SQL Lecture I

G51DBI – Databases and Interfaces

Yorgos Tzimiropoulos

yorgos.tzimiropoulos@nottingham.ac.uk

Overview of weeks 2-4

We will see how to translate English to Relational Algebra and SQL queries **and** vice versa

English: “Find all universities with > 20000 students”

Relational Algebra: $\pi_{uName}(\sigma_{enr > 20000}(University))$

SQL: Select uName From University Where University.enr>20000

Theory is easy and simple

But a sequence of simple operations is not always so obvious!

This Lecture

➢Intro to SQL
- Create Tables
- Data types
- Constraints
- Drop Tables
- Insert data, update data

SQL

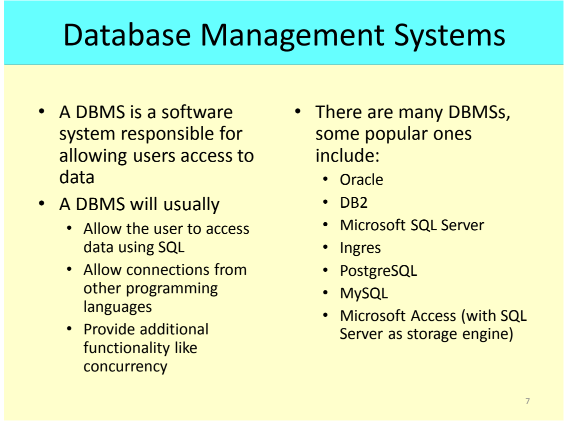
- Originally ‘Sequel’ - Structured English query Language, part of an IBM project in the 70’s
- Sequel was already taken, so it became SQL - Structured Query Language
- ANSI Standards and a number of revisions
  - SQL-89
  - SQL-92 (SQL2)
  - SQL-99 (SQL3)
  - …
  - SQL:2008 (SQL 2008)
- Most modern 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)

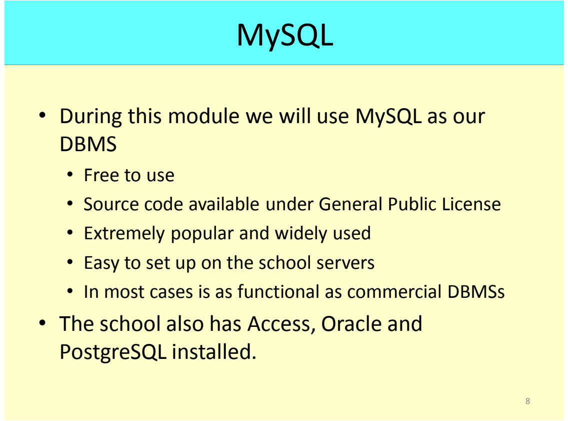
Provided Languages

- 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



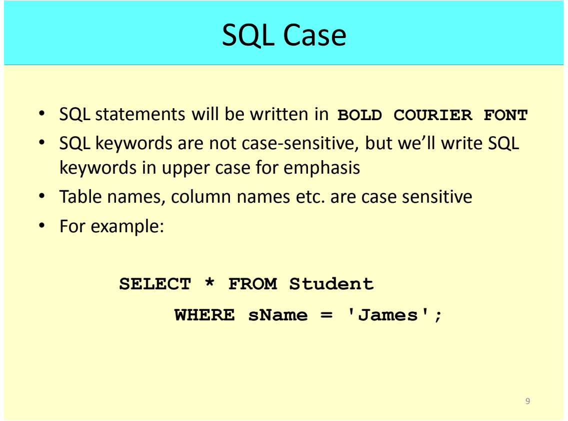
Database Management Systems

- 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)



MySQL

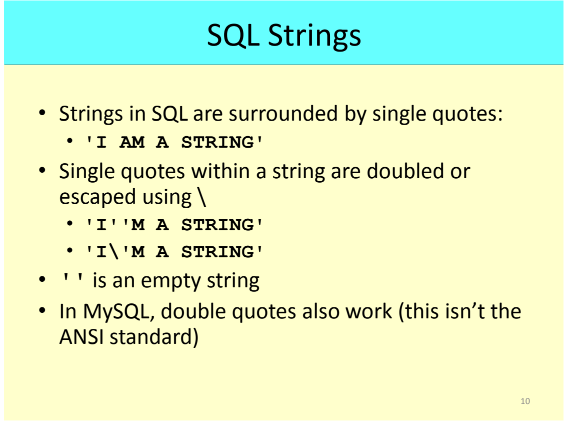
- During this module we will use MySQL as our DBMS
  - Free to use
  - Source code available under General Public License
  - Extremely popular and widely used
  - Easy to set up on the school servers
  - In most cases is as functional as commercial DBMSs
- The school also has Access, Oracle and PostgreSQL installed.



