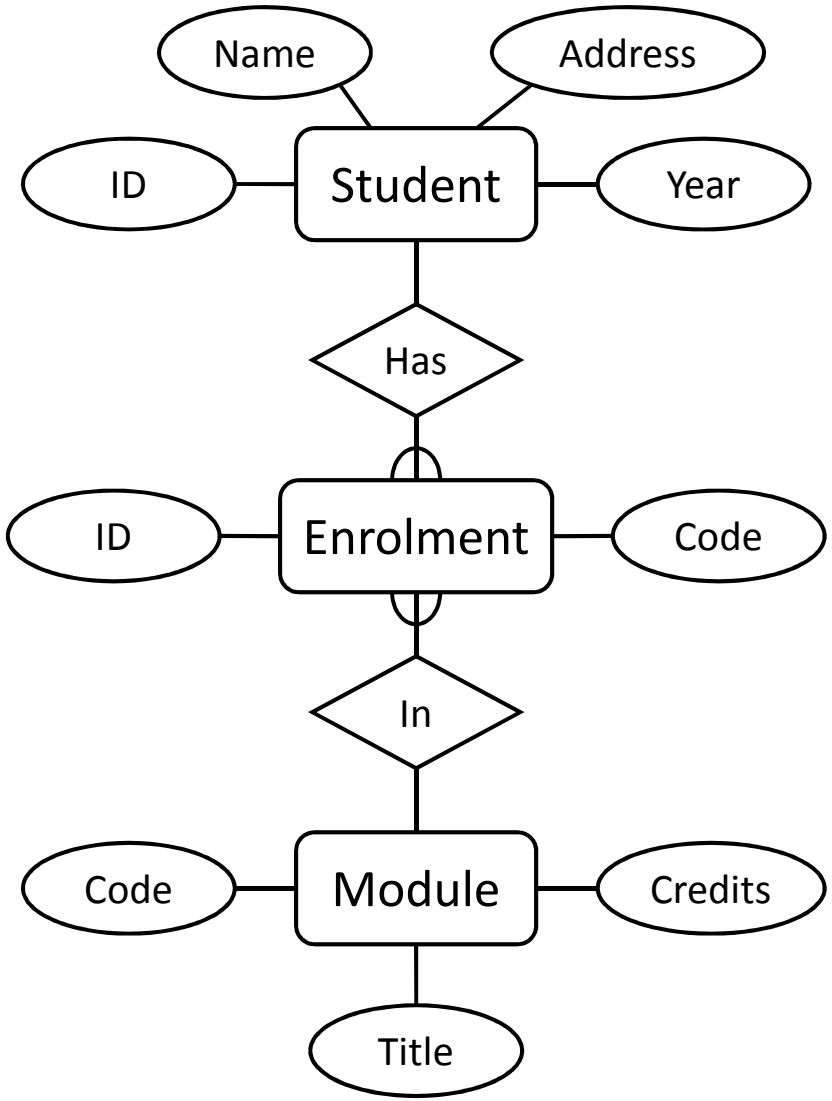
Jason Atkin

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

Last Lecture

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



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)

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)

MySQL

- During this module we will use MySQL as our DBMS
  - Free to use
  - Source code available under General Public License
  - Extremely popular and widely used
  - Easy to set up on the school servers
  - In most cases it is as functional as commercial DBMSs

SQL Case

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

```
SELECT * FROM Students
    WHERE Name = “James”;
```

Important: MySQL in Windows is not case sensitive. Do not be complacent during the coursework. Your coursework MUST work on the Linux machines (bann, clyde, etc).

SQL Strings

- Strings in SQL are surrounded by single quotes:
  - `'I AM A STRING'`
- Single quotes within a string are doubled or escaped using \
  - `'I''M A STRING'`
  - `'I\'M A STRING'`
- `''` is an empty string
- In MySQL, double quotes also work (this isn't the ANSI standard)

