Structured Query Language - SQL

PUTTING OUR CREATIONS AND CONCEPTS TO WORK!

SQL Commands Overview

- SQL (**Structured Query Language)** is used to establish, populate, modify and query a database
  - SQL is used by virtually all modern relational DBMS
  - It functions as a standard query language
    - Used by Oracle, MS-SQL Server, MySql, PostgreSQL etc.
    - The same queries can be used between these platforms with minimal modification
    - There are several different standards ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….
      - Some vendors may implement additional special features
  - It is a **Declarative** programming language
    - It's a black box of magic – you as the user tell the database what you want, not how to get it!
    - It improves productivity by simplifying working with data

SQL Commands Overview

- SQL commands can be categorised:
  - **D**ata **D**efinition **L**anguage (**DDL** – add, modify and delete tables and attributes in a relational database)

```
CREATE TABLE Employee (
        Name varchar(200),
        BirthDate date,
        CONSTRAINT employeePk PRIMARY KEY (Name)
);
```


  - Data Manipulation Language (**DML** – add, modify, delete and retrieve data in a relational

```
SELECT * FROM Employee
WHERE DatePart(Y, BirthDate) >= 1990
ORDER BY Name;
```

SQL Commands Overview - SQL Syntax

- SQL commands have several important aspects:
  1. SQL contains **Keywords** that act on the table and attributes
     - Commands are **case insensitive**
     - **SELECT** = **select** = **SeLECt**
     - Upper case is often used to highlight keywords that evoke actions

  2. Semi-colons separate individual SQL statements by indicating where a statement ends:

```
INSERT INTO tableName VALUES (x,y,z);
SELECT * FROM tableName;
```

  3. SQL Commands and Statements ignore excess white space
     - They can be written in one long sentence or broken into separate lines of text
     - Breaking up statements onto individual lines helps with readability

SQL Overview - Reading SQL syntax definitions

- We use the following syntax in these slides
  - Italic/normal font indicates a value (number or name) that the user **must** provide
  - Bold/capitalised font means **keywords**
    - words whose meanings, usage and functions you should remember!
  - Elements in square brackets (and) can appear 0 or 1 times
  - Elements in braces ({ and }) can appear 0, 1 or more times
  - The | symbol delimits **alternative choices**
  - Angle brackets (< and >) are used with | to **group choices** and indicate something that **must appear exactly once**
  - **None of the above: ,,{,},<,>, and | are part of SQL. They are symbols used to help describe possible SQL statements**
    - If in doubt, search for demonstrations

Table Creation & Manipulation

SQL INTRODUCTION – DATA DEFINITION LANGUAGE

SQL Commands Overview - Table Manipulation

- Key Table Manipulation SQL commands:
  - **CREATE** - Creates a new table in the relational database
  - **ALTER** - Modifies an existing table in a relational database
  - **DROP** - Permanently removes a table in a relational database

SQL Overview - Table Definition

```
CREATE TABLE Employees (
  RegNo   char(6),
  FirstName        char(20),
  Surname          char(20),
  Dept     char(4),
  Salary   decimal(7,2) DEFAULT 0,
  Bdate    date NOT NULL,
 CONSTRAINT thPrimaryKey PRIMARY KEY (RegNo),
 CONSTRAINT uniqueNames UNIQUE  (Surname, FirstName),
 FOREIGN KEY (Dept) REFERENCES Departments (DeptName)
);
```

Clause

Attributes, Domains + Basic Constraints

Constraints

Employees – logical table/relational schema:
(RegNo,FirstName,SurName,Dept,Salary,Bdate)

SQL Commands Overview - SQL Data Types

- When creating a table using SQL each column of the relation must assigned a data type (called an attribute domain)
  - This dictates the type and length of future data to be stored
  - Common data types include:

Data Type	Description
char(n)	Stores a fixed-length n-character string (text)
varchar(n)	Variable length character string (text) of maximum size n characters
int	An integer number (whole number)
decimal(m, n)	A decimal number of m total digits and n decimal places
date	A date value (day/month/year) – if using British standard
datetime	A date time value (day/month/year hr:min:sec AM/PM)
bit	A Boolean value ('True'/'False' or 0/1)

Note: It is important to choose data types carefully:

- Only use Int/Decimal where calculations are concerned (not for streetNo's or postcodes!)
- Ensure you assign **enough** digits/characters
  - can you store the total value - allow for price increases especially sums
  - can you store someone's complete hyphenated name?

Attribute Domains – CHAR and VARCHAR

- Text is stored as strings of characters
  - **CHAR**|CHARACTER = a Fixed or Exact length string of characters
    - CHAR(4) = a string 4 characters long
      - More characters results in truncation/error
      - Fewer characters results in padding with white space
  - **VARCHAR**| CHARACTER VARYING = a Variable length string of characters
    - VARCHAR(4) = a string of up to 4 characters in length
      - More characters results in truncation
      - Fewer characters results in fewer characters stored (no padding with white spaces)
    - Varying key word included to save disk space in long strings
    - **VARCHAR(max)** will store the maximum allowed number of characters the database can hold for a value

- Can use a character set different from the default
  - Latin, Greek, Cyrillic, …

Attribute Domains – BIT, NUMERIC

- Single Boolean values
  - **BIT**
  - Can store 0|1 or use text: 'False'|'True'
    - (Note: you must use single quotes if using the text values)

- Exact values, integer or with a fractional part

  - Various alternatives in the standard
  - **DECIMAL**(Precision,Scale)
  - **INTEGER** | **INT**

  - There is another data type called **NUMERIC** but it is rarely used and has been replaced by decimal.