SQL Case

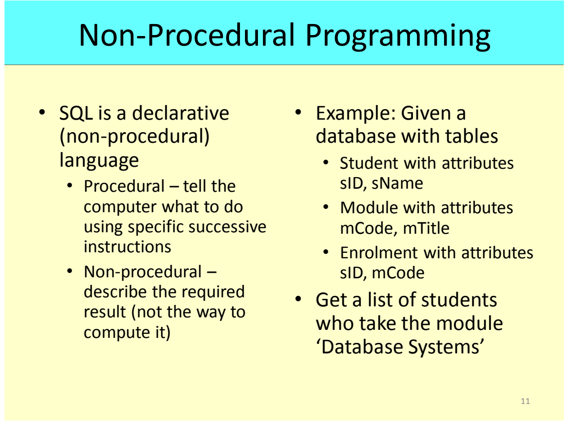
- SQL statements will be written in **BOLD COURIER FONT**
- SQL keywords are not case-sensitive, but we'll write SQL keywords in upper case for emphasis
- Table names, column names etc. are case sensitive
- For example:

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



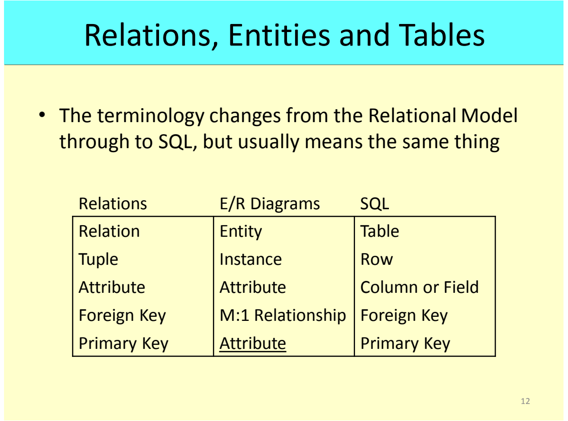
SQL Strings

- Strings in SQL are surrounded by single quotes:
  - **`'I AM A STRING'`**
- Single quotes within a string are doubled or escaped using \
  - **`'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 sID, sName
  - Module with attributes mCode, mTitle
  - Enrolment with attributes sID, mCode
- Get a list of students who take the module 'Database Systems'



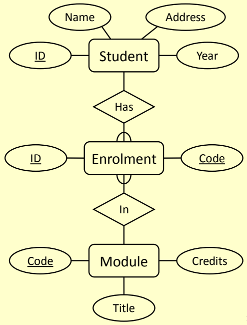
Relations, Entities and Tables

- 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;
```

- To use a database we need to type:

```
USE database-name;
```

- In School's servers you can't create databases
  - But still need to use one!

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

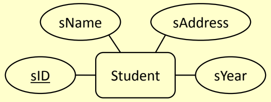
- Columns can be specified as **NULL** or **NOT NULL**
- **NOT NULL** columns cannot have missing values
- **NULL** is the default if you do not specify either

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

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

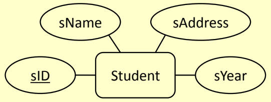
Example

Write the SQL statement to create a table for Student with the attributes listed below, where the sID number and the Student name cannot be null and, if not otherwise specified, students are in Year 1.



Example

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



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:

```
col-name INT AUTO_INCREMENT,
```

- When it comes to inserting values, you should use NULL, 0 or nothing to ensure you don't override the automatic value

Note: If the table auto_increment value isn't recalculated during deletes, you might want to reset it using:

```
ALTER TABLE table-name AUTO_INCREMENT=1;
```

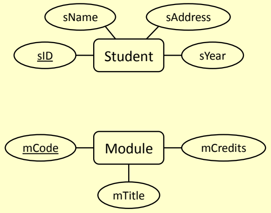
Example

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

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

Tips:

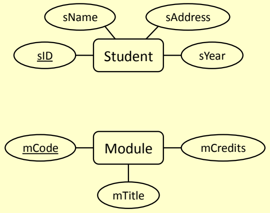
- Every module has a 6 characters code (e.g. G51DBI)

- 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
```

- 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, …)
```

Unique Constraints / CKs

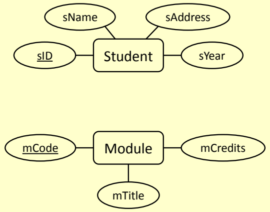
- As well as a single primary key, any set of columns can be specified as **UNIQUE**
- This has the effect of making candidate keys in the table

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

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

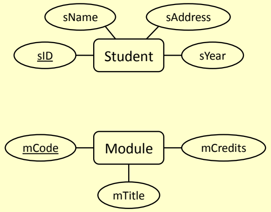
Example

```
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
);
```



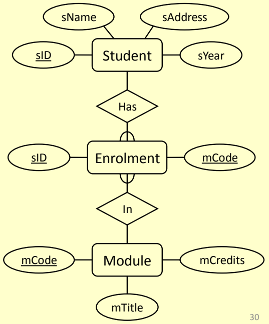
Example

```
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)
);
```



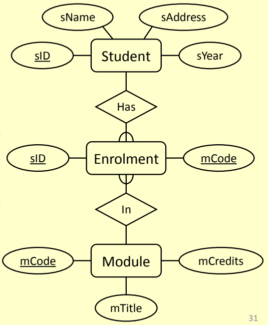
Relationships

- 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 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
  | SET DEFAULT
```

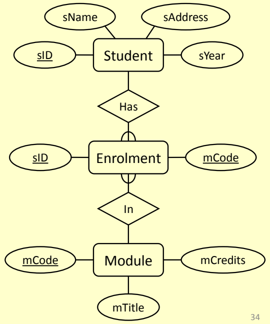
Set Default (Column Definition)

- 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

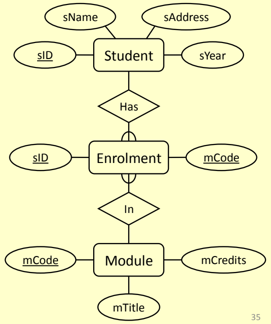
Example

```
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 fast. Ignores all foreign key constraints
  - **InnoDB** – Offers transactions and foreign keys
  - **Memory** – Stored in RAM (extremely fast)
  - **Others**

InnoDB

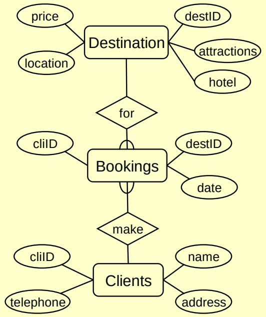
- We will use InnoDB for all tables during this module, for example:

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

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
```

- For example:

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

- To remove a constraint:

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

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);
```

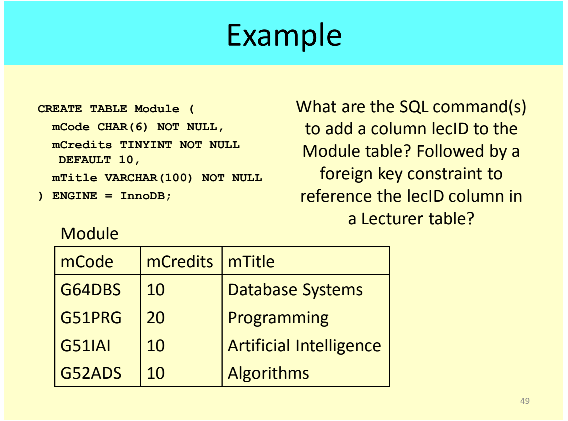
- To remove a constraint:

~~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



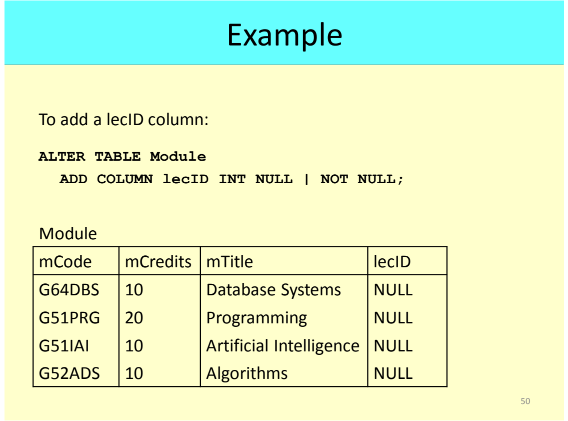
Example

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

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



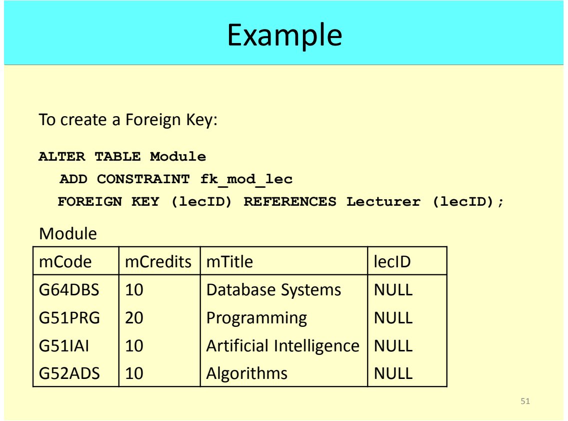
Example

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



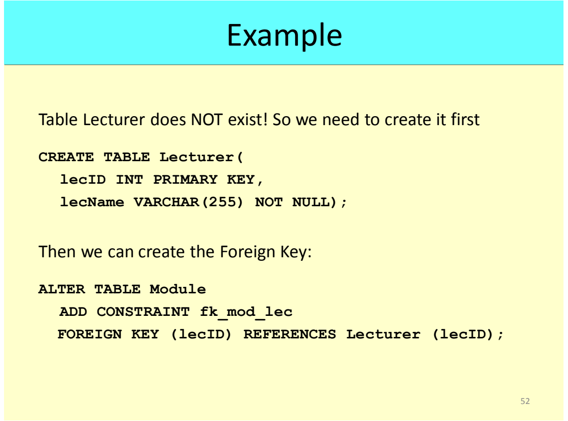
Example

To create a Foreign Key:

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

Module

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



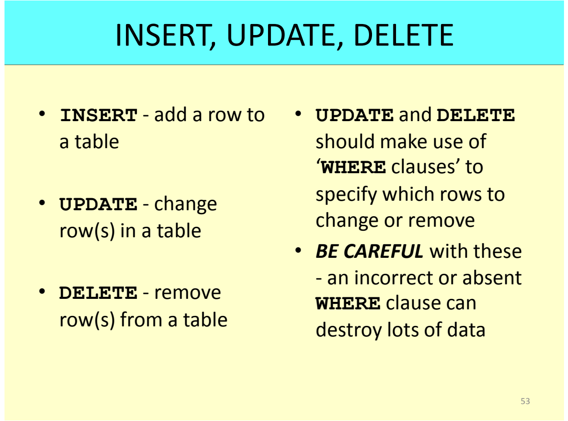
Example

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:

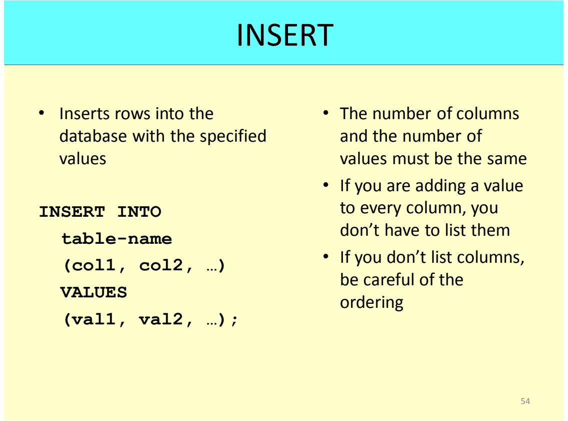
```
ALTER TABLE Module
   ADD CONSTRAINT fk_mod_lec
   FOREIGN KEY (lecID) REFERENCES Lecturer (lecID);
```



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

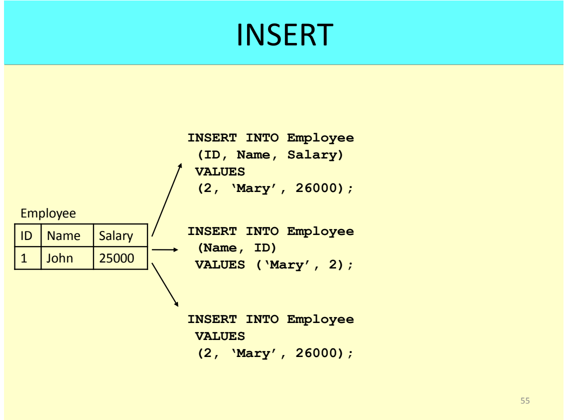


INSERT

- 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

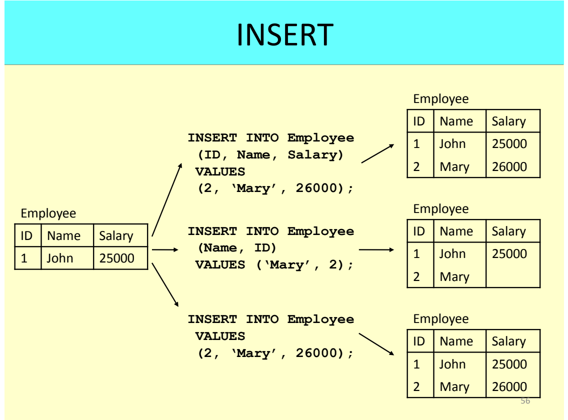
Employee

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);
```



