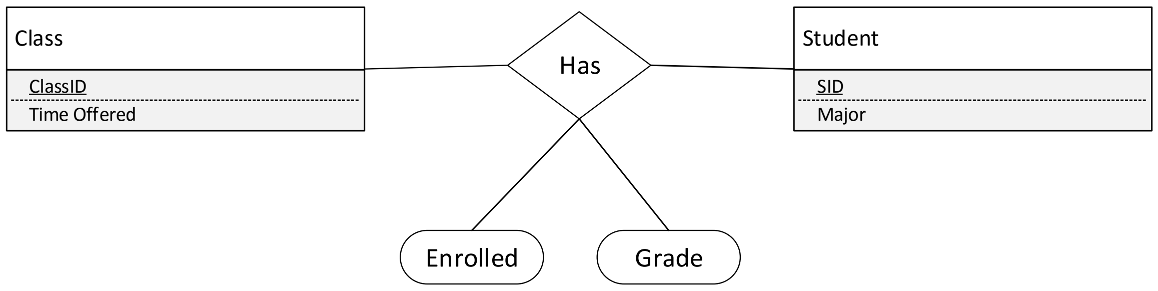
Referential integrity

Another function of the DBMS is to ensure referential integrity. This refers to the concept that a foreign key in a relation must have a corresponding entry in the table to which it refers. Let's look at the following ER diagram:



This ER diagram results in the following SQL:

```
CREATE TABLE Class (
      ClassID CHAR(20),
      TimeOffered CHAR(20),
      PRIMARY KEY (ClassID));

CREATE TABLE Students (
      StudentID INTEGER,
      Major CHAR(20),
      PRIMARY KEY (StudentID));

CREATE TABLE Enrolled (
      StudentID INTEGER,
      ClassID CHAR(20),
      Semester CHAR(10),
      Grade CHAR(2),
      PRIMARY KEY (StudentID, ClassID),
      FOREIGN KEY (StudentID) REFERENCES Students,
      FOREIGN KEY (ClassID) REFERENCES Class);
```

We want to ensure that if the foreign key StudentID exists in Enrolled, a corresponding entry exists in the Students table for that StudentID. In addition, we want to ensure that if a foreign key ClassID exists in Enrolled, a corresponding entry exists in the Class table for that ClassID.

Let's create a few tables and watch what happens:

MySQL implemented Refential integrity in the InnoDB TYPE – a storage engine done by Innobase Oy.   It also contains ACID-compliant transaction support. This storage engine became the default in 5.5.

```
carrot> mysql -h faure -p -v < create.sql
Enter password:
--------------
DROP TABLE IF EXISTS Class
--------------
--------------
DROP TABLE IF EXISTS Students
--------------
--------------
DROP TABLE IF EXISTS Enrolled
--------------
-------------
CREATE TABLE Class (
	ClassID CHAR(20),
	TimeOffered CHAR(20),
	PRIMARY KEY (ClassID))
	TYPE=InnoDB
--------------
--------------
CREATE TABLE Students (
	StudentID INTEGER,
	Major CHAR(20),
	PRIMARY KEY (StudentID))
	TYPE=InnoDB
--------------
--------------
CREATE TABLE Enrolled (
	StudentID INTEGER,
	ClassID CHAR(20),
	Semester CHAR(10),
	Grade CHAR(2),
	PRIMARY KEY (StudentID, ClassID),
	FOREIGN KEY (StudentID) REFERENCES Students (StudentID),
	FOREIGN KEY (ClassID) REFERENCES Class (ClassID))
	TYPE=InnoDB
--------------

carrot>
```

Looks like we have created our tables correctly (although there is a bug that will be illustrated shortly)

First, let’s create a normal entry in Enrolled.

```
carrot> mysql -h faure -p -v < populate.sql
Enter password:
--------------
INSERT INTO Class VALUES ('CS430', 'T TH 11-12:30')
--------------

--------------
INSERT INTO Class VALUES ('CS314', 'T TH 9:30-10:45')
--------------

--------------
INSERT INTO Students VALUES (1234, 'Computer Science')
--------------

--------------
INSERT INTO Students VALUES (5678, 'Computer Science')
--------------

--------------
INSERT INTO Enrolled VALUES (1234, 'CS314', 'Fall', 'A' )
--------------

--------------
INSERT INTO Enrolled VALUES (1234, 'CS430', 'Spring', 'A' )
--------------

carrot>
```

No problems – because both the StudentID and the ClassID already existed.