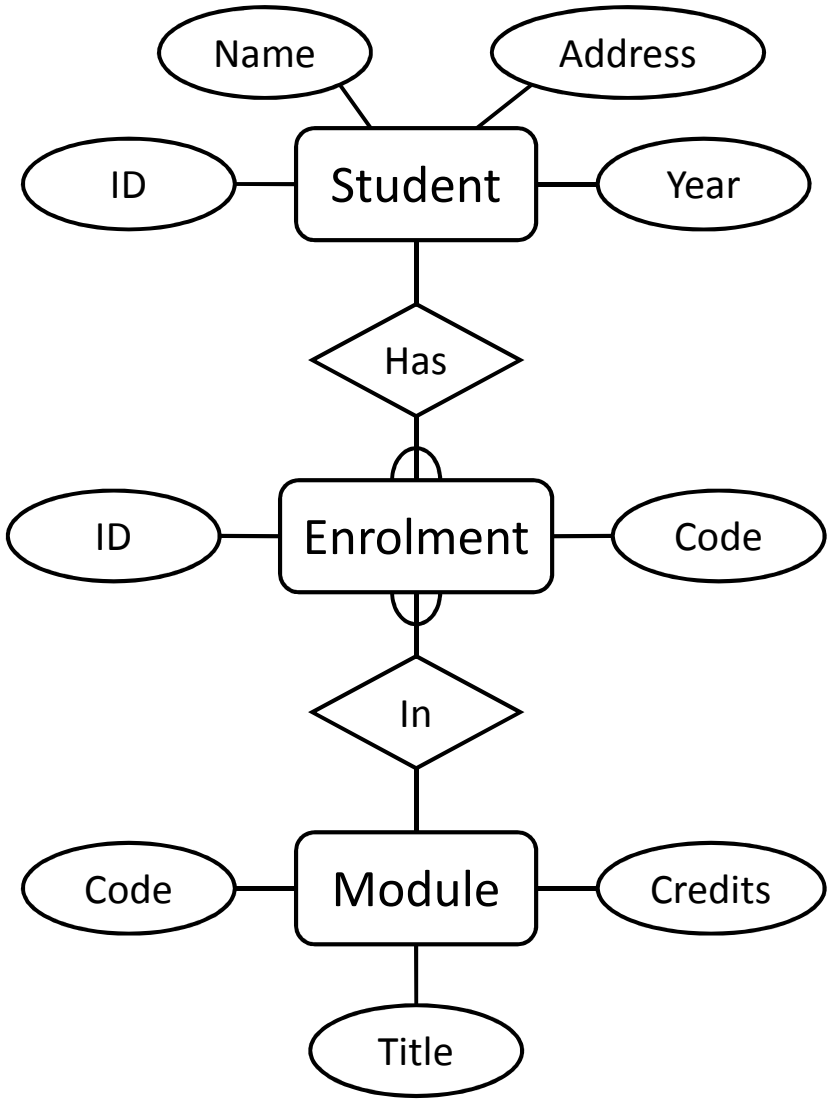
Non-Procedural Programming

- 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 ID, Name, Address
  - Module with attributes Code, Title
  - Enrolment with attributes ID, Code
- **Get a list of students who take the module 'Database Systems'**

Diagram of procedural solution

- Iterate through the modules looking for the module ‘Database systems’

- Iterate through the students
  - Iterate through the enrolments
    - If the enrolment matches the student and module then record the student name



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)

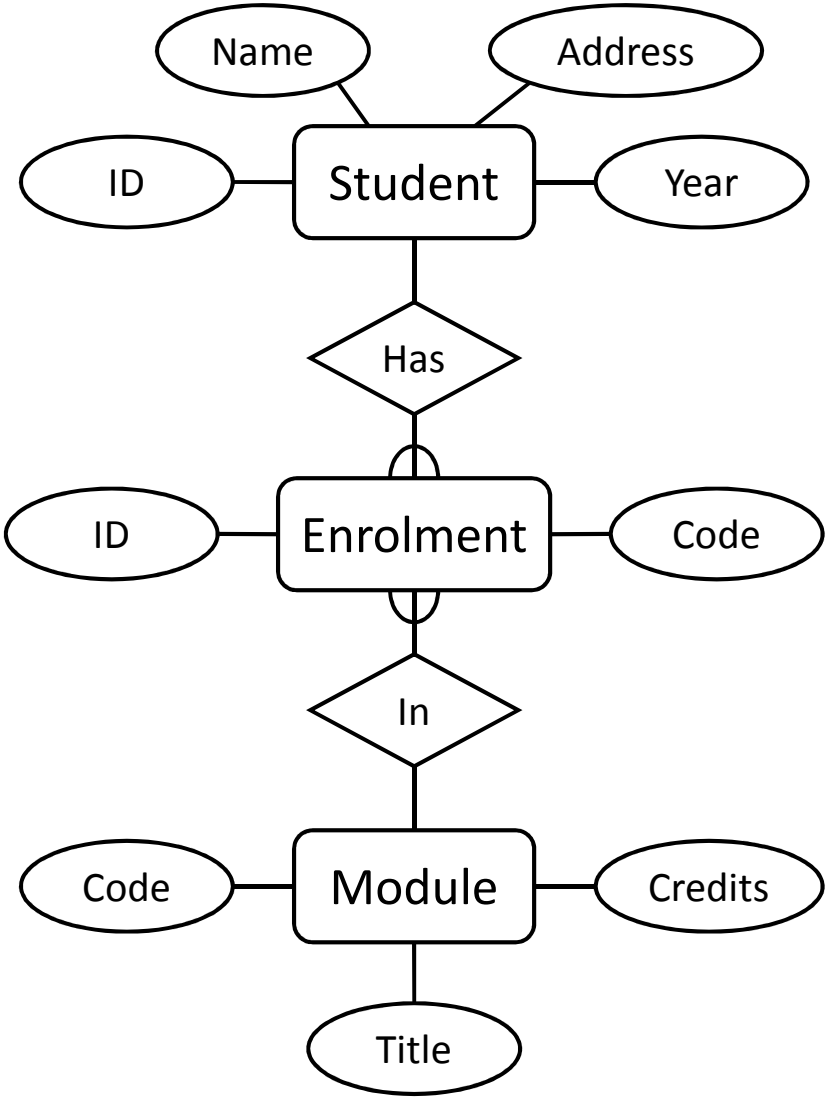
```
SELECT Name FROM Student, Enrolment
   WHERE
   (Student.ID = Enrolment.ID)
   AND
   (Enrolment.Code =
      (SELECT Code FROM Module WHERE
         Title = 'Database Systems'));
```

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
  - Often optimised for speed of search and append
  - ACID (atomicity, consistency, isolation, durability) guarantees often not met
  - Often trade off these (e.g. 'eventual consistency') guarantees of SQL for speed, especially with large databases

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



Relations, Entities and Tables

- The terminology changes between the Relational Model, E/R modelling and diagrams, and 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	Attribute	Primary Key

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>
);
```

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

Column Definitions

```
<col-name> <type>
  [NULL | NOT NULL]
  [DEFAULT default_value]
  [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 (Binary Large Object)

