Databases and Info Systems

Structured Query Language (SQL)

Dr. Seán Russell
sean.russell@ucd.ie,

School of Computer Science,
University College Dublin

March 10, 2020

Table of Contents

majors

major_id	major_name
1	Internet of Things Engineering
2	Software Engineering
3	Finance
4	Electronic and Information Engineering

modules

module_code	module_name	major_id
COMP2013J	Databases and Information Systems	2
COMP1001J	Introduction to Programming 1	1
EENG2002J	Circuits and Systems	4
EENG2003J	Digital Circuits	4

students

student_id	student_name	major_id
06373313	Sean Russell	2
04123123	David Lillis	1
12453234	Abey Campbel	4

grades

student_id	module_code	grade
06373313	COMP2013J	A+
06373313	COMP1001J	A+
04123123	EENG2002J	B
04123123	COMP1001J	A
12453234	COMP1001J	A
12453234	EENG2002J	C

Inter-Relational Constraints

- Inter-Relational Constraints means that we are talking about constraints that apply between relations
- Previous constraints we have seen (`UNIQUE, NOT NULL, PRIMARY KEY`) have all been **intra-relational** constraints that operate just within one table
- What we are talking about here are constraints on **Foreign Keys**

Foreign Keys

- A foreign key is an attribute (or group of attributes) in one table that is linked to the primary key of another table
  - They can also be linked to the same table
- Foreign keys act as a way to combine tables by representing the relationship between them
  - Foreign keys are usually used in join conditions

Inter-Relational Constraints

- This type of constraint is used to enforce **referential integrity**
  - **Integrity**: The data stored in the database is accurate and consistent
  - **Referential**: The attribute references another attribute
- **Referential Integrity**: When an attribute refers to another attribute, it can only store values that also exist in that other attribute

Referential Integrity Example

- **Referential Integrity**: When an attribute refers to another attribute, it can only store values that also exist in that other attribute
  - An attribute `module_code` in the **grades** table might refer to a `module_code` attribute in a **modules** table
  - Only codes that actually exist in the modules table can be stored in the results table

Defining Inter-Relational Constraints

- SQl allows the definition of referential integrity constraints in two ways:
  - Using `REFERENCES`
  - Using `FOREIGN KEY`
- When we are defining these constraints we can associate "reaction policies" to violations
  - What should happen when the rule is broken

REFERENCES

- The REFERENCES keyword allows us to associate a single attribute as a foreign key of another table
- This is added after the domain of the attribute in the create table statement, the syntax is like this:

```
module_code CHAR(10) REFERENCES modules(module_code),
```

- NOTE: In MySQL this will not be enforced so we will not use this one

FOREIGN KEY

- The FOREIGN KEY syntax allows us to associate a single or multiple attributes as a foreign key of another table
- This is added in the "other constraints" section of the create table statement, the syntax is like this:

```
FOREIGN KEY ( Attribute {, Attribute } ) REFERENCES table(
    Attribute {, Attribute } )
```

```
FOREIGN KEY (module_code) REFERENCES modules(module_code),
```

Example

- Setting an inter-relation constraint based on a value in one table being used as a key in another table
- Here, each student has a `major_id`, which references the `major_id` attribute in the majors table

```
CREATE TABLE students (
  student_id CHAR(8) PRIMARY KEY,
  student_name VARCHAR(30),
  major_id INT,
  FOREIGN KEY(major_id) REFERENCES majors(major_id)
);
```

- Only values for `major_id` that are stored in the majors table can be used

Adding Without Major

Available Values

```
select * from majors;
```

```
+----------+--------------------------------------+
| major_id | major_name                           |
+----------+--------------------------------------+
|        1 | Internet of Things Engineering       |
|        2 | Software Engineering                 |
|        3 | Finance                              |
|        4 | Electronic and Information Engineering |
+----------+--------------------------------------+
```

Inserting A Student

```
INSERT INTO students VALUES ("06373313", "Sean Russell", 5);
```

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
    (`week6`.`students`, CONSTRAINT `students_ibfk_1` FOREIGN KEY (`major_id`)
    REFERENCES `majors` (`major_id`))
```

Changing A Major

Current Students

```
mysql> SELECT * FROM students;
+------------+--------------+----------+
| student_id | student_name | major_id |
+------------+--------------+----------+
| 06373313   | Sean Russell |        2 |
+------------+--------------+----------+
```

Modifying Major