Now let’s try one where the Student ID does not exist.

```
carrot> cat populate_error1.sql
USE waker;
INSERT INTO Enrolled VALUES (3456, 'CS430', 'Spring', 'A' );
carrot> mysql -h faure -p -v < populate_error1.sql
Enter password:
--------------
INSERT INTO Enrolled VALUES (3456, 'CS430', 'Spring', 'A' )
--------------

ERROR 1452 (23000) at line 2: Cannot add or update a child row: a foreign key
constraint fails (`waker`.`Enrolled`, CONSTRAINT `Enrolled_ibfk_1` FOREIGN KEY
(`StudentID`) REFERENCES `Students` (`StudentID`))
carrot>
```

And of course the same thing happens if we try to add a record into Enrolled that has a non-existent ClassID

```
carrot> cat populate_error2.sql
USE waker;
INSERT INTO Enrolled VALUES (1234, 'CS530', 'Spring', 'A' );
carrot> mysql -h faure -p -v < populate_error2.sql
Enter password:
--------------
INSERT INTO Enrolled VALUES (1234, 'CS530', 'Spring', 'A' )
--------------

ERROR 1452 (23000) at line 2: Cannot add or update a child row: a foreign key
constraint fails (`waker`.`Enrolled`, CONSTRAINT `Enrolled_ibfk_2` FOREIGN KEY
(`ClassID`) REFERENCES `Class` (`ClassID`))
carrot>
```

What happens now if we want to delete a student, or we want to change their id? If we are to keep referential integrity, we need to have some kind of action… Remember, there are four kinds of action we can take.

- Do not allow the delete to occur (default)
- Cascade the delete
- Set the value to a default
- Set the value to null

Because we did not specify anything, the default should be set (i.e. delete not allowed). Let’s see if that is the case:

```
carrot> mysql -h faure -p -v < delete.sql
Enter password:
--------------
SELECT * from Students, Enrolled
--------------

StudentID       Major   StudentID       ClassID Semester        Grade
1234    Computer Science        1234    CS314   Fall    A
5678    Computer Science        1234    CS314   Fall    A
1234    Computer Science        1234    CS430   Spring  A
5678    Computer Science        1234    CS430   Spring  A
--------------
DELETE from Students where StudentID = 1234
--------------

ERROR 1451 (23000) at line 3: Cannot delete or update a parent row: a foreign key
constraint fails (`waker`.`Enrolled`, CONSTRAINT `Enrolled_ibfk_1` FOREIGN KEY
(`StudentID`) REFERENCES `Students` (`StudentID`))
carrot>
```

Yep – that's what we wanted to happen. But what if what we wanted to happen was one of the other actions – such as CASCADE. CASCADE mean that when we delete a student, we also delete the entries in any table that has StudentID as a foreign key – such as enrolled.

To do this we need to change the definition of the table. Let's modify the create script and recreate the tables to cascade. Remember when I said there was a bug in the original create script? We are about to find it.

Here is the modified script:

```
carrot> cat create.sql
USE waker;
DROP TABLE IF EXISTS Class;
DROP TABLE IF EXISTS Students;
DROP TABLE IF EXISTS Enrolled ;
CREATE TABLE Class (
        ClassID CHAR(20),
        TimeOffered CHAR(20),
        PRIMARY KEY (ClassID))
        TYPE=InnoDB;
CREATE TABLE Students (
        StudentID INTEGER,
        Major CHAR(20),
        PRIMARY KEY (StudentID))
        TYPE=InnoDB;
CREATE TABLE Enrolled (
        StudentID INTEGER,
        ClassID CHAR(20),
        Semester CHAR(10),
        Grade CHAR(2),
        PRIMARY KEY (StudentID, ClassID),
        FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
                ON DELETE CASCADE,
        FOREIGN KEY (ClassID) REFERENCES Class (ClassID)
                ON DELETE CASCADE)
        TYPE=InnoDB;
carrot>
```

When we run it we get:

```
carrot> mysql -h faure -p -v < create.sql
Enter password:
--------------
DROP TABLE IF EXISTS Class
--------------

ERROR 1217 (23000) at line 2: Cannot delete or update a parent row: a foreign key
constraint fails
carrot>
```

Why?