Attribute Domains - Numeric

- **DECIMAL**(5,2)
  - Stores a number containing 5 digits, 2 of which are decimal places
  - 5 = precision, 2 = scale
  - Numeric value from -999.99 to 999.99

REMEMBER: Choose your size (precision) carefully!
- Decimal(3,2) will only store values of items **up to $9.99**
- Choose carefully when storing growing totals!

- **INTEGER | INT**
  - Integer value, range of values implementation dependent

REMEMBER: Do **NOT** store general information in numeric fields!
- No House numbers (e.g. Unit 1A)
- No Phone numbers (e.g. 1800 CALL-ME, (08), +61, 0434 xxx xxx)
- No Account numbers (e.g. 0041 xxx xxx)
- Only values you need to add/subtract/multiply or ORDER BY

Attribute Domains - Elementary temporal

- Temporal instants
  - **DATE | DATETIME**
    - To get the current system DateTime use the function: GetDate()
  - TIME(Precision) with time zone
  - TIMESTAMP(Precision) with time zone
    - timestamp contains both date and time

- Temporal intervals (functions)
  - Dates can be added, subtracted etc. using **DATEADD()** function
  - Units of time are divided into their parts using **DATEPART()**
    - year, month
    - day, hour, minute, second

DatePart(part, theDate) see http://msdn.microsoft.com/en-au/library/ms174420.aspx
DateAdd(part, Number, theDate) see http://msdn.microsoft.com/en-us/library/ms186819.aspx

Attribute Domains - Default domain values

- These define the value of the attribute when it is not specified during row insertion

- **DEFAULT** <yourDefaultValue | NULL>
  - yourDefaultValue represents a value compatible with the domain, in the form of a constant or an expression
  - E.g specify 'True' as the default for a BIT field:
    - GoodStudent **BIT DEFAULT** 'True'
    - EnjoysDB **BIT DEFAULT** 'False'
    - DateCreated **DATETIME** GetDate()
  - If no default value is given, then **NULL** is used

NULL and NOT NULL

- **NULL** is the same as saying "I don't know"
  - NULL can have painful consequences in queries

Students(<u>studentID</u>, StudentName, GPA)

StudentID	StudentName	GPA
10014	Jennifer	5.6
11049	Deakon	*NULL*

Deakon only just started Uni and has not completed any courses yet

In SQL, other attributes like "StudentName" can be specified as **NOT NULL**

NULL and NOT NULL

- If you don't know the value of an attribute, put **NULL**

- In Relational Databases, **NULL** is a value entered to indicate many things
  - A Value that does not yet exist (to be entered later or was never collected)
  - An optional value (not relevant to the current record)
  - A missing value (the data was never captured for the given record or some other unknown reason)
- The relational schema specifies for each attribute if can be **NULL** value is allowed

  - This is different from software design where arbitrary values may be used
    - Eg in Java: -9999 or an unrealistic value may be used to represent a missing value to avoid exceptions or -1

NULL and NOT NULL

- How does the DBMS cope with tables that have NULL values ?
  - 'Jenifer' + **NULL** = NULL
    - This is a setting that can be changed!

Can I check this?

  - If x= NULL then 4*x is still NULL

  - In SQL there are three boolean values:
    - TRUE = 1 | 'True'
    - FALSE = 0 | 'False'
    - UNKNOWN = ?

The "UNKNOWN" can cause problems later. Sometimes it can return 'False' other times "UNKNOWN" depending on the query context :_

SQL Commands Overview – NULL value **Constraints**

- **NULL** Values ?
  - NULL values in keys cause problems
    - NULL values are NOT unique and cannot be used in **Primary Key** or **Unique Key** Constraints
- Example:
  - How do we access the third tuple?
  - Are the first and second tuples the same?
    - Are "John Lee" and "John White" referring to the same person??

StudentID	EmailID	Name	dateOfBirth	Address	Program	…
50001	NULL	John Lee	NULL	78 Main Street	LBCP	…
NULL	whij002	John White	23/7/1985	NULL	NULL	…
NULL	NULL	John Wilson	NULL	2 Smith Street	LBSG	…
50002	bump001	Peter Buman	3/9/1979	2 Smith Street	LBCP	…

Database Fundamentals

TABLE CREATION (PHYSICAL IMPLEMENTATION)

Properties of Relations – Concept Review

- Each **relation** within a database must have a distinct name
- Each **attribute** within a relation must have a distinct name
- Each **value** of an attribute must all be of the from the same domain (data type)
- Each **value** within a relation contains exactly one atomic value (no complex/multi-field data = 1NF)
- Each **tuple** (or row in a Table) is distinct (no duplicate tuples)

- Order of attributes has no significance
- Order of tuples has no significance

Logical schemas are implemented as **Tables** in a **Relational Database**

SQL Overview - Table Definition

- An SQL table (relation) consists of
  - An ordered set of attributes - (think column names)
  - The order is the order in which they were listed in the **CREATE TABLE** statement

- A set of zero or more constraints - (Think of these as rules about the data)
  - **NOT NULL**,
  - **CONSTRAINT** pkName **PRIMARY KEY** (col1, col2…)
  - **CONSTRAINT** uniqueName **UNIQUE** (col1, col2…)
  - **CONSTRAINT** fkName **FOREIGN KEY** (col1, col2…) **REFERENCES** otherTable(col1,col2…),
  - **CONSTRAINT** constraintName **CHECK** colName someTrueFalseExpression etc..

Used to represent **important** CKs

SQL Overview - Table Definition

- The **CREATE TABLE** statement defines a relational schema that creates an empty instance
  - It creates a table with no data!

- Think creating an excel sheet with rules about columns data.

