Instructor Teaching Tracker

Brandon R. Russell

University of Maryland University College

Contents

[Statement of Work 3](#_Toc435897074)

[Entity Relationship Diagram 4](#_Toc435897075)

[Table Design 5](#_Toc435897076)

[Data Definition 6](#_Toc435897077)

[Insert Data 13](#_Toc435897078)

[Queries 18](#_Toc435897079)

[Conclusion 27](#_Toc435897080)

# Statement of Work

1. This database supports the Instructor Teaching Tracker application. It keeps track of Instructor, course, and section data, as well as classes taught and schedule. The purpose of this is to assist Instructors with scheduling/maintaining classes in an efficient manner.

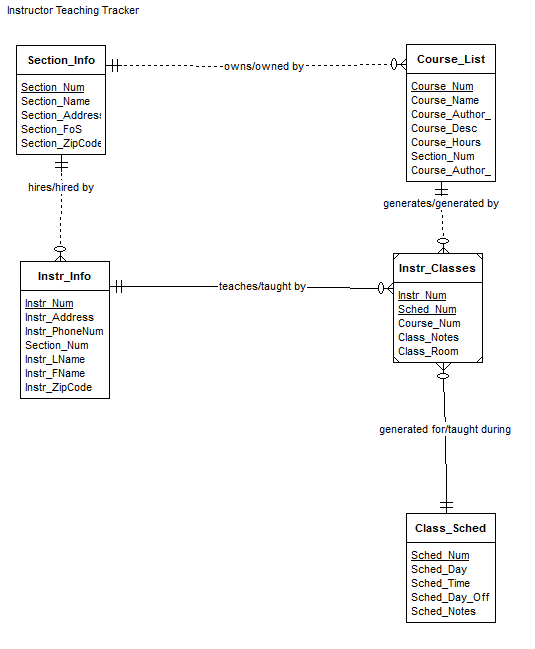
The Instructor table has the name, address, phone number and section number. The course list has the course name, author, description, hours and section number. The section table has the section number, name, address and field of study. The Instructor Classes table shows the Instructor, schedule, course, room number and Instructor notes. The class schedule gives the day/time a class is taught, a down day schedule and notes.

For this application a section can have many Instructors, but an Instructor can only belong to one section. A section can control many courses, but a course can belong to only one section. An instructor can teach many classes and a class can be taught by many instructors; this is handled with an associative entity. Many classes can be generated from one course, but each class can only be generated from a single course. One class can have only one schedule, while many classes can have the same schedule.

Assumptions: This application tracks classes taught by instructors and includes a class schedule. It is assumed classes will always meet a specific schedule criteria of Monday-Wednesday-Friday, Tuesday-Thursday, or Saturday. Down days are tracked for individual classes since this will vary based upon the specific class.

1. Database: Oracle 11g on Windows platform.
2. Software: ER Assistant, Oracle SQL developer, and Oracle SQL developer data modeler.
3. DDL – drop and create the database objects including tables, views, triggers, and sequences. DML – populate and query data.

# Entity Relationship Diagram



# Table Design

|  |  |  |
| --- | --- | --- |
| Section\_Info | | |
| Section\_Num | INTEGER | Not null - pk |
| Section\_Name | VARCHAR(100) |  |
| Section\_Address | VARCHAR(100) |  |
| Section\_FoS | VARCHAR(100) |  |
| Section\_ZipCode | VARCHAR(10) |  |

|  |  |  |
| --- | --- | --- |
| Course\_List | | |
| Course\_Num | INTEGER | Not null – pk |
| Course\_Name | VARCHAR(100) |  |
| Course\_Author\_FName | VARCHAR(20) |  |
| Course\_Author\_LName | VARCHAR(20) |  |
| Course\_Desc | VARCHAR(1000) |  |
| Course\_Hours | INTEGER |  |
| Section\_Num | INTEGER | Not null - fk |

|  |  |  |
| --- | --- | --- |
| Instr\_List | | |
| Instr\_ID | INTEGER | Not null – pk |
| Instr\_Address | VARCHAR(100) |  |
| Instr\_PhoneNum | VARCHAR(20) |  |
| Section\_Num | INTEGER | Not null - fk |
| Instr\_LName | VARCHAR(20) |  |
| Instr\_FName | VARCHAR(20) |  |
| Instr\_ZipCode | VARCHAR(10) |  |

|  |  |  |
| --- | --- | --- |
| Class\_Sched | | |
| Sched\_Num | INTEGER | Not null - pk |
| Sched\_Day | VARCHAR(100) |  |
| Sched\_Time | VARCHAR(10) |  |
| Sched\_Day\_Off | VARCHAR(100) |  |
| Sched\_Notes | VARCHAR(1000) |  |

|  |  |  |
| --- | --- | --- |
| Instr\_Classes | | |
| Instr\_ID | INTEGER | Not null – pk/fk |
| Sched\_Num | INTEGER | Not null – pk/fk |
| Course\_Num | INTEGER | Not null - fk |
| Class\_Notes | VARCHAR(1000) |  |
| Class\_Room | VARCHAR(5) |  |

# Data Definition

SET echo on

SET serveroutput on

**/\* Drop tables, sequence, and other objects you create\*/**

DROP TABLE Instr\_Classes;

Table INSTR\_CLASSES dropped.