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);
```

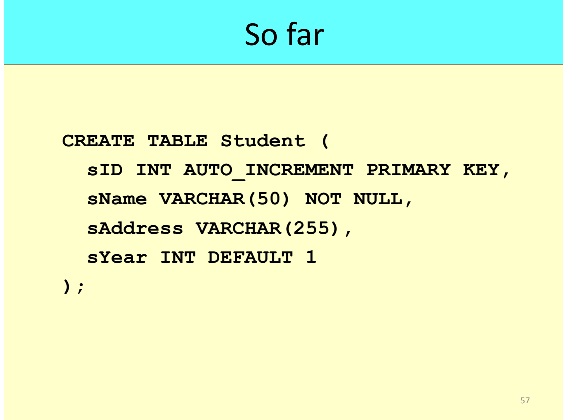
Employee

ID	Name	Salary
1	John	25000
2	Mary	

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

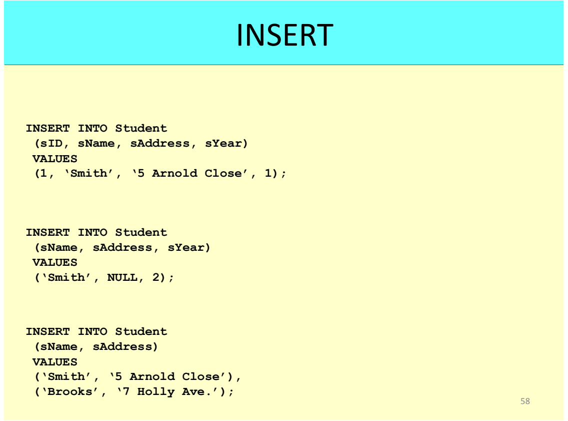
Employee

ID	Name	Salary
1	John	25000
2	Mary	26000



So far

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

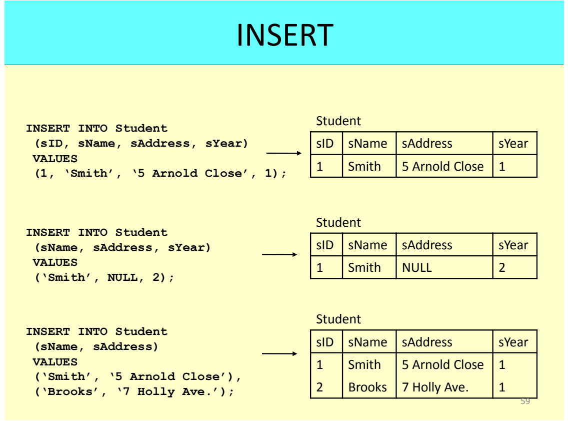


INSERT