- Syntax


```
CREATE TABLE TableName (
  AttributeName Domain DefaultValueColConstraints
  {, AttributeName Domain DefaultValueColConstraints}
  {CONSTRAINT conName, TableConstraint}
);
```

SQL Overview - Table Definition

```
CREATE TABLE Departments(
Dept char(4),
DeptName varchar(100) NOT NULL,
CONSTRAINT deptPk PRIMARY KEY(Dept),
CONSTRAINT uniqueName UNIQUE(DeptName)
);
```

Clause

Attributes, Domains
(+ Basic Constraints)

Constraints

Constraint Name

Is this correct?

Departments – logical table/relational schema:
(Dept, DeptName)

SQL Overview - Table Definition

```
CREATE TABLE Employees (
 RegNo char(6),
 FirstName      char(20),
 Surname        char(20),
 Dept   char(4),
 Salary decimal(7,2) DEFAULT 0,
 Bdate  date NOT NULL,
 CONSTRAINT employeePk PRIMARY KEY (RegNo),
 CONSTRAINT  uniqueName UNIQUE  (Surname, FirstName),
 CONSTRAINT  deptFk FOREIGN KEY (Dept) REFERENCES Departments (DeptName)
);
```

Clause

Attributes, Domains
+ Basic Constraints

Constraints

Is this correct?

Employees – logical table/relational schema:
(RegNo,FirstName,SurName,Dept,Salary,Bdate)

Implementation of the Student Info Database

Student

StudentID	EmailID	StudentName
50001	leej001	John Lee
50002	bump001	Peter Buman
50003	brod001	David Browns

Course

CourseID	CourseName
12529	Data modelling
12510	Java Programming
12524	Intel Sys Tech

Enrolment

Student	Course	Mark
50001	12529	80
50002	12510	75
50003	12510	85
50003	12529	89
50002	12529	50
50001	12524	85

SQL Commands Overview - CREATE TABLE

- Used to establish a table in a relational database
- Syntax:
  - CREATE TABLE <tableName> (table attributes listed here)
  - The list of attributes specifies the column names, data type and any additional constraints imposed on the data
  - Example:


```
CREATE TABLE Student (
StudentID char(5) PRIMARY KEY,
EmailID varchar(25) UNIQUE,
StudentName varchar(100) NOT NULL
);
```

OK Approach

```
CREATE TABLE Student (
StudentID char(5),
EmailID varchar(25) UNIQUE,
StudentName varchar(100) NOT NULL,
CONSTRAINT studentPk PRIMARY KEY (StudentID)
);
```

Better Approach

Student

StudentID	EmailID	StudentName
50001	leej001	John Lee
50002	bump001	Peter Buman
50003	brod001	David Browns

Student – logical table/relational schema: (StudentID, EmailID, StudentName)

SQL Commands Overview - CREATE TABLE

- Used to establish a table in a relational database
- Syntax:
  - CREATE TABLE <tableName> (table attributes listed here)
  - The list of attributes specifies the column names, data type and any additional constraints imposed on the data
  - Example:


```
CREATE TABLE Course (
      CourseID char(5) PRIMARY KEY,
      CourseName varchar(100) NOT NULL
);
```

OK Approach

```
CREATE TABLE Course (
      CourseID char(5),
      CourseName varchar(100) NOT NULL,
      CONSTRAINT coursePk PRIMARY KEY (CourseID)
);
```

Better Approach

Course

CourseID	CourseName
12529	Data modelling
12510	Java Programming
12524	Intel Sys Tech

Course– logical table/relational schema: (CourseID, CourseName)

SQL Commands Overview - CREATE TABLE

- Used to establish a table in a relational database
- Syntax:
  - CREATE TABLE <tableName> (table attributes listed here)
  - The list of attributes specifies the column names, data type and any additional constraints imposed on the data


```
CREATE TABLE Enrolment(
Student char(5) REFERENCES Student(StudentID),
Course char(5),
FOREIGN KEY (Course) REFERENCES Course(CourseID),
PRIMARY KEY (Student, Course)
);
```

OK Approach

```
CREATE TABLE Enrolment(
Student char(5),
Course char(5),
CONSTRAINT coursePk PRIMARY KEY (Student, Course),
CONSTRAINT studentFk FOREIGN KEY (Student) REFERENCES Student(StudentID),
CONSTRAINT courseFk FOREIGN KEY (Course) REFERENCES Course(CourseID)
);
```

Better Approach

Note: Foreign Key **data types** must match **EXACTLY** their primary key table data types & length

Enrolment

Student	Course	Mark
50001	12529	80
50002	12510	75
50003	12510	85

Enrolment – logical table/relational schema: (Student, Email)

Database Fundamentals

TABLE CREATION, DATA MANIPULATION. QUERY WRITING

TABLE CREATION - CONSTRAINTS

SQL Commands Overview - **CREATE TABLE Constraints**

- Students(StudentID, StudentName, GPA)
- Enrolment(StudentID, CourseID, Mark)

1. What if we insert a tuple into Enrolment, but no there is no corresponding student?

2. What if we delete a student?

1. Disallow the delete
2. Remove all of the enrolment records for that student
3. SQL allows a third via NULL

As the DBA you get to choose!