DROP TABLE Instr\_Info;

Table INSTR\_INFO dropped.

DROP TABLE Course\_List;

Table COURSE\_LIST dropped.

DROP TABLE Section\_Info;

Table SECTION\_INFO dropped.

DROP TABLE Class\_Sched;

Table CLASS\_SCHED dropped.

DROP SEQUENCE SectionNum\_Seq;

Sequence SECTIONNUM\_SEQ dropped.

DROP SEQUENCE InstrNum\_Seq;

Sequence INSTRNUM\_SEQ dropped.

DROP SEQUENCE CourseNum\_Seq;

Sequence COURSENUM\_SEQ dropped.

DROP SEQUENCE SchedNum\_Seq;

Sequence SCHEDNUM\_SEQ dropped.

**/\* Create 5 tables \*/**

CREATE TABLE Section\_Info

(

Section\_Num INTEGER NOT NULL,

Section\_Name VARCHAR (100),

Section\_Address VARCHAR (100),

Section\_FoS VARCHAR (100),

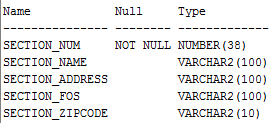
Section\_ZipCode VARCHAR(10),

CONSTRAINT pk\_section PRIMARY KEY (Section\_Num)

);

Table SECTION\_INFO created.

DESCRIBE Section\_Info;



CREATE TABLE Instr\_Info

(

Instr\_Num INTEGER NOT NULL,

Instr\_Address VARCHAR (100),

Instr\_PhoneNum VARCHAR (20),

Section\_Num INTEGER NOT NULL,

Instr\_LName VARCHAR (20),

Instr\_FName VARCHAR (20),

Instr\_ZipCode VARCHAR (10),

CONSTRAINT pk\_Instr PRIMARY KEY (Instr\_Num),

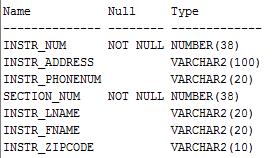
CONSTRAINT fk\_Instr\_Section FOREIGN KEY (Section\_Num)

REFERENCES Section\_Info

);

Table INSTR\_INFO created.

DESCRIBE Instr\_Info;



CREATE TABLE Course\_List

(

Course\_Num INTEGER NOT NULL,

Course\_Name VARCHAR (100),

Course\_Author\_FName VARCHAR (20),

Course\_Author\_LName VARCHAR(20),

Course\_Desc VARCHAR (1000),

Course\_Hours INTEGER,

Section\_Num INTEGER NOT NULL,

CONSTRAINT pk\_Course PRIMARY KEY (Course\_Num),

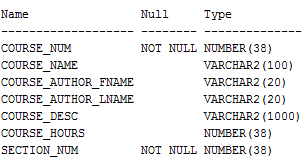
CONSTRAINT fk\_Course\_Section FOREIGN KEY (Section\_Num)

REFERENCES Section\_Info

);

Table COURSE\_LIST created.

DESCRIBE Course\_List;



CREATE TABLE Class\_Sched

(

Sched\_Num INTEGER NOT NULL,

Sched\_Day VARCHAR (100),

Sched\_Time VARCHAR (10),

Sched\_Day\_Off VARCHAR (100),

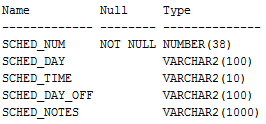
Sched\_Notes VARCHAR (1000),

CONSTRAINT pk\_Schedule PRIMARY KEY (Sched\_Num)

);

Table CLASS\_SCHED created.

DESCRIBE Class\_Sched;



CREATE TABLE Instr\_Classes

(

Instr\_Num INTEGER NOT NULL,

Sched\_Num INTEGER NOT NULL,

Course\_Num INTEGER NOT NULL,

Class\_Notes VARCHAR (1000),

Class\_Room VARCHAR (5),

CONSTRAINT pk\_Class PRIMARY KEY (Instr\_Num, Sched\_Num),

CONSTRAINT fk\_Instr FOREIGN KEY (Instr\_Num)

REFERENCES Instr\_Info

ON DELETE CASCADE,

CONSTRAINT fk\_Sched FOREIGN KEY (Sched\_Num)

REFERENCES Class\_Sched

ON DELETE CASCADE,

CONSTRAINT fk\_Course FOREIGN KEY (Course\_Num)

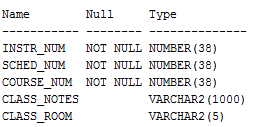
REFERENCES Course\_List

ON DELETE CASCADE

);

Table INSTR\_CLASSES created.

DESCRIBE Instr\_Classes;



**/\* Create indexes on foreign keys\*/**

CREATE INDEX fk\_Course\_List on Course\_List(Section\_Num);

Index FK\_COURSE\_LIST created.

CREATE INDEX fk\_Instr on Instr\_Info(Section\_Num);

Index FK\_INSTR created.

CREATE INDEX fk\_InstrClass on Instr\_Classes(Instr\_Num);

Index FK\_INSTRCLASS created.

CREATE INDEX fk\_ClassSched on Instr\_Classes(Sched\_Num);

Index FK\_CLASSSCHED created.