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

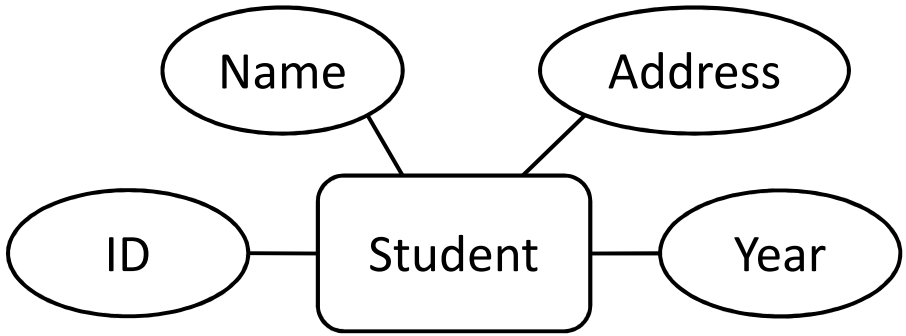
- Columns can be specified as **`NULL`** or **`NOT NULL`**
- **`NOT NULL`** columns **cannot** have missing values
- **`NULL`** is the **default** if you do not specify either

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

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

Example

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



Actually trying this…

- Login to the linux boxes (avon for you)
- If you have not created your database, do so and record your password (see instructions in lab introduction)
- Start mysql
- Use the command line – end commands with ;

AUTO_INCREMENT

- If you specify a column as **AUTO_INCREMENT**, a value (usually max(col) + 1) is automatically inserted when data is added. ***This is useful for Primary Keys***
- For example:

```
sID INT AUTO_INCREMENT,
```

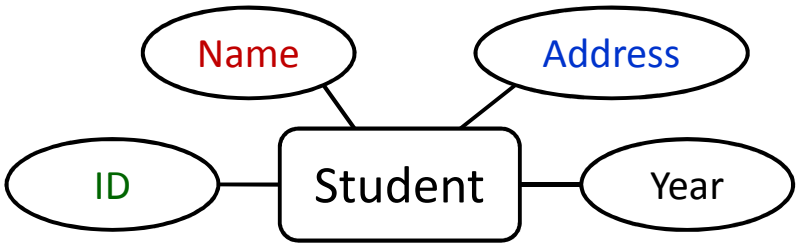
- When it comes to inserting values, you should ensure that you do not specify a value for the column

Note: The table auto_increment value is **not** recalculated during deletes. You might want to reset it using:

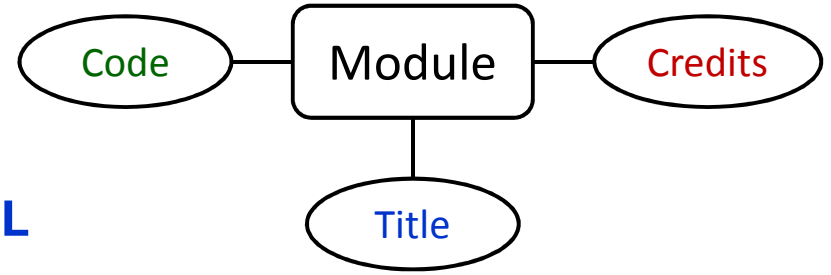
```
ALTER TABLE <name> AUTO_INCREMENT=1;
```

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



Constraints

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

- MySQL Constraints
  - **PRIMARY KEY**
  - **UNIQUE**
  - **FOREIGN KEY**
  - **INDEX**

- 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

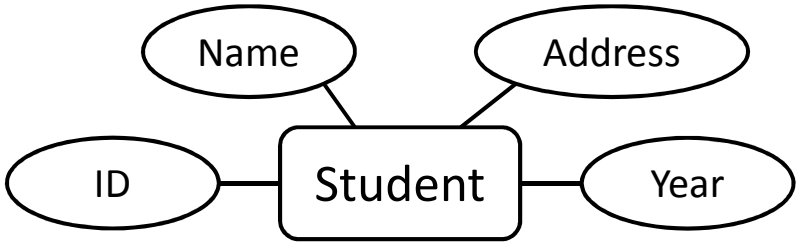
Primary Keys

- 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

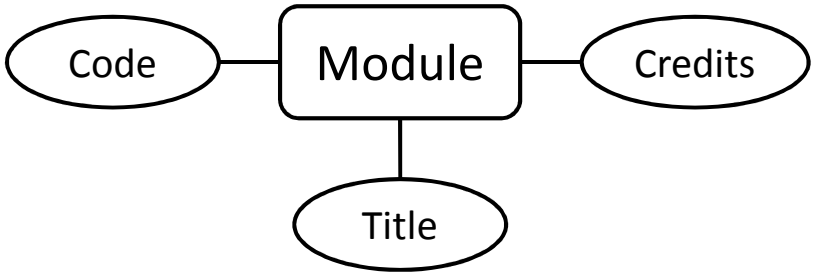
```
CONSTRAINT <name>
  PRIMARY KEY
  (col1, col2, …)
```

