THE UNIVERSITY OF SYDNEY

School of Information Technologies

COMP9120 Database Management Systems **Semester 2, 2016**

Tutorial Week 5: Assignment 1 & Integrity Constraints

Exercise 1. Assignment 1

This week's tutorial is an in-class opportunity to work on Assignment 1 with your group.

During the tutorial, your tutor will also come around to your group for 6 to 8 minutes, and ask you how your E-R model addresses the question shown below. While your tutor ***will not give you answers or solutions for the assignment***, he may prompt you to think about whether your conceptual model can convey certain information in the functional requirements.

You may also ask general questions about E-R diagrams, but remember that your tutor needs to attend to all groups in the class, ***hence please keep to the 8-minute limit!***

The assignment description mentions that "the company must be able to justify the total cost of each vehicle hire".

Question: *How can your model be used to justify the total cost of each vehicle hire?*

Drilling down into this question:

What entity types and relationship types have you created to address this requirement, and how do they capture the information for justifying the total cost of a vehicle hire? If the company was asked for an itemised list of costs that lead to the total cost of a vehicle hire, would your model be able to address this question?

The exercises below are formulated such that they can be carried out in your own time and are Important in giving you practice with the Integrity Constraint features of Relational Database Management Systems.

Exercise 2. Data Integrity Constraints

Download the University schema (available from the 'Schemas' page in Resources section on elearning). Use SQL Developer to connect to your Oracle 12c account and run the schema file to create all the relations and populate them with data. You might need to refresh your client after running the script to see the created tables (eg: right click Tables node and click on Refresh).

a) Key Constraints

The schema already contains some appropriate key and NOT NULL constraints**.** All key constraints are named so that you can drop them easily. You can add more constraints with the following syntax:

```
ALTER TABLE table ADD CONSTRAINT name ...
```

Here `table` is the name of the table you wish to modify, and `name` is the name for the constraint. Supplying a name for the constraint is recommended in order to make it easier to drop them later, as well as making it easier to trace back to which constraint caused a violation.

One example of a business rule we can enforce using a key constraint is that:

“Two units-of-study cannot be taught in the same room at the same time in a given semester;”

```
ALTER TABLE Lecture ADD CONSTRAINT ClassroomConflict
   UNIQUE(classroomId,semester,year,classTime);
```

After adding this uniqueness constraint, check what happens if you try to add a row in the lecture table that violates this constraint.

b) Referential Integrity

The schema already contains some appropriate foreign key constraints**.** To *change* a constraint you must drop it before adding a replacement one (of the same name):

```
    ALTER TABLE table DROP CONSTRAINT name;
```

Looking through the current schema, you will notice:

1. WhenOffered is missing a foreign key constraint on uosCode which references the UnitOfStudy table. Go ahead and add this using:

```
ALTER TABLE WhenOffered ADD CONSTRAINT
WhenOffered_fk_uos FOREIGN KEY (uosCode) REFERENCES
UnitOfStudy ON DELETE CASCADE;
```

You will notice that there is an ON DELETE CASCADE option specified on the foreign key constraint. This means that if a row with a uosCode existing in WhenOffered is deleted from UnitOfStudy then rows in WhenOffered having this uosCode are also deleted. Try deleting a row from UnitOfStudy for a unit of study referenced in WhenOffered.

2. Consider whether the default of ‘`ON DELETE NO ACTION`’ is appropriate for foreign key constraints in the current schema. For instance, if a unit of study is deleted, what should happen to corresponding records in the `Required` table?

The Required table shouldn’t have references to units of study that no longer exist. We can modify the existing ON DELETE option as follows:

```
ALTER TABLE Requires DROP CONSTRAINT
Requires_uoSCode_fk_UOS;

ALTER TABLE Requires ADD CONSTRAINT
Requires_uoSCode_fk_UOS FOREIGN KEY (uoSCode)
REFERENCES UnitOfStudy(uoSCode) ON DELETE CASCADE;
```

c) Domain Constraints

You can add check constraints with the following syntax:

```
ALTER TABLE table ADD CONSTRAINT name CHECK ( condition )
```

Here `condition` is the condition that should hold true for each record in the table.

Add domain constraints for the following:

1. The system shall contain the transcript records of each student that show each unit of study the student has completed, the semester the student took the course, and the grade the student received (all grades are in the set {'F','P','CR','D','HD','W'});

```
ALTER TABLE Transcript ADD CONSTRAINT valid_grades
CHECK(grade IN ('F','P','CR','D','HD','W'));
```

What happens if you try and modify a grade to 'M'?

2. The system shall contain information about the unit-of-studies offered, and for each unit of study the system shall contain whether the unit is offered in semester S1, or in semester S2, on in winter semester (WS) or summer semester (SS);

```
ALTER TABLE WhenOffered ADD CONSTRAINT
valid_semesters CHECK(semester IN
('S1','S2','WS','SS'));
```

What happens if you try to modify a semester to 'S3'?

d) Assertions

Assertions are not implemented in most mainstream RDBMS', but you should be able to express them with the standard syntax. Below are some examples for how to write assertions to enforce some business rules:

1. The number of students enrolled in a unit-of-study must equal the current enrolment;

```
CREATE ASSERTION EnrollmentAssert CHECK (
   NOT EXISTS (SELECT 1
   FROM UoSOffering o
   WHERE enrollment != (SELECT COUNT(*)
   FROM Transcript t
   WHERE t.uosCode=o.uosCode AND
   t.semester=o.semester AND
   t.year=o.year)
   )
);
```

Try executing the underlined code followed by a semicolon. Is this assertion be violated?

2. The room assigned to a unit-of-study must have at least as many seats as the maximum allowed enrolment for the unit;

```
CREATE ASSERTION RoomCapacityAssert CHECK (
    NOT EXISTS ( SELECT 1
    FROM (UoSOffering NATURAL JOIN Lecture)
    NATURAL JOIN ClassRoom
    WHERE seats < maxEnrollment )
);
```

Try executing the underlined code followed by a semicolon. Is this assertion be violated?

3. A student cannot be registered for more than 24 credit points in a given semester;

```
CREATE ASSERTION MaxCreditsAssert CHECK (
    NOT EXISTS ( SELECT 1
    FROM Transcript NATURAL JOIN UnitOfStudy
    GROUP BY studId, semester, year
    HAVING SUM(credits) > 24 )
);
```

The group by clause will be covered in a later lecture, however, try executing the underlined code followed by a semicolon. Is this assertion be violated?

e) Triggers

Some of the behaviour intended from assertions (which are not implemented by most mainstream RDBMS') can be achieved through triggers.

1. We can write a trigger to update the enrolment number for a unit of study offering *when a student is added* (in Transcript), as shown below:

```
CREATE TRIGGER UpdateEnrol
AFTER INSERT ON Transcript
FOR EACH ROW
BEGIN
    UPDATE UoSOffering U
    SET enrollment=enrollment+1
  WHERE U.uosCode=:NEW.uosCode AND
    U.semester=:NEW.semester AND
    U.year=:NEW.year;
END;
```

2. This trigger will not guarantee that the first assertion will always hold true. What other triggers are needed?

Consider:

- What happens if enrolment exceeds maxenrolment? (should update be attempted before insert?)

- What happens if Transcript is later updated to change the course enrolled?
- What happens if Transcript entry is deleted?
- What happens if UoSOffering.maxenrolment is updated?

Triggers should be added to deal with such situations.