CREATE INDEX fk\_CourseClass on Instr\_Classes(Course\_Num);

Index FK\_COURSECLASS created.

**/\* Create 2 views \*/**

**/\*This view shows all classes Instructor 1 is teaching on Schedule 1 \*/**

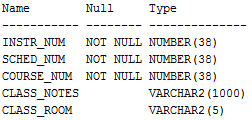
CREATE OR REPLACE VIEW Instr\_Teaching\_View AS

SELECT \* from Instr\_Classes

WHERE Instr\_Num ='1' AND Sched\_Num ='1';

View INSTR\_TEACHING\_VIEW created.

DESCRIBE Instr\_Teaching\_View;



**/\*This view will show which section each Instructor belongs to\*/**

CREATE OR REPLACE VIEW Section\_Instructors\_View AS

SELECT section\_num, s.section\_name, i.instr\_fname, i.instr\_lname

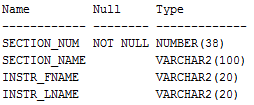
FROM section\_info s

INNER JOIN instr\_info i

USING (section\_num);

View SECTION\_INSTRUCTORS\_VIEW created.

DESCRIBE Section\_Instructors\_View;



**/\* Create trigger \*/**

**/\*This trigger will display a message when a row is added to Instr\_Classes\*/**

CREATE OR REPLACE TRIGGER SchedClass\_Trigger AFTER INSERT ON Instr\_Classes

FOR EACH ROW

BEGIN

dbms\_output.put\_line ('A class has been added!!');

END;

/

Trigger SCHEDCLASS\_TRIGGER compiled

**/\* Create sequence\*/**

CREATE SEQUENCE SectionNum\_Seq

START WITH 1

INCREMENT BY 1;

Sequence SECTIONNUM\_SEQ created.

CREATE SEQUENCE InstrNum\_Seq

START WITH 1

INCREMENT BY 1;

Sequence INSTRNUM\_SEQ created.

CREATE SEQUENCE CourseNum\_Seq

START WITH 1

INCREMENT BY 1;

Sequence COURSENUM\_SEQ created.

CREATE SEQUENCE SchedNum\_Seq

START WITH 1

INCREMENT BY 1;

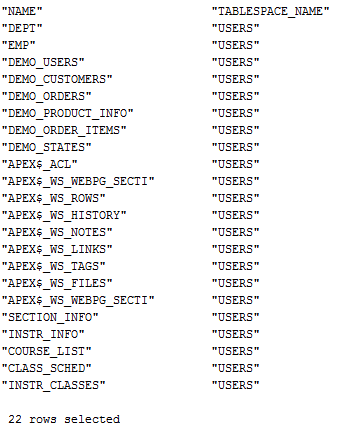
Sequence SCHEDNUM\_SEQ created.

**/\* Data Dictionary query \*/**

PURGE recyclebin;

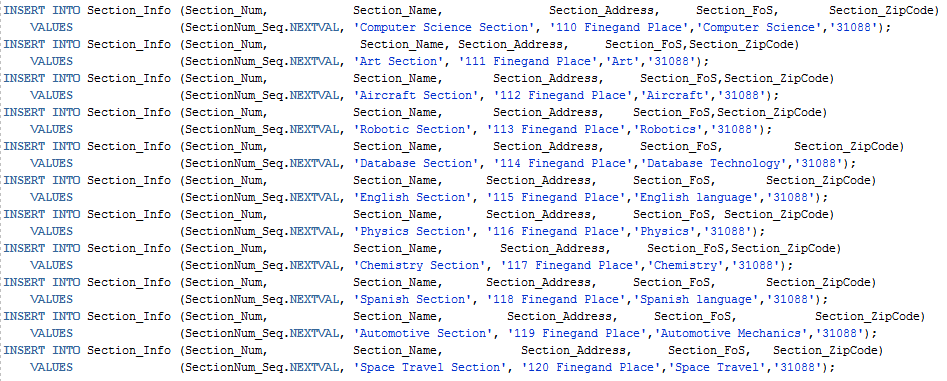
**/\*The next query returns the first 20 characters of all table names and then their tablespace names\*/**

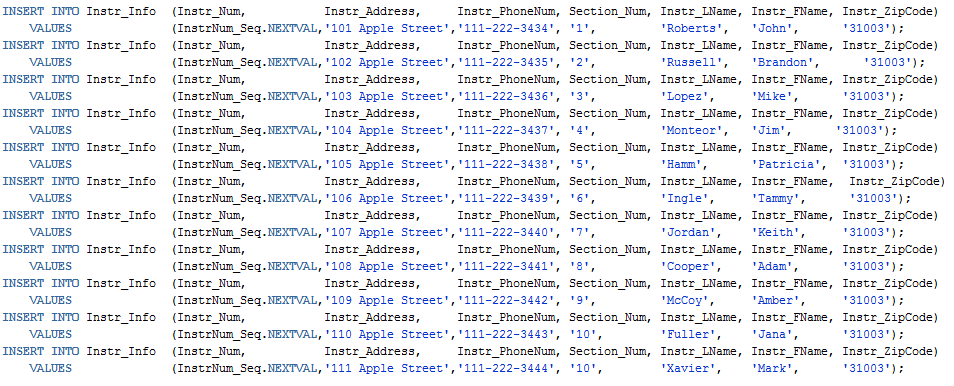
SELECT /\*fixed\*/ substr(table\_name, 1,20) as name, tablespace\_name from user\_tables;