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)
);
```



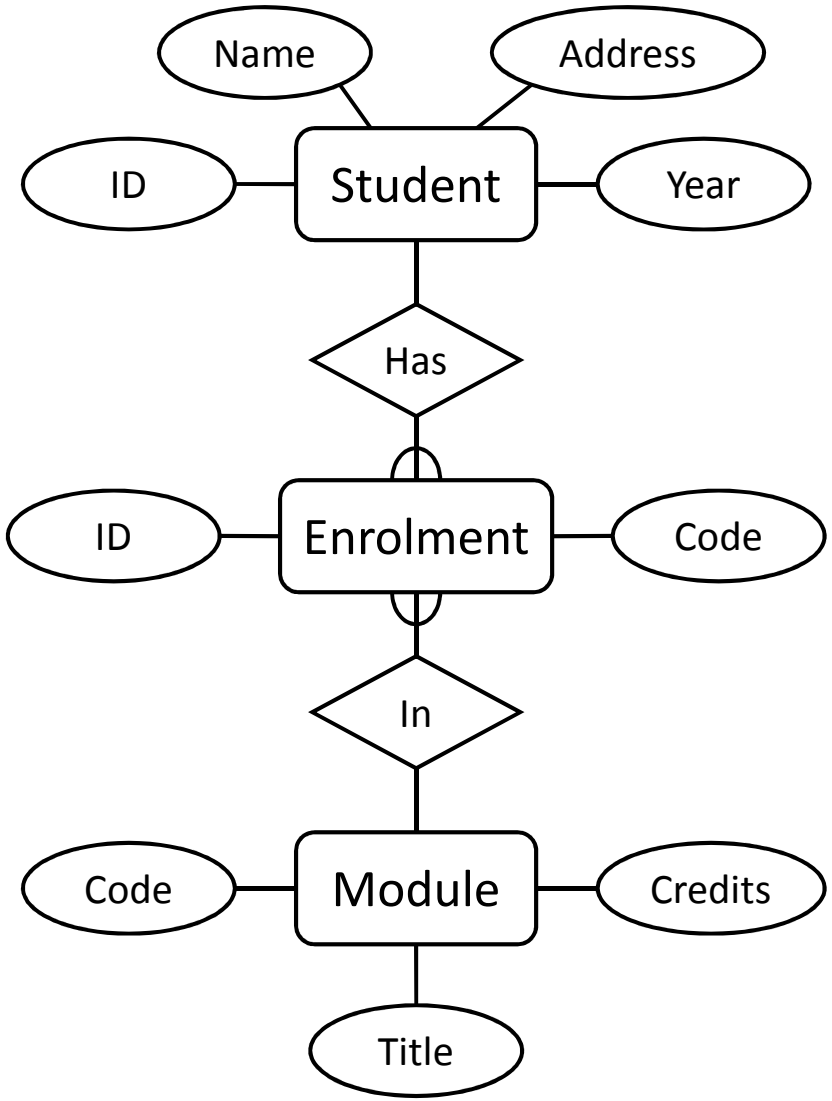
Unique Constraints

- 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, …)
```