SQL Commands Overview - **CREATE TABLE Constraints**

- Constraints are clauses that need to be satisfied by data in the database
  - These statements are enforced by the DBMS but may also be imposed at the application/user interface level (e.g. A DropDownList, checkbox)
    - Ensures data validity
    - Constraints are how databases understand the semantics (meaning) of data
  - Constraints are like contracts to guard against bad data
  - Data that does not meet the rules of a given constraint will not be saved to the database
    - The whole tuple (new record) gets rejected and the DBMS throws an SQL Error
      - PRIMARY KEY Violation
      - FOREIGN KEY Violation
      - CHECK CONSTRAINT Violation etc…

SQL Commands Overview - CREATE TABLE Constraints

- Two major types:
  1. **INTRA** – Relational
     - Those that affect columns within a table
  2. **INTER** – Relational
     - Those that affect columns and values across tables


```
CREATE TABLE Enrolment(
        Student char(5),
        Course char(5),
        Mark int,
    1   CONSTRAINT EnrolmentPk PRIMARY KEY (Course, Student),
    2   CONSTRAINT CourseFk FOREIGN KEY (Course) REFERENCES  Course(CourseID),
    2   CONSTRAINT StudentFk FOREIGN KEY (Student) REFERENCES Student(StudentID),
    1   CONSTRAINT checkMark CHECK  (Mark >=0 AND Mark <= 100)
);
```

Enrolment

Student	Course	Mark
50001	12529	80
50002	12510	75
50003	12510	85

Note: Foreign Key **data types** must match **EXACTLY** their primary key table data types & length

Intra-Relational

CONSTRAINTS

Constraints - **Intra-relational** (Column)

- **INTRA-Relational** constraints come in two types:
  - **Column constraints** – checked each time the column value is modified
    - **NOT NULL**
    - **PRIMARY KEY** (for single attribute primary key)
    - **UNIQUE** (for single attribute alternate keys)
    - **CHECK** (described later)
  - Example:
    - FirstName char(20) **UNIQUE**
    - Surname char(20) **UNIQUE**


```
CREATE TABLE ColumnLevelConstraints  (
        Id INT PRIMARY KEY,
        StartDate DATE NOT NULL,
        EndDate DATE NOT NULL,
        DateChecked DATE CHECK(DateChecked > '01/Aug/2015') NOT NULL
)
```

OK Approach

```
CREATE TABLE ColumnLevelConstraints(
        Id INT,
        StartDate DATE NOT NULL,
        EndDate DATE NOT NULL,
        DateChecked DATE NOT NULL,
        CONSTRAINT thePK PRIMARY KEY (Id),
        CONSTRAINT dateCheck CHECK(DateChecked > '01/Aug/2015')
)
```

Better Approach

Constraints - **Intra-relational** (Table)

- **INTRA-Relational** constraints come in two types:
  - **Table constraints** – these are checked if any modification happens to the row regardless of the column value having changed or not
    - Can be used for **unique**, **primary key**, and **check**
    - Must be used if more than one attribute is in the constraints
  - Example:
    - FirstName char(20),
    - Surname char(20),
    - **UNIQUE** (**FirstName, Surname**)


```
CREATE TABLE TableLEvelConstraints
(
        Id INT PRIMARY KEY,
        StartDate DATE NOT NULL,
        EndDate DATE NOT NULL,
        dateChecked DATE NOT NULL,
        CONSTRAINT validDateCheck CHECK(dateChecked BETWEEN StartDate AND EndDate)
)
```

SQL Commands Overview - **CREATE TABLE** Constraints

- **INTRA-Relational** constraints come in two types – (placed on columns within a table)

  - Domain constraint
    - The Data Type (varchar(n), int, decimal(6,2) etc)
    - Ensures data is of the correct type
  - **PRIMARY KEY** constraint
    - PRIMARY KEY (StudentID)
    - Ensures a record does not get entered twice or more times
  - **UNIQUE** value constraints
    - UNIQUE (EmailID)
    - Forces each emailID to be unique (a candidate/alternate primary key)
  - **NOT NULL** value constraints
    - StudentName **NOT NULL**
    - Forces a value to be entered

SQL Commands Overview - **CREATE TABLE** Constraints

- **CHECK** Constraints
  - CHECK Mark >=0 AND Mark <= 100
  - Ensures a value is in a given range (can be dates, decimals etc)

- "*There cannot be more than 21 people in the DB practical class*"
  - In practice, we don't specify many such constraints. Why?

- Not a constraint but a **property**:
- **IDENTITY**
  - PRIMARY KEY (StudentID) **IDENTITY**
  - Makes the StudentID an auto-incrementing number (ie always unique!)
  - Only works for **int** data type

Inter-Relational

CONSTRAINTS

Constraints - **Inter-relational** (Table)

- **Inter-Relational Constraints** – (placed on columns between tables)
  - Foreign key (referential) constraints
    - **FOREIGN KEY** (Student) **REFERENCES** Student(StudentID)
    - Is a contract between tables to ensure a related record exists (i.e. no orphan records)

Constraints - **Inter-relational** (Table)

customer

acctNo	custName	custAddress
1101	Mary Jane	123 Lane
3311	James	424 Sa
2211	Bob	111 Drive

PK acctNo

customerPurchases

acctNo	product	storeName
1101	Coffee maker	HardlyNormal
3311	Milk	Foodville
1101	Knife and fork set	HardlyNormal
1101	Quilt set	Quilts'n'Things
2211	Coffee maker	HardlyNormal
3311	Milk	Coz

PK (acctNo, product, storeName)

FK (acctNo) → customer(acctNo)

- A **Key** is a <u>minimal</u> set of one or more attributes that can be used to identify each tuple uniquely (eg a TFN, studentID)

- A **Foreign Key** is a set of one or more attributes that define a relationship between entities (relations) in a relational model

Constraints - **Inter-relational** (Table)