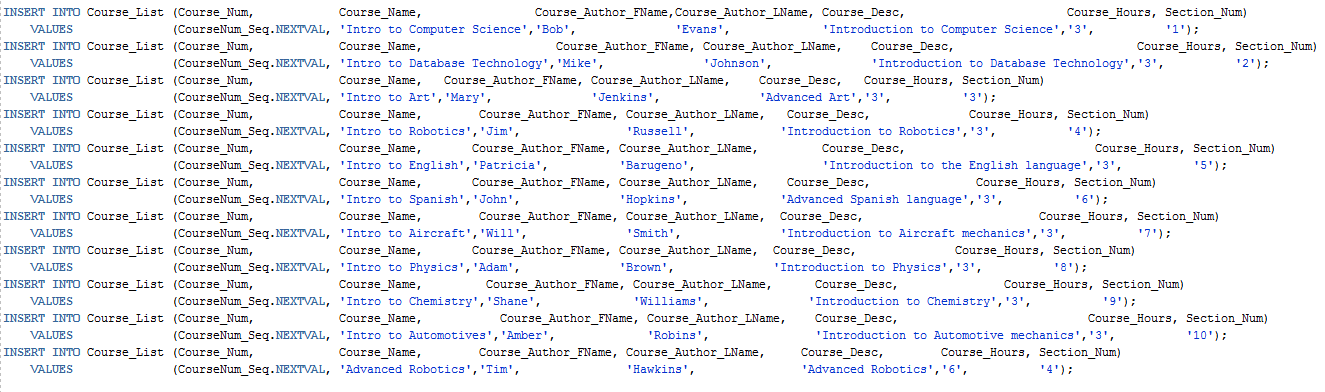


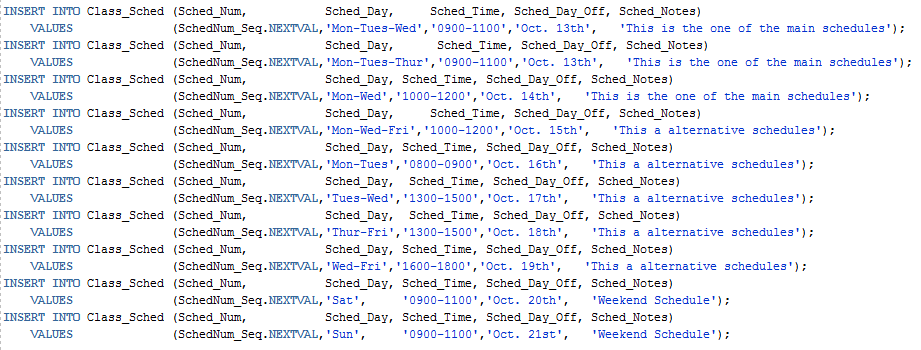
# Insert Data

**/\* Insert 10 or more rows into each table \*/**









INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('1', '1', '1', 'Computer one is broke in this room','A');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('2', '2', '2', 'Room Ready','B');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('3', '3', '3', 'The Instructor Chair is broke','C');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('4', '4', '4', 'A/C is not working','D');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('5', '5', '5', 'Computer two is broke in this room','E');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('6', '6', '6', 'No whiteboard in this room','F');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('7', '7', '7', 'Student station 3 is missing a keyboard','G');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('8', '8', '8', 'Room Ready','H');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('9', '9', '9', 'Room Ready','I');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('10', '10', '10', 'Room Ready','J');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('10', '9', '10', 'Room Ready','J');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('9', '8', '10', 'Room Ready','X');

1 row inserted.

A class has been added!!

INSERT INTO Instr\_Classes (Instr\_Num, Sched\_Num, Course\_Num, Class\_Notes, Class\_Room)

VALUES ('1', '2', '9', 'Room Ready','Z');

1 row inserted.

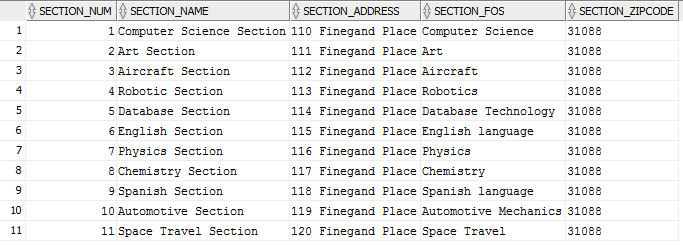
A class has been added!!

commit;

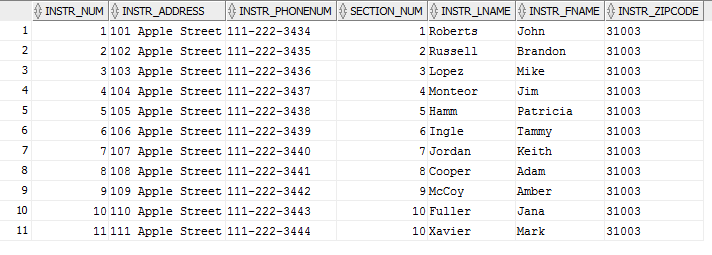
Commit complete.