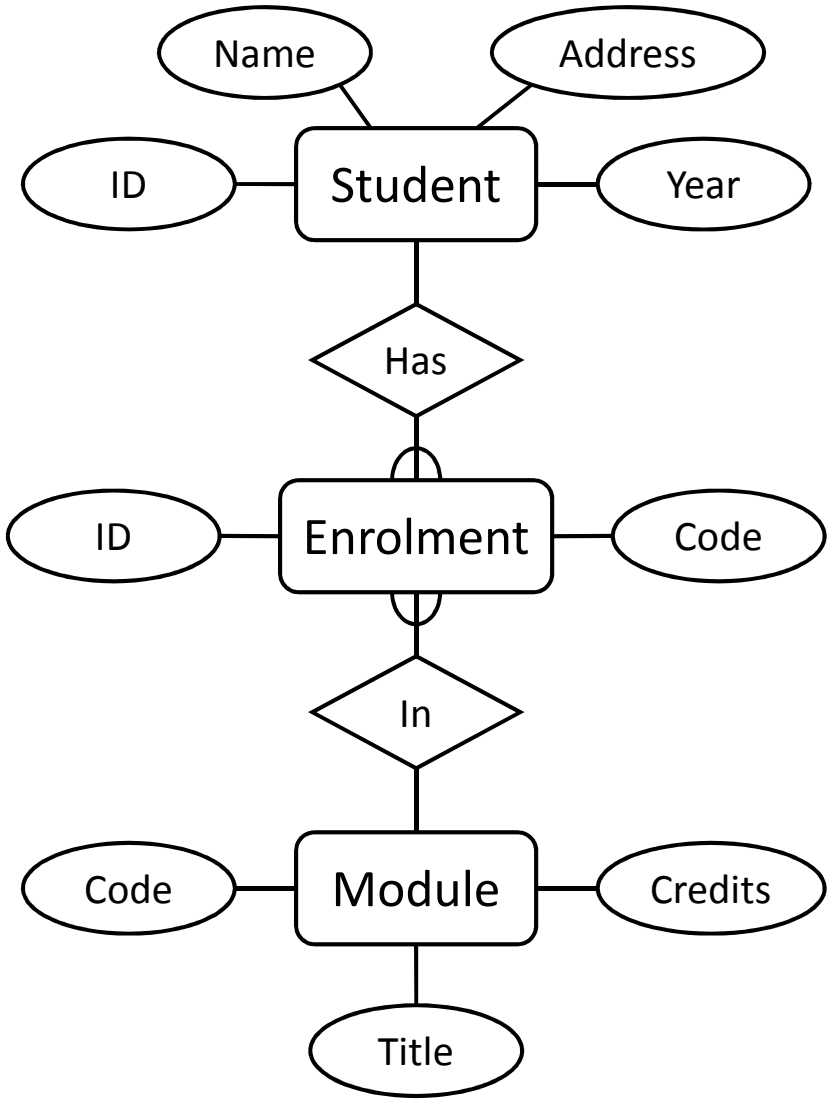
Relationships

- 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

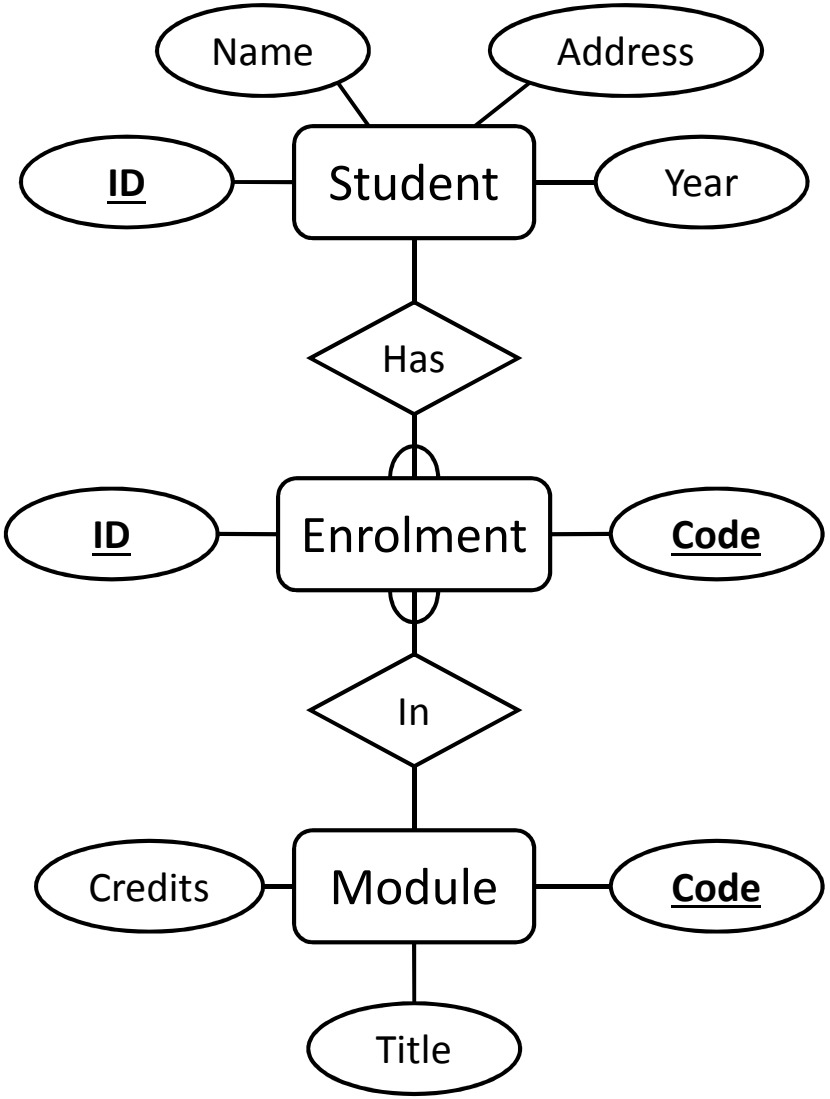
- **Foreign Keys are also defined as constraints**
- You need to provide
  - The columns which make up the foreign key
  - The referenced table
  - The columns which are referenced by the foreign key
- You can optionally provide reference options

```
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
```

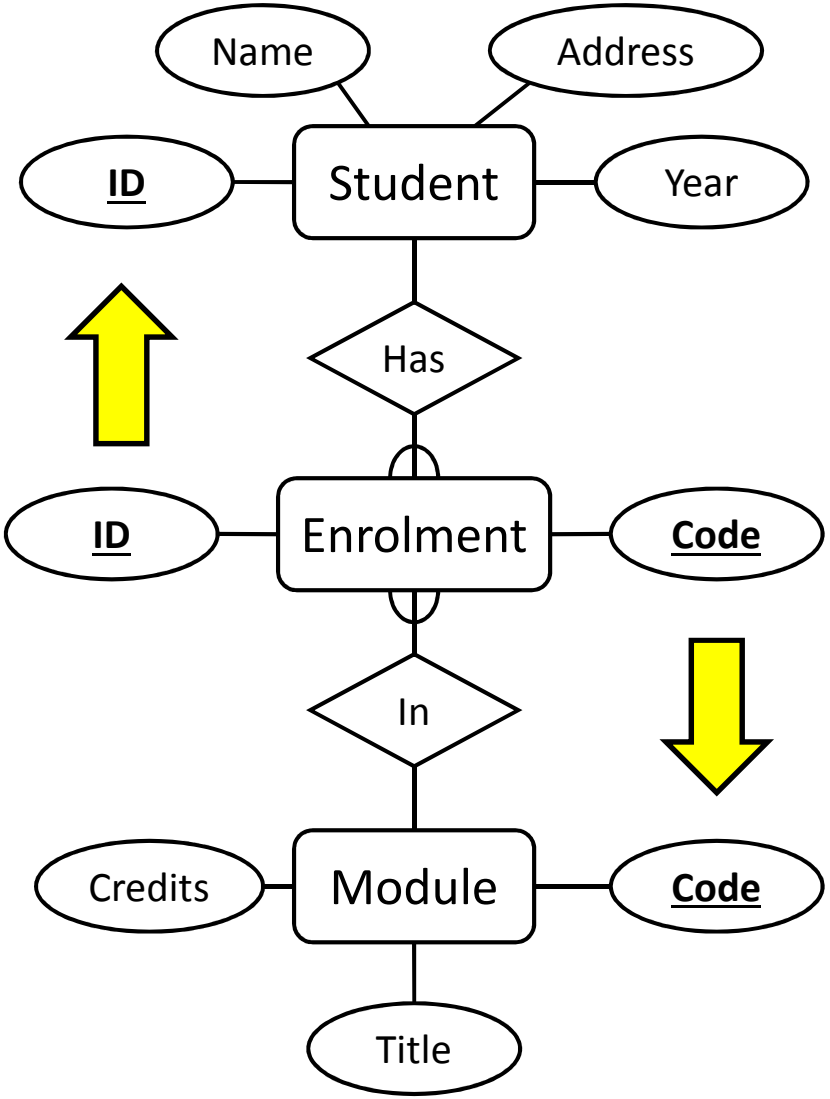
Example

```
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,
   CONSTRAINT en_fk2
      FOREIGN KEY (mCode)
      REFERENCES Module (mCode)
      ON UPDATE CASCADE
);
```



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,
   CONSTRAINT en_fk2
      FOREIGN KEY (mCode)
      REFERENCES Module (mCode)
      ON UPDATE CASCADE
);
```



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

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

G51DBS Database Systems

Jason Atkin

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>
);
```

Example tables

Student

sID	sName	sAddress	sYear
1	Smith	5 Arnold Close	1
2	Brooks	7 Holly Avenue	1
3	Anderson	15 Main Street	2