```
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);
```

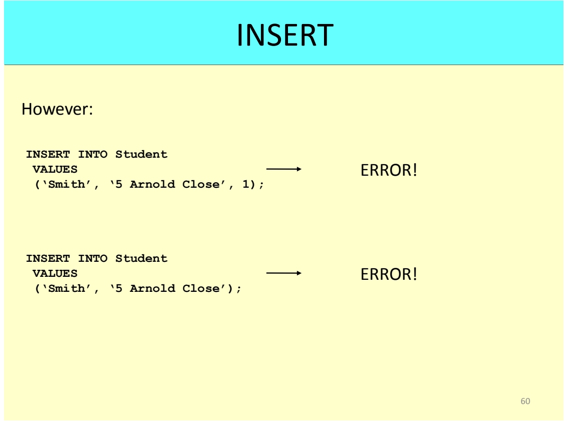
Student

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

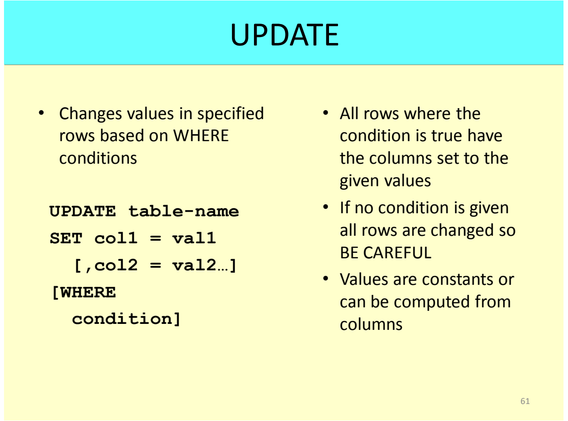
However:

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

→ ERROR!

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

→ ERROR!

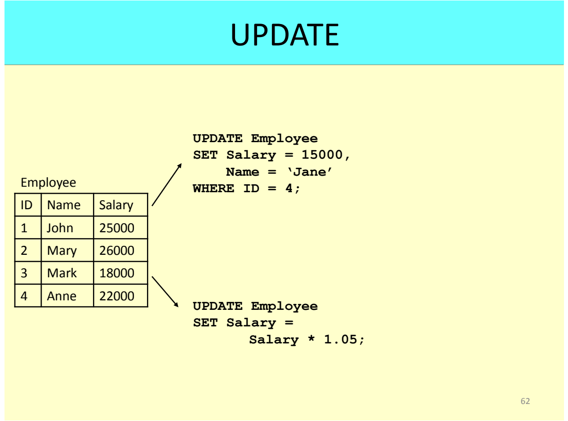


UPDATE

- 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



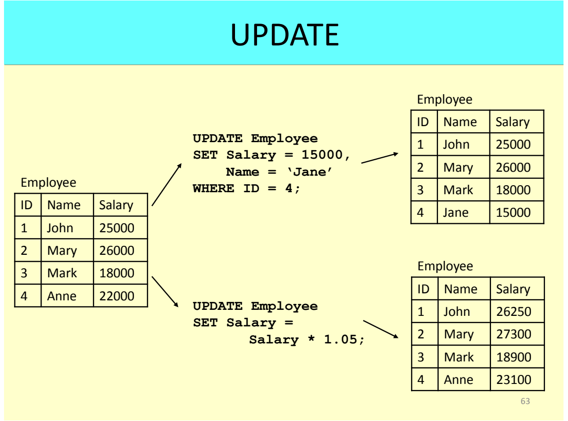
UPDATE

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 =
       Salary * 1.05;
```