**/\* Verify that each table has 10 or more rows of data \*/**

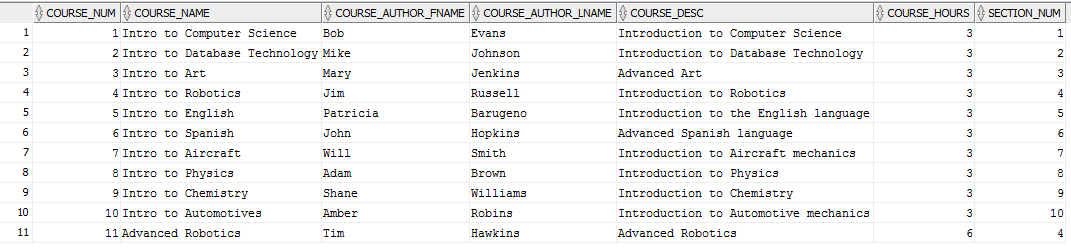
SELECT /\*fixed\*/ \* FROM Section\_Info;



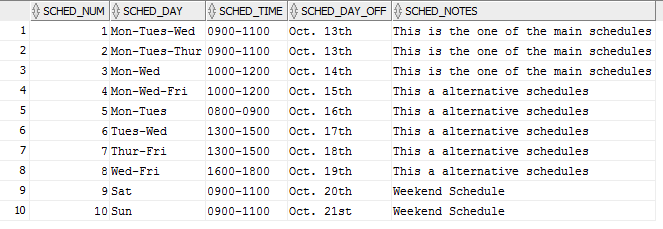
SELECT /\*fixed\*/ \* FROM Instr\_Info;



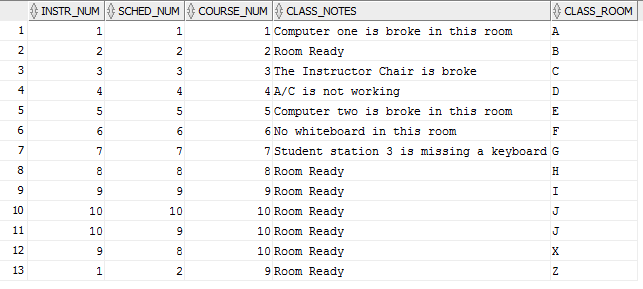
SELECT /\*fixed\*/ \* FROM Course\_List;



SELECT /\*fixed\*/ \* FROM Class\_Sched;



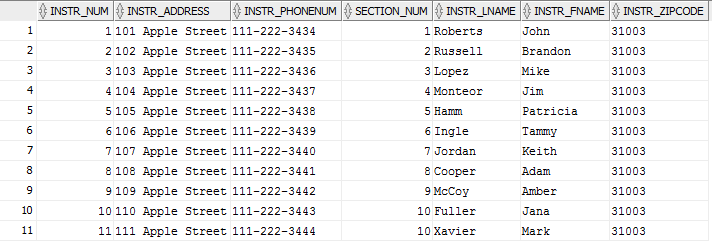
SELECT /\*fixed\*/ \* FROM Instr\_Classes;



# Queries

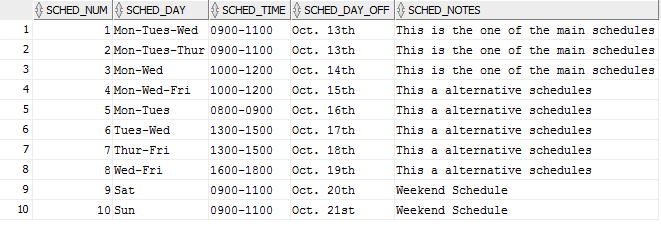
**/\*1. Select all columns and all rows from one table.\*/**

SELECT /\*fixed\*/ \* FROM Instr\_Info;



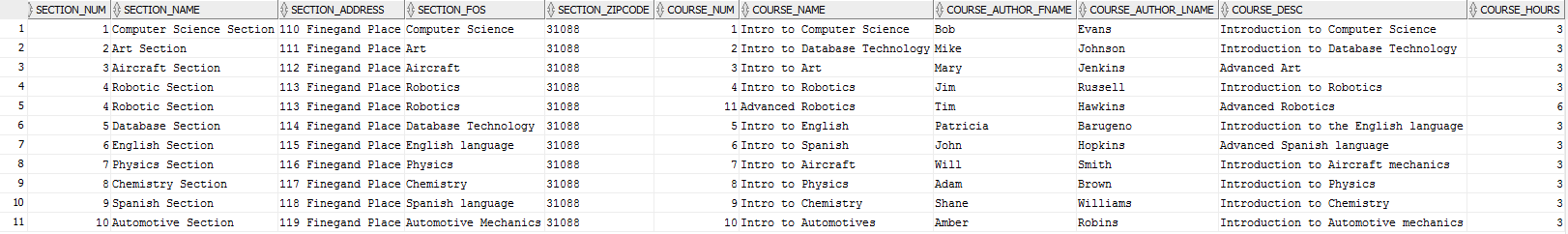
**/\*2. Select 5 columns and all rows from one table. \*/**