Module

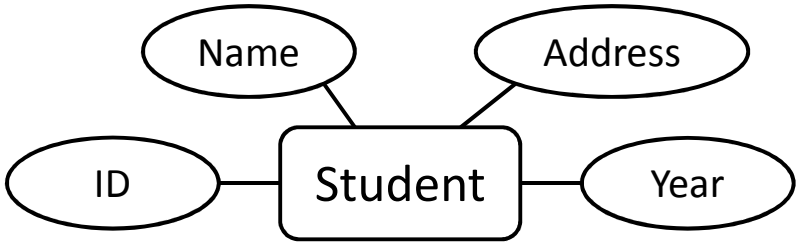
mCode	mCredits	mTitle
G51DBS	10	Database Systems
G51PRG	20	Programming
G51IAI	10	Artificial Intelligence
G52ADS	10	Algorithms

Enrolment

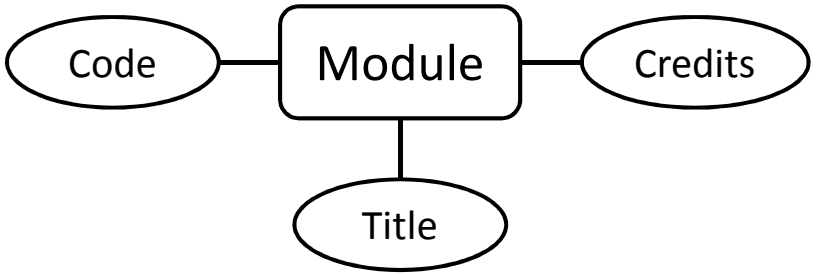
sID	mCode
1	G51DBS
1	G51PRG
1	G51IAI
2	G51DBS
2	G51PRG
3	G52ADS

Last Lecture's Example

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

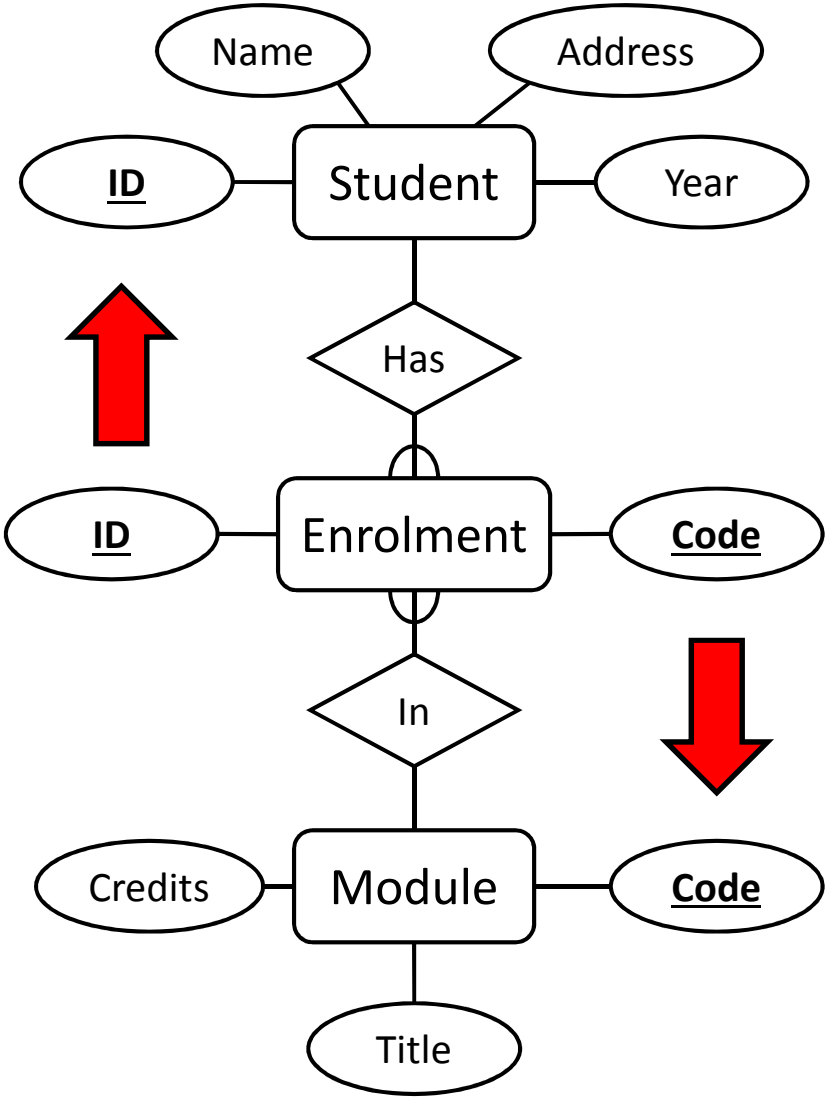



```
CREATE TABLE Module (
  mCode CHAR(6) NOT NULL,
  mCredits TINYINT
      NOT NULL DEFAULT 10,
  mTitle VARCHAR(100)
      NOT NULL,
  CONSTRAINT pk_mod
      PRIMARY KEY (mCode)
) ENGINE=InnoDB;
```



Last Lecture's 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,
   CONSTRAINT en_fk2
      FOREIGN KEY (mCode)
      REFERENCES Module (mCode)
      ON UPDATE CASCADE
);
```



This Lecture

- Storage Engines
- More SQL
  - Deleting tables : DROP TABLE
  - Changing table structures : ALTER TABLE
  - Adding, changing and deleting data
    - INSERT, UPDATE, and DELETE
  - The Information Schema
- Further Reading
  - Database Systems, Connolly and Begg, Chapter 6.3
  - The Manga Guide to Databases, Chapter 4

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 fast. Ignores all foreign key constraints
  - **InnoDB** – Offers transactions and foreign keys
  - **Memory** – Stored in RAM (extremely fast)
  - **Blackhole** – Deletes everything you put in it!

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 1
) ENGINE = InnoDB;
```