```
mysql> UPDATE majors SET major_id = 5 WHERE major_id = 2;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint
     fails (`week6`.`students`, CONSTRAINT `students_ibfk_1` FOREIGN KEY (`major_id`)
     REFERENCES `majors` (`major_id`))
```

Deleting a Major

Current Students

```
mysql> SELECT * FROM students;
+------------+--------------+----------+
| student_id | student_name | major_id |
+------------+--------------+----------+
| 06373313   | Sean Russell |        2 |
+------------+--------------+----------+
```

Deleting Major

```
mysql> DELETE FROM majors WHERE major_id = 2;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint
     fails (`week6`.`students`, CONSTRAINT `students_ibfk_1` FOREIGN KEY (`major_id`)
     REFERENCES `majors` (`major_id`))
```

Viewing Constraints

- To see what inter-relational constraints are on a table, we cannot use describe

```
DESCRIBE students;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| student_id   | char(8)     | NO   | PRI | NULL    |       |
| student_name | varchar(30) | YES  |     | NULL    |       |
| major_id     | int         | YES  | MUL | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

- Instead, we can use SHOW CREATE TABLE
- This has a more complex output, but is more detailed
- IT gives you the SQL command that can create this table

SHOW CREATE TABLE

```
SHOW CREATE TABLE students;
+----------+--------------------------------------------------------------------------------+
| Table    | Create Table                                                              |
+----------+--------------------------------------------------------------------------------+
| students | CREATE TABLE `students` (
  `student_id` char(8) NOT NULL,
  `student_name` varchar(30) DEFAULT NULL,
  `major_id` int DEFAULT NULL,
  PRIMARY KEY (`student_id`),
  KEY `major_id` (`major_id`),
  CONSTRAINT `students_ibfk_1` FOREIGN KEY (`major_id`) REFERENCES `majors` (`major_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+----------+--------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Table of Contents

Reaction Policies

- In our example, making a change to `major_id` in the majors table may break our rules
    - A student with a major that does not exist
    - The FOREIGN KEY constraint prevents this from happening
    - This can be because of insertions, updates or deletions
- Depending on the database design, we may want to allow this type of change, but for the database to automatically react to make sure that integrity is maintained

Reaction Policies

- There are different reactions that can happen when constraint is broken

  CASCADE The change that was made in the external table should also be made here.

  SET NULL The current value of the field is set to NULL so we are no longer connected to the other table

  SET DEFAULT Whatever the default value is for the attribute is assigned in place of the current value

  NO ACTION or RESTRICT This prevents the change from taking place on the external table

- RESTRICT is the default, when no reaction policy is set

What to React To?

- The reaction we want may depend on what happened to break the rules
  - It might be an UPDATE operation where the key is changed
  - It might be a DELETE operation where the row is removed
- The FOREIGN KEY constraint allows us to specify what we want to happen in each of these circumstances

Updated

- Choosing What should happen when the key is updated
  - ON UPDATE CASCADE
  - ON UPDATE SET NULL
  - ON UPDATE SET DEFAULT
  - ON UPDATE NO ACTION
  - ON UPDATE RESTRICT

Deleted

- Choosing What should happen when the row is deleted
  - ON DELETE CASCADE
  - ON DELETE SET NULL
  - ON DELETE SET DEFAULT
  - ON DELETE NO ACTION
  - ON DELETE RESTRICT

Cascade Example

```
CREATE TABLE modules (
  module_code CHAR(9) PRIMARY KEY,
  module_name VARCHAR(40),
  major_id INT,
  FOREIGN KEY(major_id) REFERENCES majors(major_id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
);
```

- When the major is deleted, then the rows matching this `major_id` are removed from the modules table
- When the value of `major_id` is changed in the majors table, then the value of `major_id` is changed for every module with the matching value

Cascade Example

Set NULL Example

```
CREATE TABLE students (
  student_id CHAR(8) PRIMARY KEY,
  student_name VARCHAR(30),
  major_id INT,
  FOREIGN KEY(major_id) REFERENCES majors(major_id)
  ON DELETE SET NULL
  ON UPDATE SET NULL
);
```

- When the major is deleted, then the `major_id` of each student with that value is set to NULL
- When the value of `major_id` is changed in the majors table, then the value of `major_id` is set to NULL for every student with the matching value

SET NULL EXAMPLE

More Complex Example