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;
```

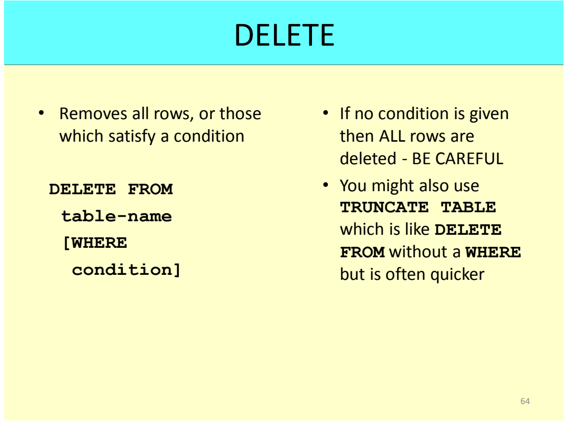
Employee

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

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

Employee

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

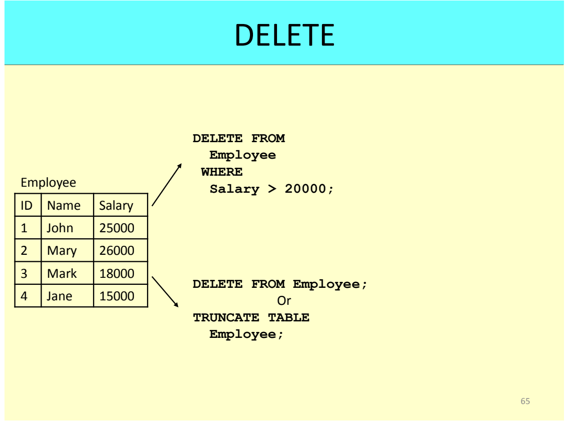


DELETE

- Removes all rows, or those which satisfy a condition

```
DELETE FROM
  table-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



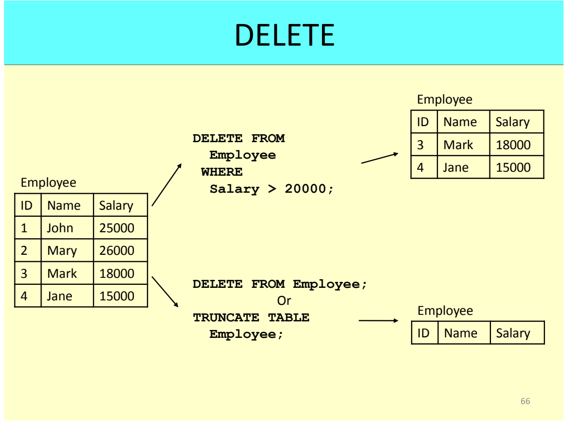
DELETE

Employee

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

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

```
DELETE FROM Employee;
          Or
TRUNCATE TABLE
  Employee;
```



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;
          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-name;
```

`columns` can be

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

Simple 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

Simple SELECTs

```
SELECT sName FROM Student;
```

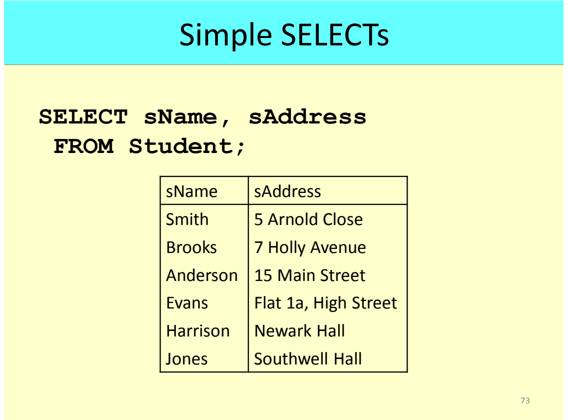
Simple SELECTs