Note: **All** tables in a relationship must be InnoDB for foreign key constraints to work

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**

- You can delete multiple tables in single command, using a list, e.g.:

```
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 (drop the 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 (you lost the data)**, 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 a column to a table:

```
ALTER TABLE <table>
    ADD COLUMN <col-name>
    <col-def>
    [FIRST | AFTER <col2>]
```

- For example:

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

- To remove a column from a table:

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

- To remove a column from a table:

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

Altering Columns

- To change a column's name and/or definition:

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

- To change the definition of a column only:

```
ALTER TABLE <table>
    MODIFY COLUMN
    <col-name>
    <col-definition>
```

Note: Changing the column **type** 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 Constraints

- To add a constraint:

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

- For example:

```
ALTER TABLE Module
  ADD CONSTRAINT
    un_title
    UNIQUE (mTitle)
```

- To remove a constraint:

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

- Examples:

```
ALTER TABLE <table>
  DROP INDEX <name>
ALTER TABLE <table>
  DROP FOREIGN KEY <name>
ALTER TABLE <table>
  DROP PRIMARY KEY
```

Note: Primary key does not need a ‘name’ – there can only be one

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
G51DBS	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;
```

Module

mCode	mCredits	mTitle	lecID
G51DBS	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);
```

Module

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

Change engine to InnoDB:

- I was asked this in labs…

Source:
http://dev.mysql.com/doc/refman/5.0/en/converting-tables-to-innodb.html

To convert a non-InnoDB table to use InnoDB use ALTER TABLE:

ALTER TABLE t1 ENGINE=InnoDB;

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

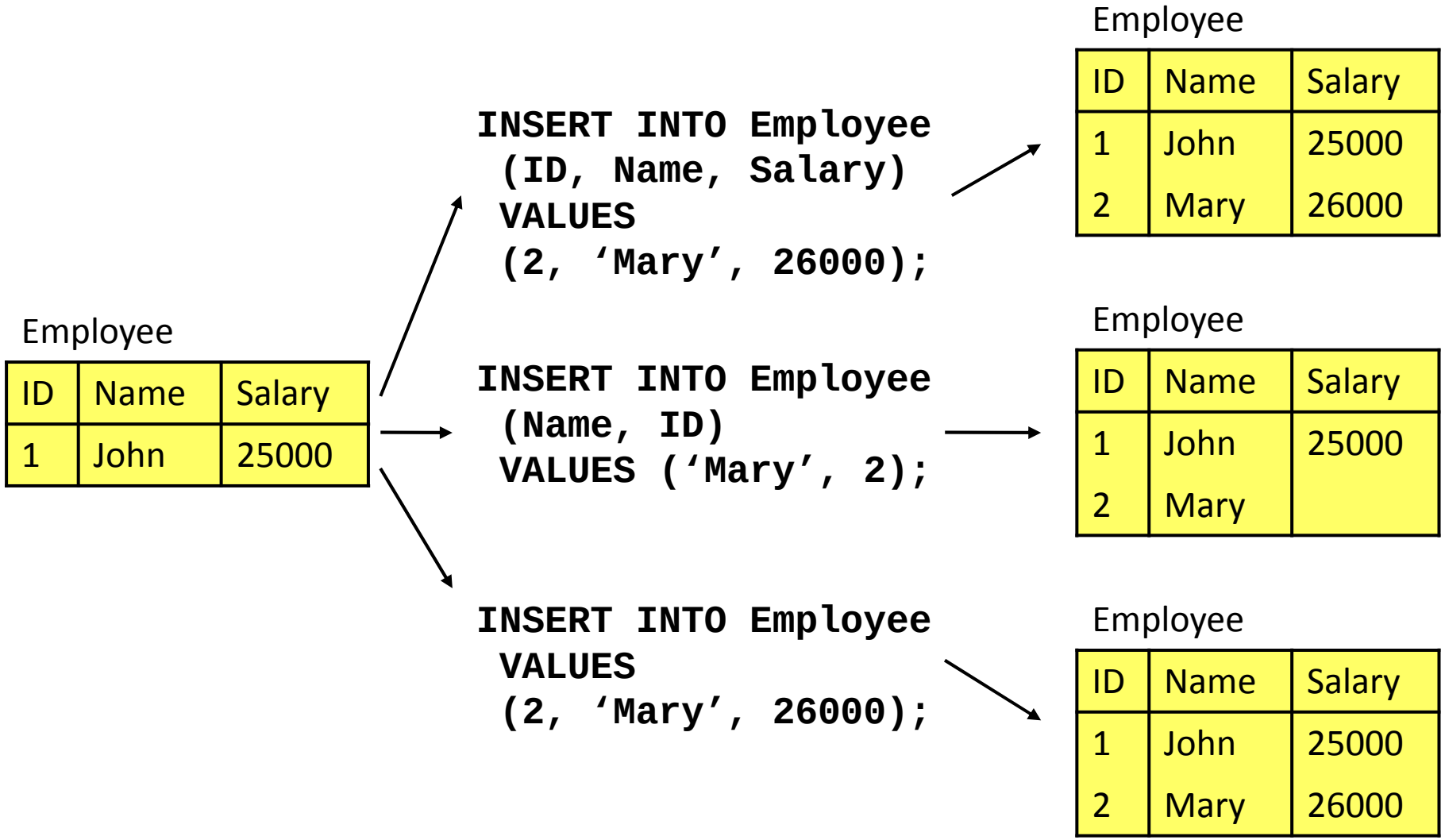
INSERT

- Inserts rows into the database with the specified values