```
CREATE TABLE grades(
  student_id CHAR(8),
  module_code CHAR(9),
  grade VARCHAR(2),
  PRIMARY KEY(student_id,module_code),
  FOREIGN KEY(student_id) REFERENCES students(student_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY(module_code) REFERENCES modules(module_code)
    ON UPDATE CASCADE ON DELETE RESTRICT
);
```

- What are the effects here?

Effects

1. If we try to delete a student from the students table, but they have grades in the grades table it will be blocked (ON DELETE RESTRICT)
2. If we change a students id in the students table, the matching student id will be changed in the grades table (ON UPDATE CASCADE)
3. If we try to delete a module from the modules table, but there are grades for that module in the grades table it will be blocked (ON DELETE RESTRICT)
4. If we change a modules' code in the modules table, then the same module code in the grades table will be change to match (ON UPDATE CASCADE)

Testing 1

- A student with grades

```
DELETE FROM students WHERE student_id = "06373313";
```

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key
    constraint fails (`week6`.`grades`, CONSTRAINT `grades_ibfk_1` FOREIGN KEY
    (`student_id`) REFERENCES `students` (`student_id`) ON DELETE RESTRICT ON
    UPDATE CASCADE)
```

- A student with no grades

```
DELETE FROM students WHERE student_id = "14232232";
```

```
Query OK, 1 row affected (0.01 sec)
```

Testing 2

```
SELECT * from grades where module_code = "COMP1001J";
```

```
+------------+-------------+-------+
| student_id | module_code | grade |
+------------+-------------+-------+
| 04123123   | COMP1001J   | A     |
| 06373313   | COMP1001J   | A+    |
| 12453234   | COMP1001J   | A     |
+------------+-------------+-------+
3 rows in set (0.00 sec)
```

```
UPDATE students SET student_id = "16373313" WHERE student_id = "06373313";
```

```
SELECT * from grades where module_code = "COMP1001J";
```

```
+------------+-------------+-------+
| student_id | module_code | grade |
+------------+-------------+-------+
| 04123123   | COMP1001J   | A     |
| 12453234   | COMP1001J   | A     |
| 16373313   | COMP1001J   | A+    |
+------------+-------------+-------+
3 rows in set (0.00 sec)
```

Testing 3

- A module with grades

```
DELETE FROM modules where module_code = "COMP2013J";
```

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key
    constraint fails (`week6`.`grades`, CONSTRAINT `grades_ibfk_2` FOREIGN KEY
    (`module_code`) REFERENCES `modules` (`module_code`) ON DELETE RESTRICT ON
    UPDATE CASCADE)
```

- A module with no grades

```
DELETE FROM modules WHERE module_code = "EENG2003J";
```

```
Query OK, 1 row affected (0.01 sec)
```

Testing 4

```
SELECT * from grades WHERE grade = "A";
```

```
+------------+-------------+-------+
| student_id | module_code | grade |
+------------+-------------+-------+
| 04123123   | COMP1001J   | A     |
| 12453234   | COMP1001J   | A     |
+------------+-------------+-------+
2 rows in set (0.00 sec)
```

```
UPDATE modules SET module_code = "COMP1004J" WHERE module_code =
    "COMP1001J";
```

```
SELECT * from grades WHERE grade = "A";
```

```
+------------+-------------+-------+
| student_id | module_code | grade |
+------------+-------------+-------+
| 04123123   | COMP1004J   | A     |
| 12453234   | COMP1004J   | A     |
+------------+-------------+-------+
2 rows in set (0.00 sec)
```

Table of Contents

employees

emp_id	name	title	salary	dept_id	join_date
1234	Sean Russell	Trainer	50000	10	2018-03-01
4567	Jamie Heaslip	Manager	47000	10	2004-10-21
6542	Leo Cullen	Trainer	45000	10	2012-12-01
1238	Brendan Macken	Technician	25000	20	2001-09-10
1555	Sean O'Brien	Designer	50000	20	1999-06-24
1899	Brian O'Driscoll	Manager	45000	20	1998-02-27
2525	Peter Stringer	Designer	25000	30	2017-01-16
1585	Denis Hickey	Architect	20000	30	2009-08-07
1345	Ronan O'Gara	Manager	29000	30	2019-12-25

departments

dept_id	dept_name	office	division	manager_id
10	Training	Lansdowne	D1	4567
20	Design	Belfield	D2	1899
30	Implementation	Donnybrook	D1	1345
40	Strategy	Terenure	D2	NULL

Nested Queries

- A **nested query** is an SQL query that is contained within another query
    - This is sometimes called a **subquery**
- Usually used in the WHERE clause
- Subqueries can return:
    - A single value (called a **scalar**)
    - A single column
    - A single row
    - A table (multiple columns/rows)

- When a subquery return a single value (scalar), it can be used in the same way as an ordinary single value
- Find the name of the employee(s) who earn the most money