- INTER-Relational Constraints run across several relations (tables)
  - **FOREIGN KEY**:
    - table constraint used to define a foreign key (for single or multiple attribute foreign keys)
    - **FOREIGN KEY** (Attribute {, Attribute}) **REFERENCES** TableName (Attribute {, Attribute})
  - Example


```
CREATE TABLE Hotel(
Hno char(4),
HotelName varchar(200),
HotelAddress varchar(200),
CONSTRAINT hotelPk PRIMARY KEY(Hno)
);
```

```
CREATE TABLE Room(
        Rno char(4),
        Hno char(4) REFERENCES Hotel(Hno),
        Type varchar(6),
        Price decimal(6,2),
        PRIMARY KEY(Rno, Hno)
);
```

OK Approach

```
CREATE TABLE Room(
        Rno char(4),
        Hno char(4),
        Type varchar(6),
        Price decimal(6,2),
        CONSTRAINT roomPk PRIMARY KEY(Rno, Hno),
        CONSTRAINT theHotel FOREIGN KEY (Hno) REFERENCES Hotel(Hno)
);
```

Better Approach

Constraints - **Inter-relational** (Table)

- INTER-Relational Constraints run across several relations (tables)
  - Constraint Naming:
    - Naming a constraint allows it to be drop from the table when needed (i.e. when altering the table)
    - Otherwise, the whole table has to be dropped and re-created: data lost


```
CREATE TABLE Room(
        Rno char(4),
        Hno char(4) REFERENCES Hotel(Hno),
        Type varchar(6),
        Price decimal(6,2),
        PRIMARY KEY(Rno, Hno)
);
```

OK Approach

```
CREATE TABLE Room(
        Rno char(4),
        Hno char(4),
        Type varchar(6),
        Price decimal(6,2),
        PRIMARY KEY(Rno, Hno),
        CONSTRAINT theHotel FOREIGN KEY (Hno) REFERENCES Hotel(Hno)
);
```

Your Constraint name!

Better Approach

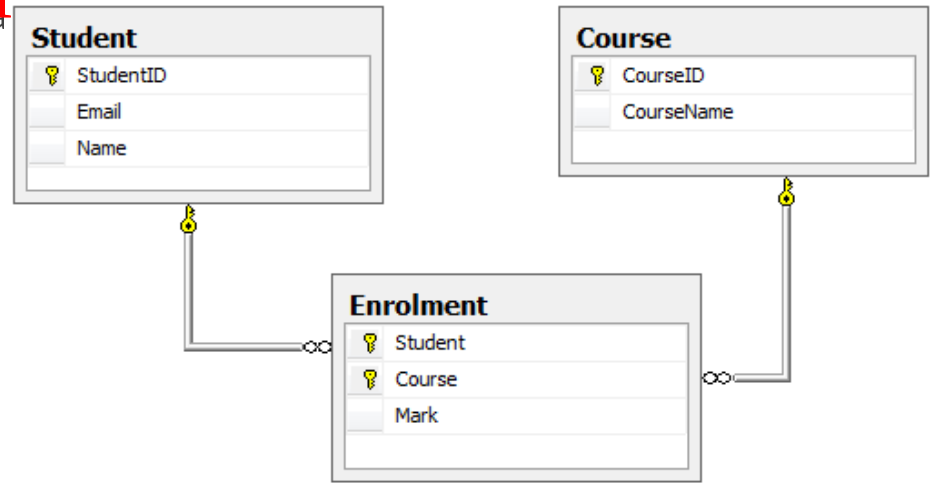
Constraints - **Inter-relational** Reaction Policy

- **Foreign key** constraints can have reaction policies in response to violations of referential integrity
  - These operate on the referencing (Secondary) table, after changes to the referenced (Primary) table
    - Violations may be introduced by updates on the referenced attribute or by row deletions
    - The reaction policy restores referential integrity or prevents the change from taking place (i.e. terminates the query with an error)
      - E.g., Deleting a hotel → Delete all the Rooms? Bookings?
  - Syntax
    - on <delete | update>
      <cascade | set NULL | set default | no action>
  - Example:

```
CREATE TABLE  Room (
…,
FOREIGN KEY(HNo) REFERENCES Hotel (HNo)
ON UPDATE CASCADE
ON DELETE CASCADE
);
```