Easily fixed, we delete the table with the foreign keys first.

```
carrot> vi create.sql
carrot> mysql -h faure -p -v < create.sql
Enter password:
--------------
DROP TABLE IF EXISTS Enrolled
--------------

--------------
DROP TABLE IF EXISTS Class
--------------

--------------
DROP TABLE IF EXISTS Students
--------------

--------------
CREATE TABLE Class (
	ClassID CHAR(20),
	TimeOffered CHAR(20),
	PRIMARY KEY (ClassID))
	TYPE=InnoDB
--------------

--------------
CREATE TABLE Students (
	StudentID INTEGER,
	Major CHAR(20),
	PRIMARY KEY (StudentID))
	TYPE=InnoDB
--------------

--------------
CREATE TABLE Enrolled (
	StudentID INTEGER,
	ClassID CHAR(20),
	Semester CHAR(10),
	Grade CHAR(2),
	PRIMARY KEY (StudentID, ClassID),
	FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
		ON DELETE CASCADE,
	FOREIGN KEY (ClassID) REFERENCES Class (ClassID))
	TYPE=InnoDB
--------------
```

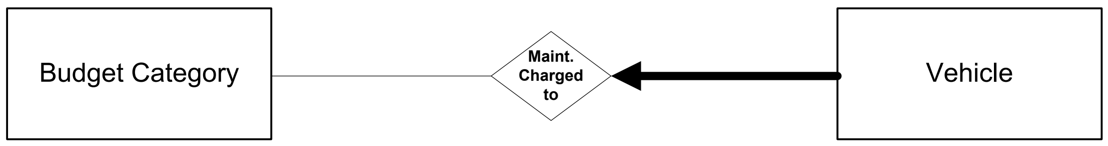
Now let's repopulate and see what happens when we try to delete the student.

```
carrot> mysql -h faure -p -v < delete.sql
Enter password:
--------------
SELECT * from Students
--------------

StudentID	Major
1234	Computer Science
5678	Computer Science
--------------
SELECT * from Enrolled
--------------

StudentID	ClassID	Semester	Grade
1234	CS314	Fall	A
1234	CS430	Spring	A
5678	CS314	Fall	A
5678	CS430	Spring	A
--------------
DELETE from Students where StudentID = 1234
--------------

--------------
SELECT * from Students
--------------

StudentID	Major
5678	Computer Science
--------------
SELECT * from Enrolled
--------------

StudentID	ClassID	Semester	Grade
5678	CS314	Fall	A
5678	CS430	Spring	A
carrot>
```

The other 2 actions that can take place when the referential integrity is challenged is to set the value to a default or to be set to null.

Imagine the following scenario – you are working for the Road and Bridge department of the local county. Every year the county has a working budget that tracks the maintenance costs of each vehicle against a budget category. When we get to the end of the year, we no longer care about what category it was charged to in the yearly budget, we only care that it was charged against the 2010 budget. In fact, a whole new set of budget categories are created for the new year that may or may not look like the old year.

This looks like the following:



In this scenario, you might not want to clog your budget tables with every year's categories, so when you delete the 2010 budget categories, you could create a generic 2010 budget that everything gets changed to.

The InnoDB TYPE does not currently support SET DEFAULT.

The second scenario is to set the value to NULL – which basically means unknown. This is allowed unless the value in the child's CREATE TABLE clause is set to NOT NULL.

Why might you want to do that?

(When you don't care to remember the id of a deleted employee associated with a charge).