```
SELECT name, salary FROM employees WHERE salary = (SELECT MAX(salary)
    FROM employees);
```

```
+--------------+--------+
| name         | salary |
+--------------+--------+
| Sean Russell |  50000 |
| Sean O'Brien |  50000 |
+--------------+--------+
2 rows in set (0.01 sec)
```

- A nested query returning a single column can be thought of as a set of values.
- We can compare attributes with values from this set to see if its equal, greater than, less than, etc.
  - Using $= > < >= <= <> != $
- Two other keywords are important:

  ANY returns true if the comparison is true for any value in the set

  ALL returns true if the comparison is true for all the values in the set

Example 1

- Find the names of employees who work in departments in Division 'D1

```
SELECT name FROM employees WHERE dept_id = ANY(SELECT dept_id FROM
    departments WHERE division='D1');
```

  - The subquery finds a list of all the department numbers (`dept_id`) for departments that are in division D1
  - When selecting from the employees table, it will match any employee whose `dept_id` is in the set returned by the subquery
- We could also have solved this with a join

Example 2

- Find the employees of department number 10 who have the same first name as a member of department 20
- We can use the SUBSTRING_INDEX function from https://dev.mysql.com/doc/refman/8.0/en/string-functions.html
- SUBSTRING_INDEX(name,' ',1) returns a substring of name, starting at the beginning and ending just before the first space character

```
SELECT name FROM employees WHERE dept_id=10 AND
    SUBSTRING_INDEX(name,' ',1)=ANY(SELECT SUBSTRING_INDEX(name,'
    ',1) FROM employees WHERE dept_id=20 );
```

Example 3

- Find the name of the Department in which there is no employee named Sean

```
SELECT dept_name FROM departments WHERE dept_id != ALL(SELECT
    DISTINCT dept_id FROM employees WHERE name LIKE 'Sean %');
```

  - The subquery finds the `dept_id` for departments who do have an employee named Sean.
  - We want to find departments whose numbers are different to all of these
  - Note that we change from **ANY** to **ALL** because this is a negative match

Example 4

- Find the name of all employees who earn more money that everybody in the Implementation department

```
SELECT name, salary FROM employees WHERE salary > ALL(SELECT salary
      FROM employees JOIN departments USING(dept_id) WHERE
      dept_name='Implementation');
```

- Remember the Closure Property: the output of an operation is a relation
- The output of a SELECT query is a relation, so another query can be run on it
- In SQL, this is OK if you use an alias for the (temporary) table
- A silly example:

```
SELECT name FROM (SELECT * FROM employees WHERE job='Manager') AS
    managers;
```

- Here, the managers table doesn't actually exist, but can be used in a more complex SELECT query just like any real table

Table of Contents

- SQL provides the ability to use **views** in your schema
- A view is a **virtual table** based on a query
- It looks just like a normal table, but it is not stored directly in the database
- We can allow users to see a subset of one or more tables, without giving them access to the table itself

- Every view has a name and a SELECT query that defines it
- Example:

```
CREATE VIEW managers AS SELECT * FROM employees WHERE emp_id =
    ANY(SELECT manager_id FROM departments);
```

- After this, managers looks just like an ordinary table.
- Changes in the employees table will automatically be seen in the manager table
- To delete a view: `DROP VIEW managers;`

Changing Data

- For some **simple** views, you can update and delete just like a normal table, and these changes will be made to the underlying tables
- Full details: (https://dev.mysql.com/doc/refman/8.0/en/view-updatability.html)
- Some examples of When you can't
  - If it contains aggregate functions
  - If it uses DISTINCT
  - If it uses GROUP BY
  - Subquery in SELECT list (somtimes)
  - Certain joins
  - If it refers to a non-updatable biew in FROM

Style Guide

- As a database designer and programmer, SQL gives you a lot of freedom about how you choose names for tables/databases/variables and how you choose to capitalise.
- It's a good idea to follow a consistent style, which makes your queries more understandable and is more professional overall.
- A good style guide is by Simon Holywell at https://www.sqlstyle.guide (linked on Moodle).