Constraints - **Inter-relational** Reaction Policy

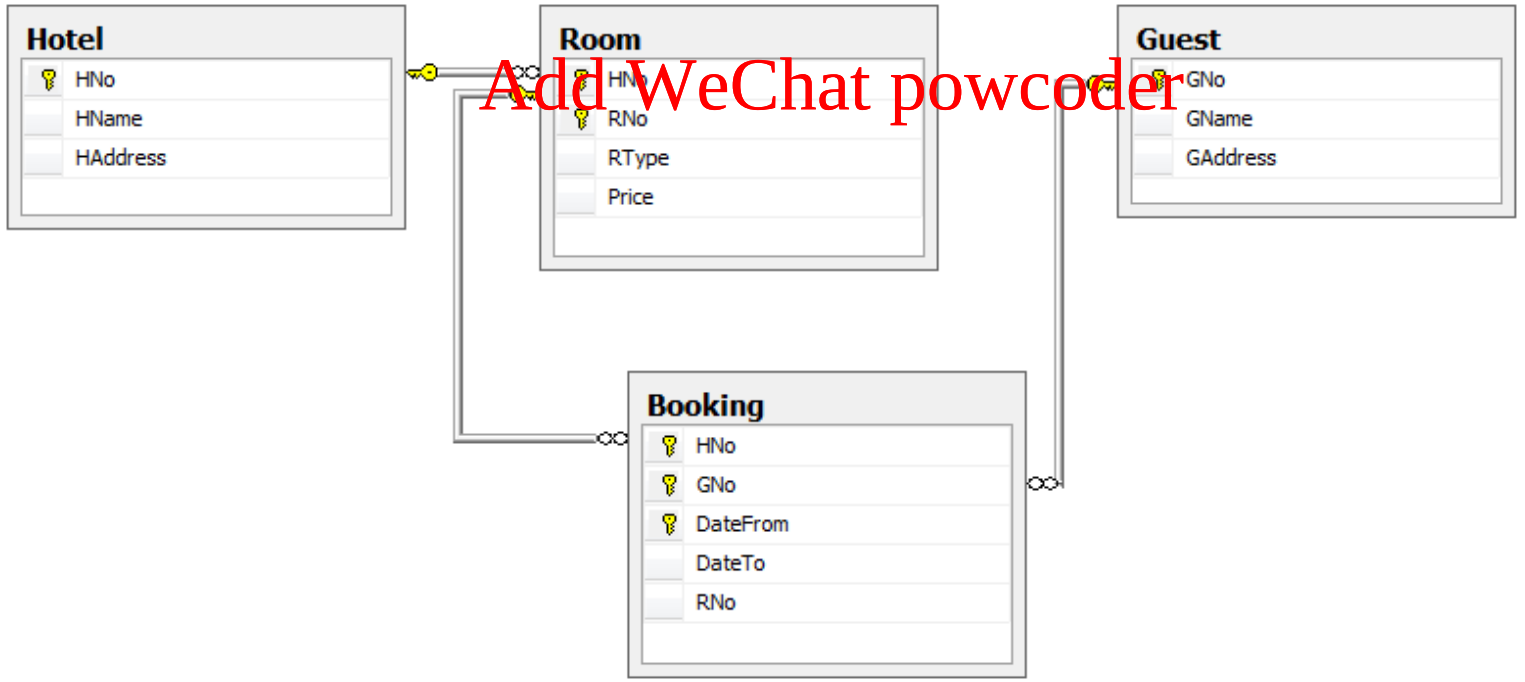
- Consider the enrolment table referencing the student table when a student is deleted
  - **ON DELETE NO ACTION**: forbid the deletion if any enrolments exist for the student.
    - **Do not permit the change –** default behavior
  - **ON DELETE CASCADE**: delete all enrolments for the student
    - Are you going to lose anything important?
  - **ON DELETE SET NULL:** set id to NULL for any related enrolments
    - Does the FK column allow NULL Values? (is it part of a PK/NOT NULL!?) – Orpha
  - **ON DELETE SET DEFAULT**: assign the default ID value to all enrolments for the student



Constraints - **Inter-relational** Reaction Policy

- Foreign key constraints can have reaction policies in response to violations of referential integrity
  - Example:

```
CREATE TABLE  Room (
…,
FOREIGN KEY(HNo) REFERENCES Hotel (HNo)
ON UPDATE CASCADE
ON DELETE CASCADE
```

Hotel Database Example

```
CREATE TABLE Hotel(
Hno char(4),
HotelName varchar(200),
HotelAddress varchar(200),
CONSTRAINT hotelPk PRIMARY KEY(Hno)
);
```

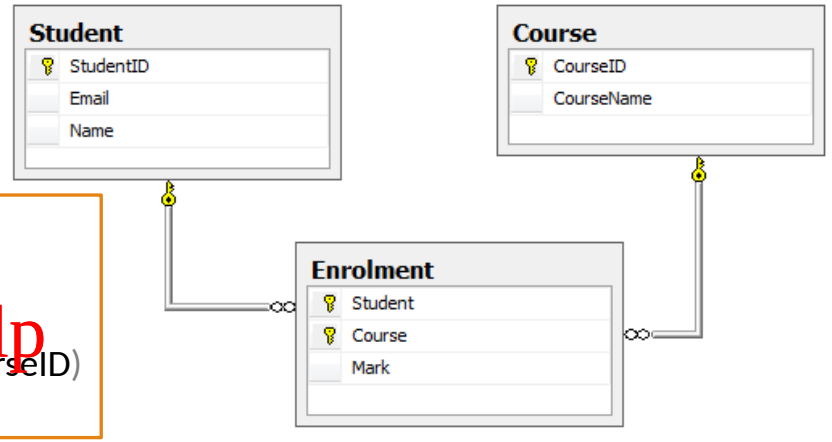
```
CREATE TABLE Room(
        Rno char(4),
        Hno char(4),
        Type varchar(6),
        Price decimal(6,2),
        CONSTRAINT rookPk PRIMARY KEY(Rno, Hno),
        CONSTRAINT theHotel FOREIGN KEY (Hno) REFERENCES Hotel(Hno)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```



```
CREATE TABLE Booking(
Hno char(4),
Gno char(4),
dateFrom date NOT NULL,
dateTo date,
Rno char(4),
CONSTRAINT bookingPk PRIMARY KEY(Hno, Gno, dateFrom),
CONSTRAINT hotelRoom FOREIGN KEY (Rno, Hno) REFERENCES Room (Rno, Hno)
ON  DELETE CASCADE ON UPDATE CASCADE
);
```

Student Database Example

```
CREATE TABLE Student(
StudentID char(5),
Email varchar(50) NOT NULL,
Name varchar(100) NOT NULL,
CONSTRAINT studentPk PRIMARY KEY (StudentID),
CONSTRAINT uniqueEmail UNIQUE(Email)
);
```




```
CREATE TABLE Course (
CourseID char(5),
CourseName varchar(100) NOT NULL,
CONSTRAINT coursePk PRIMARY KEY(CourseID)
);
```