```
INSERT INTO <table>
   (col1, col2, …)
   VALUES
   (val1, val2, …);
```

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

INSERT


Employee
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);
Employee
ID Name Salary
1 John 25000
2 Mary 26000
Employee
ID Name Salary
1 John 25000
2 Mary
Employee
ID Name Salary
1 John 25000
2 Mary 26000

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
 ('John', '5 Arnold Close'),
 ('Brooks', '7 Holly Ave.');
```

Student

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

UPDATE

- Changes values in specified rows based on SET conditions

```
UPDATE <table>
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 = 'Anne'
WHERE ID = 4
```

Employee

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

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

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>
 [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
- **i.e. Deletes ALL contents**

DELETE

Employee

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

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

Employee

ID	Name	Salary
3	Mark	18000
4	Jane	15000

```
DELETE FROM Employee;
```

Or

```
TRUNCATE TABLE
  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>;
```

<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;
```

Student

sName
Smith
Brooks
Anderson
Evans
Harrison
Jones

Sample SELECTs

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

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 condition (as the WHERE clause) first

e.g. Before running

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

run

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

to see what it will delete

Information Schema

- SQL '92 defines a set of system views that can be used to access metadata
- Metadata is 'data about data'. In this case, data about all of your tables
- This system database is called the information_schema
- Lots of DBMSs support this, but ***as usual*** support varies

- MySQL also gives us a few custom commands as well:
  - SHOW – Shows information on tables
  - DESCRIBE – Shows information on the columns in a specific table
- These are fine, but remember **SHOW is MySQL specific**, so don't become too reliant on it

Listing Tables

- To list all of your tables using SHOW:

```
SHOW tables;
```

- To use the information_schema to get this information:

```
SELECT TABLE_NAME
  FROM information_schema.tables
WHERE TABLE_SCHEMA = 'jaa';
```

information_schema tables

```
+---------------------------------------+
| TABLE_NAME                            |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| GLOBAL_STATUS                         |
| GLOBAL_VARIABLES                      |
| KEY_COLUMN_USAGE                      |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| PROFILING                             |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| SESSION_STATUS                        |
| SESSION_VARIABLES                     |
| STATISTICS                            |
| TABLES                                |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
+---------------------------------------+
```

Showing Columns

- You can describe a table using:

DESCRIBE <table-name>;

```
mysql> DESCRIBE Student;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| sID      | int(11)     | NO   | PRI | NULL    |       |
| sName    | varchar(64) | NO   |     | NULL    |       |
| sAddress | varchar(255)| YES  |     | NULL    |       |
| sYear    | tinyint(4)  | NO   |     | 1       |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

desc information_schema.tables

```
+-----------------+---------------------+------+-----+---------+-------+
| Field           | Type                | Null | Key | Default | Extra |
+-----------------+---------------------+------+-----+---------+-------+
| TABLE_CATALOG   | varchar(512)        | YES  |     | NULL    |       |
| TABLE_SCHEMA    | varchar(64)         | NO   |     |         |       |
| TABLE_NAME      | varchar(64)         | NO   |     |         |       |
| TABLE_TYPE      | varchar(64)         | NO   |     |         |       |
| ENGINE          | varchar(64)         | YES  |     | NULL    |       |
| VERSION         | bigint(21) unsigned | YES  |     | NULL    |       |
| ROW_FORMAT      | varchar(10)         | YES  |     | NULL    |       |
| TABLE_ROWS      | bigint(21) unsigned | YES  |     | NULL    |       |
| AVG_ROW_LENGTH  | bigint(21) unsigned | YES  |     | NULL    |       |
| DATA_LENGTH     | bigint(21) unsigned | YES  |     | NULL    |       |
| MAX_DATA_LENGTH | bigint(21) unsigned | YES  |     | NULL    |       |
| INDEX_LENGTH    | bigint(21) unsigned | YES  |     | NULL    |       |
| DATA_FREE       | bigint(21) unsigned | YES  |     | NULL    |       |
| AUTO_INCREMENT  | bigint(21) unsigned | YES  |     | NULL    |       |
| CREATE_TIME     | datetime            | YES  |     | NULL    |       |
| UPDATE_TIME     | datetime            | YES  |     | NULL    |       |
| CHECK_TIME      | datetime            | YES  |     | NULL    |       |
| TABLE_COLLATION | varchar(32)         | YES  |     | NULL    |       |
| CHECKSUM        | bigint(21) unsigned | YES  |     | NULL    |       |
| CREATE_OPTIONS  | varchar(255)        | YES  |     | NULL    |       |
| TABLE_COMMENT   | varchar(80)         | NO   |     |         |       |
+-----------------+---------------------+------+-----+---------+-------+
21 rows in set (0.00 sec)
```

'jaa' for me

Name of the table

More Information

- The **`information_schema.tables`** table has other columns that might be of use:

```
SELECT TABLE_NAME, TABLE_TYPE,
       TABLE_ROWS, ENGINE
  FROM information_schema.tables
 WHERE TABLE_SCHEMA = 'jaa';
```

- TABLE_TYPE – Table or view
- TABLE_ROWS – Current row count
- ENGINE – What storage engine the table uses

Showing More Detail

- You can show a little more information, including constraints with:

```
SHOW CREATE TABLE <table-name>;
```

```
mysql> show create table Student;
| Student | CREATE TABLE `Student` (
  `sID` int(11) NOT NULL,
  `sName` varchar(64) COLLATE utf8_unicode_ci NOT NULL,
  `sAddress` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `sYear` tinyint(4) NOT NULL DEFAULT '1',
  PRIMARY KEY (`sID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
1 row in set (0.00 sec)
```

Actually trying this…

- Login to the linux boxes (avon for you)
- If you have not created your database, do so and record your password (see instructions in lab introduction)
- Start mysql
- Use the command line – end commands with ;
- Or ‘source’ the text file
- Be aware of “show tables”, “desc <table>”, “select * from <table>”

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