SELECT /\*fixed\*/ Sched\_Num, Sched\_Day, Sched\_Time, Sched\_Day\_Off, Sched\_Notes FROM Class\_Sched;



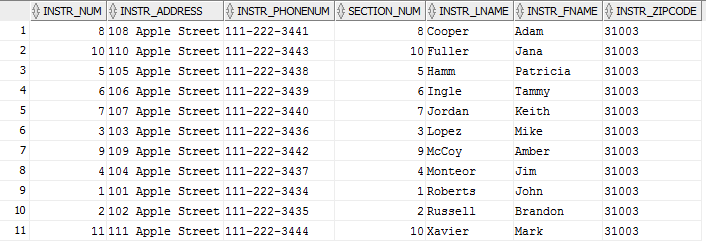
**/\*3. Select all columns and all rows from 2 tables (need a join).\*/**

SELECT /\*fixed\*/ \* FROM Section\_Info INNER JOIN Course\_List USING (Section\_Num);



**/\*4. Select and Order data retrieved from one table \*/**

SELECT /\*fixed\*/ \* FROM Instr\_Info ORDER BY Instr\_LName;



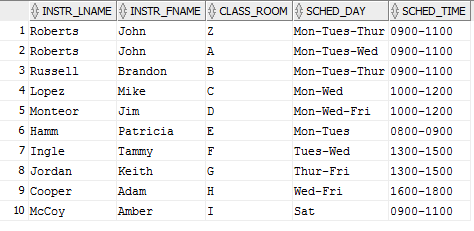
**/\*5. Select 5 columns and 10 rows from 3 tables (need joins).\*/**

SELECT /\*fixed\*/ a.Instr\_LName, a.Instr\_FName, b.Class\_Room, c.Sched\_Day, c.Sched\_Time

FROM (Instr\_Info a INNER JOIN Instr\_Classes b ON a.Instr\_Num=b.Instr\_Num)

INNER JOIN Class\_Sched c ON b.Sched\_Num=c.Sched\_Num

WHERE ROWNUM < 11;

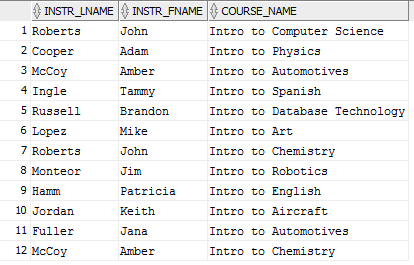


**/\*6. Select distinct rows using joins using 3 tables.\*/**

SELECT DISTINCT /\*fixed\*/ a.Instr\_LName, a.Instr\_FName, c.Course\_Name

FROM (Instr\_Info a INNER JOIN Instr\_Classes b ON a.Instr\_Num=b.Instr\_Num)

INNER JOIN Course\_List c ON b.Course\_Num=c.Course\_Num;

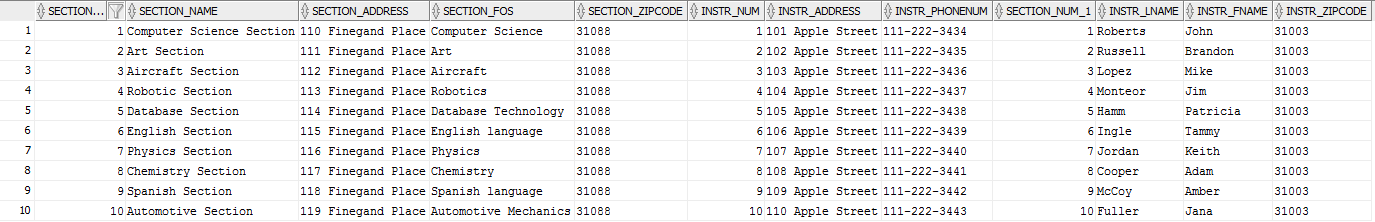


**/\*7. Select all columns and 10 rows from 2 tables (need a join).\*/**

SELECT /\*fixed\*/ \* FROM Section\_Info INNER JOIN Instr\_Info ON

Section\_Info.Section\_Num=Instr\_Info.Section\_Num

WHERE ROWNUM < 11;



**/\*8. Use group by & having in a select statement using one or more table(s).\*/**

**/\*Return a list of sections with 2 active classes being taught.\*/**

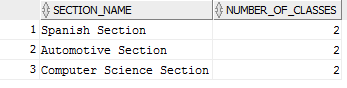
SELECT /\*fixed\*/ Section\_Name, COUNT(Instr\_Classes.Instr\_Num) AS Number\_Of\_Classes FROM (Section\_Info INNER JOIN Instr\_Info ON

Section\_Info.Section\_Num=Instr\_Info.Section\_Num) INNER JOIN Instr\_Classes ON

Instr\_Info.Instr\_Num=Instr\_Classes.Instr\_Num

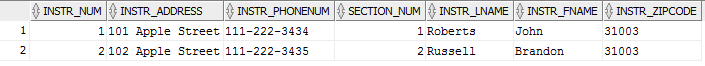
GROUP BY Section\_Name

HAVING COUNT(Instr\_Classes.Instr\_Num) = 2;