```
SELECT sName FROM Student;
```

sName
Smith
Brooks
Anderson
Evans
Harrison
Jones

Simple SELECTs

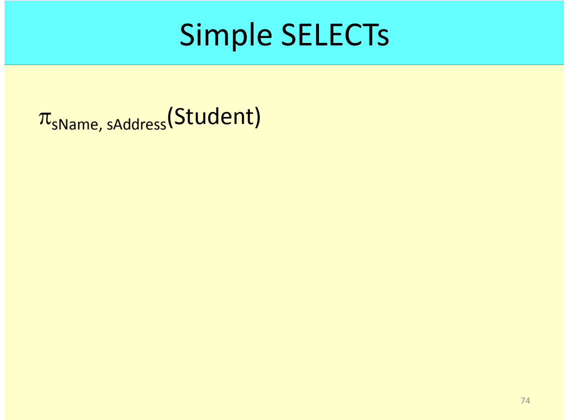
```
SELECT sName, sAddress
 FROM Student;
```



Simple SELECTs

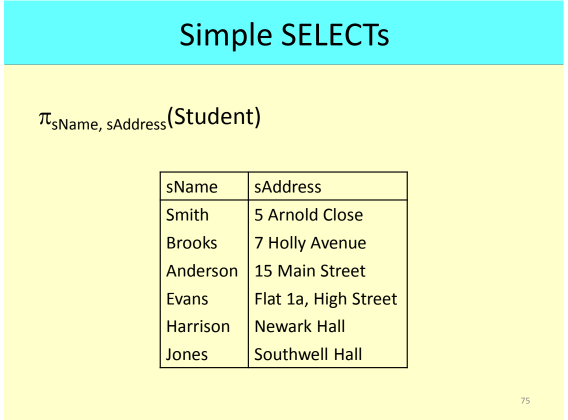
```
SELECT sName, sAddress
 FROM Student;
```

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



Simple SELECTs

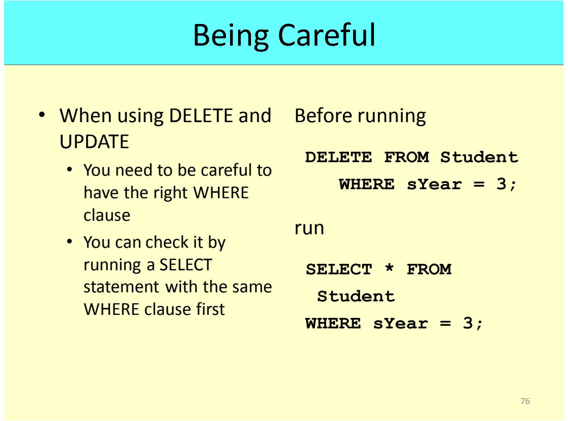
$\pi_{\text{sName, sAddress}}$(Student)



Simple SELECTs

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

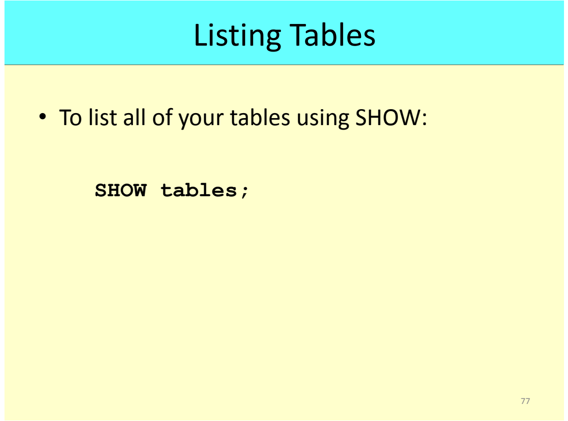
- When using DELETE and UPDATE
  - You need to be careful to have the right WHERE clause
  - You can check it by running a SELECT statement with the same WHERE clause first

Before running

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

run

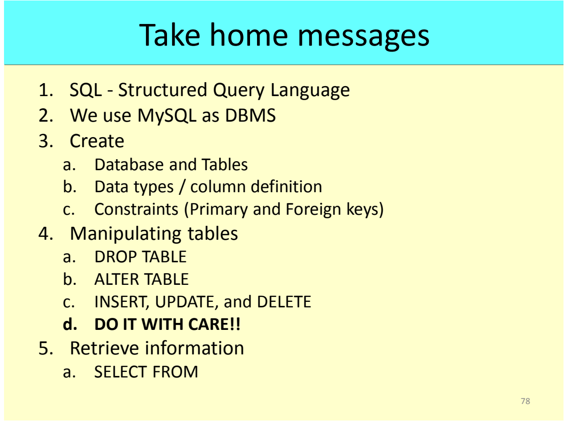
```
SELECT * FROM
 Student
WHERE sYear = 3;
```



Listing Tables

- To list all of your tables using SHOW:

```
SHOW tables;
```



Take home messages

1. SQL - Structured Query Language
2. We use MySQL as DBMS
3. Create
   a. Database and Tables
   b. Data types / column definition
   c. Constraints (Primary and Foreign keys)
4. Manipulating tables
   a. DROP TABLE
   b. ALTER TABLE
   c. INSERT, UPDATE, and DELETE
   d. DO IT WITH CARE!!
5. Retrieve information
   a. SELECT FROM

Thanks for your attention!

Any questions??