```
CREATE TABLE Enrolment (
Student char(5),
Course char(5) REFERENCES Course(CourseID)
  ON UPDATE CASCADE
  ON DELETE CASCADE,
Mark int,
CONSTRAINT validMark CHECK  (Mark >=0 AND Mark <= 100),
CONSTRAINT enrolmentPk PRIMARY KEY (Student,Course),
CONSTRAINT theStudent FOREIGN KEY(Student) REFERENCES Student(StudentID)
   ON UPDATE CASCADE
   ON DELETE CASCADE
);
```

Alt: Check (Mark BETWEEN 0 AND 100)

One big clause!!

Constraints - Naming

- Naming a constraint allows you to drop it from the table when needed
  - Otherwise, the whole table has to be dropped and re-created: data lost
  - It can also help with debugging a query when reading error messages

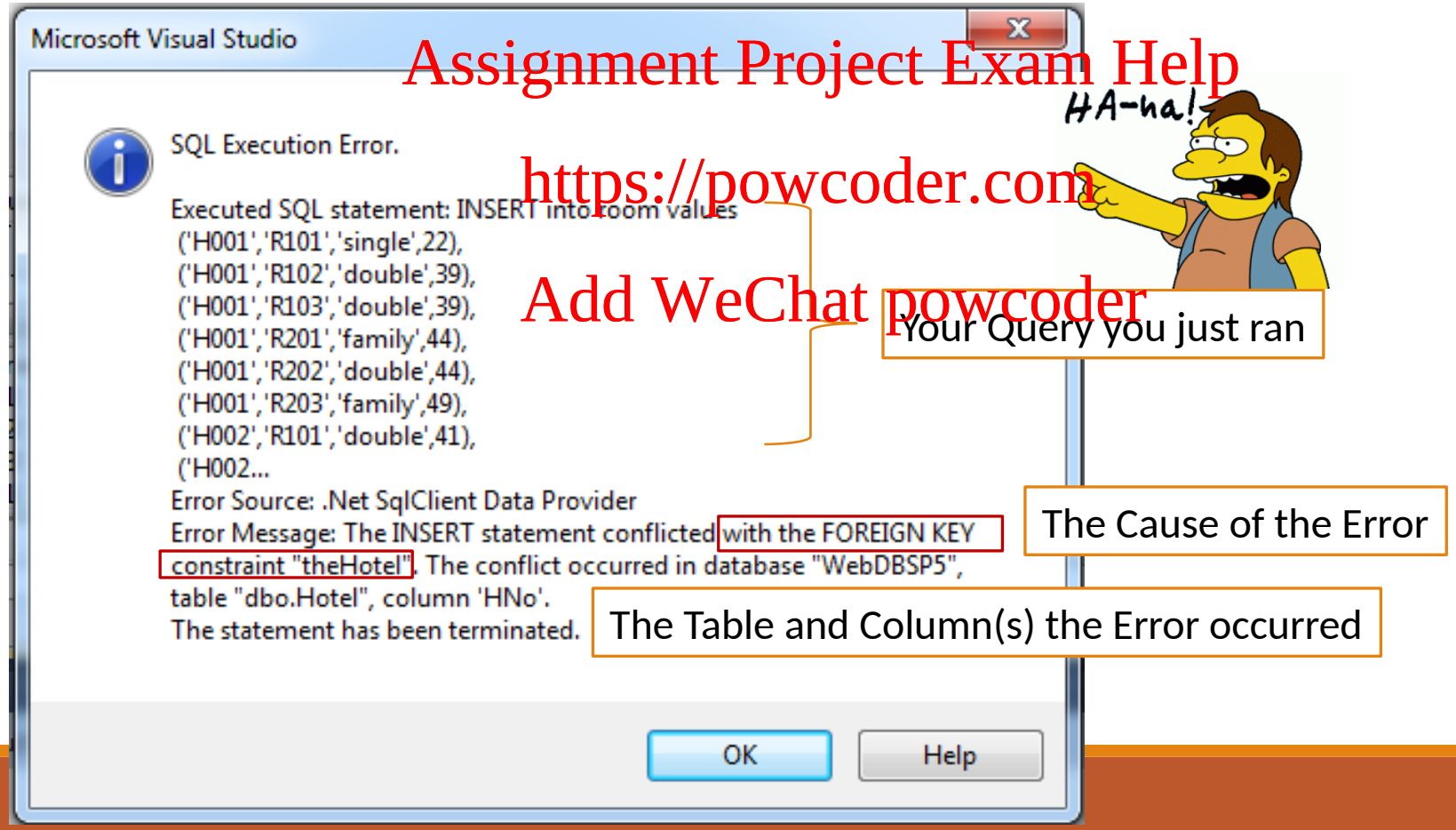

```
CREATE TABLE Enrolment(
Student char(5),
Course char(5) REFERENCES Course(CourseID)
  ON UPDATE CASCADE
  ON DELETE CASCADE,
Mark int,
CONSTRAINT validMark  CHECK  (Mark >=0 AND Mark <= 100),
CONSTRAINT enrolmentPk PRIMARY KEY (Student,Course),
CONSTRAINT EnrolmentStudentID   FOREIGN KEY(Student)
  REFERENCES Student(StudentID)
  ON UPDATE CASCADE
  ON DELETE CASCADE
);
```

Same Enrolment table but with constraint names!

Don't forget brackets!!!

Constraints - Naming

```
Msg 547, Level 16, State 0, Line 1
The INSERT statement conflicted with the FOREIGN KEY constraint "theHotel".
The conflict occurred in database "WebDBSP5", table "dbo.Hotel", column 'HNo'.
The statement has been terminated.
```



Your Query you just ran

The Cause of the Error

The Table and Column(s) the Error occurred

Impact of Table Constraints

HOW TABLE CONSTRAINTS IMPACT ON ADDING NEW RECORDS

SQL Commands Overview - INSERT INTO **Constraints**

- Domain (Data Type) Constraints
  - All data in a given column must be of the same type
  - In the newly created Enrolment table, trying to add a descriptive mark will throw an error:


```
CREATE TABLE Enrolment(
Student char(5),
Course char(5),
Mark int,
PRIMARY KEY (Course, Student),
FOREIGN KEY (Course) REFERENCES  Course(CourseID),
FOREIGN KEY (Student) REFERENCES Student(StudentID),
CHECK  (Mark >=0 AND Mark <= 100),
);

INSERT INTO enrolment VALUES ('50001', '12510', 'HD');
```

Enrolment

StudentID	CourseId	Mark
50001	12529	80
50002	12510	75
50003	12510	HD

A ‘HD’ is not an integer!

SQL Commands Overview - INSERT INTO **Constraints**

- **CHECK** (Tuple) Constraints
  - All data in a given column must be within a given range
  - Marks must be >= 0 and <= 100


```
CREATE TABLE Enrolment(
Student char(5),
Course char(5),
Mark int,
PRIMARY KEY (Course, Student),
FOREIGN KEY (Course) REFERENCES  Course(CourseID),
FOREIGN KEY (Student) REFERENCES Student(StudentID),
CHECK  (Mark >=0 AND Mark <= 100),
);

INSERT INTO enrolment VALUES ('50001', '12510', 780);
```

Enrolment

StudentID	CourseId	Mark
50001	12529	80
50002	12510	75
50003	12510	780

780 is greater than 100!!!

SQL Commands Overview - INSERT INTO **Constraints**

- **PRIMARY KEY** Constraints
- A key is a minimal set of attributes the value combinations of which are unique in the table
  - Minimal means it is composed from as few value combinations as possible. Sometimes, the set has only one attribute (eg StudentID)
- Key is used to uniquely identify each tuple
  - No two tuples in a table should have the same key value.


```
CREATE TABLE Student (
StudentID char(5) PRIMARY KEY,
StudentName varchar(100) NOT NULL
);

INSERT INTO Student VALUES('5001', 'John Lee');
INSERT INTO Student VALUES('5001', 'Peter Buman');
```

Student

StudentID	StudName
50001	John Lee
50001	Peter Buman
50003	David Browns

The same StudentID cannot be inserted twice

SQL Commands Overview - INSERT INTO **Constraints**

- **UNIQUE** Constraints
  - A relation can have many keys. E.g. StudentID is a key and EmailID is another key BUT only one of them can be the primary key (PK)
    - All others are called candidate/alternate keys (CKs)


```
CREATE TABLE Student (
StudentID char(5) PRIMARY KEY,
EmailID varchar(25) UNIQUE,
StudentName varchar(100) NOT NULL
);
```

Student

StudentID	EmailID	StudName
50001	lee001	John Lee
50002	lee001	Peter Buman
50003	bro001	David Browns

```
INSERT INTO Student VALUES('5001', 'lee001', 'John Lee');
INSERT INTO Student VALUES('5001', 'lee001', 'Peter Buman');
```

The same EmailID cannot be inserted twice

SQL Commands Overview – NULL value **Constraints**

- **NOT NULL** Value Constraint
  - Prevents no value being entered for the specified column
    - The constraint requires that the specified attribute for all existing tuples must not be empty or blank
- Example:


```
CREATE TABLE Course (
CourseID char(5) PRIMARY KEY,
CourseName varchar(100) NOT NULL
);
```

Course

CourseID	CourseName
12529	Web & DB
12510	(NULL)
12530	(NULL)

```
INSERT INTO Course VALUES ('12529', 'DB Fundamntals');
INSERT INTO Course VALUES('12510', NULL);
INSERT INTO Course (CourseID) VALUES ('12530')
```

The CourseName cannot be Empty/NULL

Database Fundamentals

TABLE CREATION, DATA MANIPULATION. QUERY WRITING

TABLE MODIFICATION

Relational Schema Modifications

- Two SQL key words are defined to update table schemas: **ALTER** and **DROP**
  - **ALTER TABLE** TableName …
  - **ALTER DOMAIN** DomainName …
  - **DROP TABLE** TableName restrict | cascade
  - **DROP DOMAIN** DomainName restrict | cascade

- Examples

```
ALTER TABLE Student ADD  Address varchar(100);
ALTER TABLE Course ADD maxClassSize int DEFAULT 30;
DROP TABLE Course CASCADE;
```

For Full Example see: http://msdn.microsoft.com/en-au/library/ms190273.aspx

SQL Commands Overview - **ALTER TABLE**

- Used to add modify an existing relation:
  - Adding new attributes
    - **ALTER TABLE** tableName **ADD** newAttribute dataType;
    - **ALTER TABLE** Student **ADD** dateOfBirth Date;
  - Removing existing attributes
    - **ALTER TABLE** Student **DROP** dateOfBirth;
  - You should **not need to do this** if you have designed your database correctly!

SQL Commands Overview - **DROP TABLE**

- Used to completely remove a table from an existing database

  DROP TABLE Enrolment

  DROP TABLE Student

  DROP TABLE Course

- Note: where Foreign Keys are involved the order in which tables are dropped is important.

  - Course and Student tables are referred to by the Enrolment table

    → They cannot be dropped before Enrolment as this would violate the Foreign key constraints (contracts)

SQL Commands Overview – Table Creation

- In practice as developers and system administrators we rarely write table creation statements
  - CASE tools are used instead (Computer Aided Software Engineering)
    - Unified Modelling Language (UML Diagram) → SQL generation tools (DBDesigner Fork etc)
    - MS-SQL Diagram → Relational database table tool
    - MS Graphical Table editor
    - All are Graphical and some even bypass the SQL