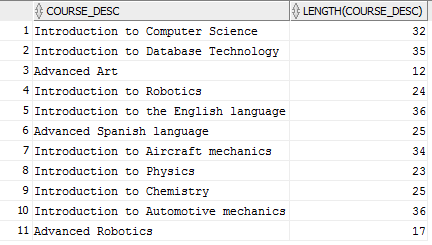
**/\*9. Use a IN clause to select data from one or more tables\*/**

SELECT /\*fixed\*/ \* FROM Instr\_Info WHERE Instr\_LName IN ('Roberts', 'Russell');



**/\*10. Select Length of one column from one table (use Length function)\*/**

SELECT /\*fixed\*/ Course\_Desc, LENGTH (Course\_Desc) FROM Course\_List;



**/\*11. Use a column alias\*/**

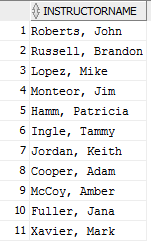
SELECT /\*fixed\*/ Instr\_FName AS InstructorName FROM Instr\_Info;



**/\*12. Perform an advanced query of your choice from chapter 8 (Database Systems Text Book - Coronel, Morris & Rob)\*/**

**/\*Return Instructor Full Name in following format: Last, First\*/**

SELECT /\*fixed\*/ Instr\_LName || ', ' || Instr\_FName AS InstructorName FROM Instr\_Info;



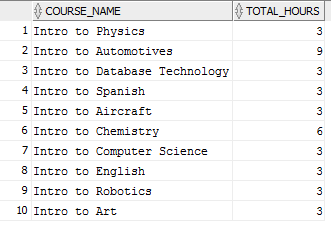
**/\*13. Use an aggregate function and perform another query\*/**

**/\*Return total number of course hours taught for each active class\*/**

SELECT /\*fixed\*/ Course\_Name, SUM(Course\_Hours) AS Total\_Hours

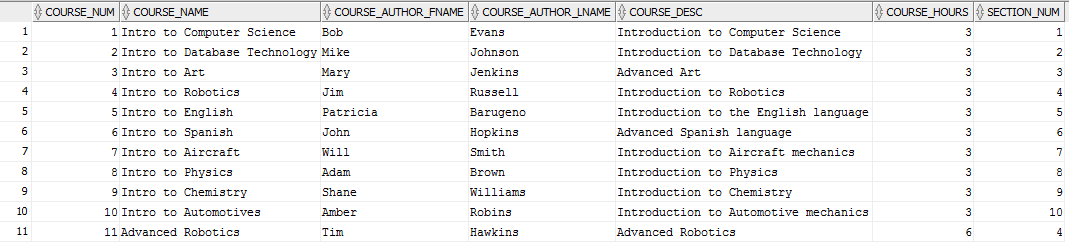
FROM Course\_List INNER JOIN Instr\_Classes ON Course\_List.Course\_Num=Instr\_Classes.Course\_Num

GROUP BY Course\_Name;



**/\*14. Use the UPDATE command and change some data.\*/**

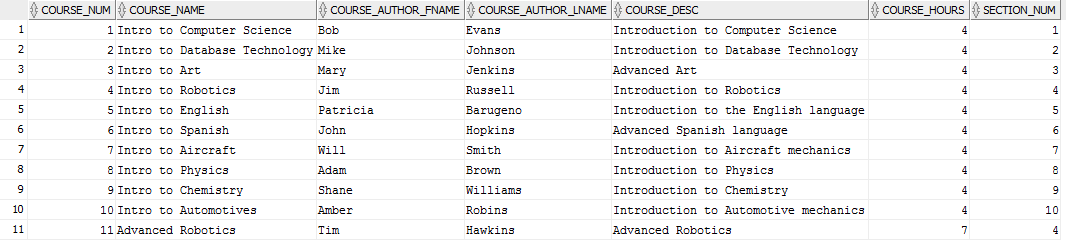
SELECT /\*fixed\*/ \* FROM Course\_List;



UPDATE Course\_List SET Course\_Hours=Course\_Hours+1;

11 rows updated.

SELECT /\*fixed\*/ \* FROM Course\_List;



ROLLBACK;

Rollback complete.

**/\*15. Write an advanced SQL statement with a type I subquery (chapter 8).\*/**

**/\*Return list of Instructors teaching classes with "Auto" in course name using Type I query\*/**

SELECT /\*fixed\*/ Instr\_Num, Instr\_LName, Instr\_FName

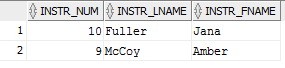
FROM Instr\_Info

WHERE Instr\_Num IN

(SELECT Instr\_Num FROM Instr\_Classes INNER JOIN Course\_List

ON Instr\_Classes.Course\_Num=Course\_List.Course\_Num

WHERE Course\_Name LIKE '%Auto%');



**/\*16. Write an advanced SQL statement with type II subquery (chapter 8).\*/**

**/\*Return list of Instructors teaching classes with "Auto" in course name using Type II query\*/**

SELECT /\*fixed\*/ Instr\_Num, Instr\_LName, Instr\_FName

FROM Instr\_Info

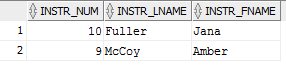
WHERE EXISTS

(SELECT \* FROM Instr\_Classes INNER JOIN Course\_List

ON Instr\_Classes.Course\_Num=Course\_List.Course\_Num

WHERE Instr\_Info.Instr\_Num=Instr\_Classes.Instr\_Num

AND Course\_Name LIKE '%Auto%');



**/\*17. Perform additional advanced SQL statement. (chapter 8).\*/**

**/\*Return a list of courses and how many times they are currently being taught\*/**

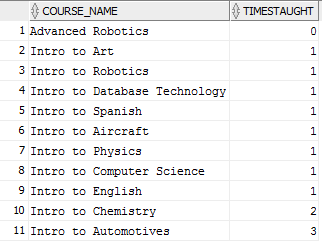
SELECT /\*fixed\*/ Course\_Name, COUNT(Instr\_Classes.Course\_Num) AS TimesTaught

FROM Course\_List LEFT JOIN Instr\_Classes

ON Instr\_Classes.Course\_Num=Course\_List.Course\_Num

GROUP BY Course\_Name

ORDER BY TimesTaught;



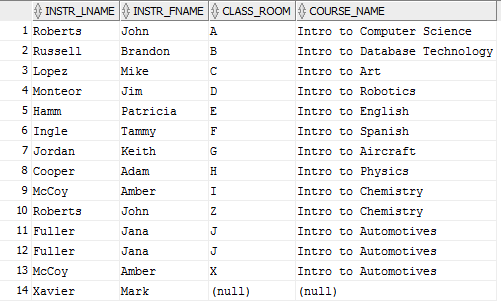
**/\*18. Perform additional advanced SQL statement. (chapter 8).\*/**

**/\*Return all Instructor names, class rooms, and courses taught\*/**

SELECT /\*fixed\*/ a.Instr\_LName, a.Instr\_FName, b.Class\_Room, c.Course\_Name

FROM (Instr\_Info a LEFT JOIN Instr\_Classes b ON a.Instr\_Num=b.Instr\_Num)

LEFT JOIN Course\_List c ON b.Course\_Num=c.Course\_Num;



**/\*19. Perform additional advanced SQL statement. (chapter 8).\*/**

**/\*Return a list of sections not currently offering any courses\*/**

SELECT /\*fixed\*/ Section\_Name

FROM Section\_Info

WHERE NOT EXISTS

(SELECT Section\_Num FROM Course\_List

WHERE Course\_List.Section\_Num=Section\_Info.Section\_Num);



**/\*20. Perform additional advanced SQL statement. (chapter 8).\*/**

**/\*5 table join-Show all courses being taught with Section & Instructor Name,**

**Class Room, and Schedule; ordered by Section and Instructor last name\*/**

SELECT /\*fixed\*/ e.Section\_Name, a.Instr\_LName || ', ' || a.Instr\_FName AS Instructor,

b.Class\_Room, c.Course\_Name, d.Sched\_Day || ', ' || d.Sched\_Time AS Schedule

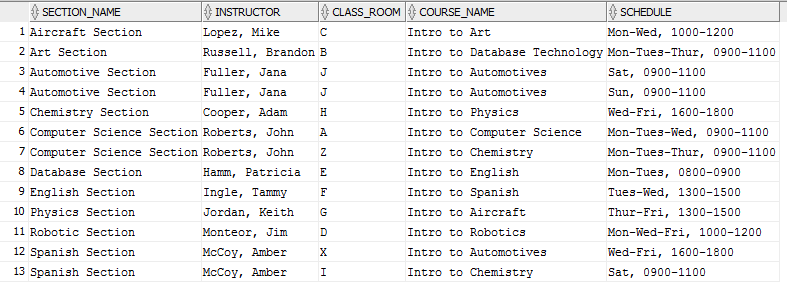
FROM (((Instr\_Info a INNER JOIN Instr\_Classes b ON a.Instr\_Num=b.Instr\_Num)

INNER JOIN Course\_List c ON b.Course\_Num=c.Course\_Num)

INNER JOIN Class\_Sched d ON b.Sched\_Num=d.Sched\_Num)

INNER JOIN Section\_Info e ON a.Section\_Num=e.Section\_Num

ORDER BY e.Section\_Name, a.Instr\_LName;



# Conclusion

This was a challenging project and I learned quite a lot from it. The more important things I felt I learned are as follows:

1. SQL Developer is a very powerful tool for database development. When I first started using it I thought it was just a simple tool for creating SQL statements. However, I quickly realized it is a very effective management tool and encompasses everything needed to create and maintain databases. Being able to utilize the query builder when creating select statements made creating queries much more efficient. Additionally, having a bird’s eye view of all the tables proved an effective multiplier in development.
2. I was unfamiliar with triggers before I began working on this project. I now know triggers are very powerful miniature programs embedded within a SQL file which can fire on specific events, similar to event-driven programming. In my project I was able to use a trigger to print a message to the screen when a new row of data was inserted into a table.
3. I knew of the JOIN operator before the beginning of this project, but this project has deepened my understanding of their true power. For example, I was able to create a join between two tables and then use a subquery to return a list of Instructors teaching classes about “Auto” with the LIKE operator. I was also able to use a LEFT JOIN and aggregate function together to compile a list of how many times a course was being taught.
4. I have built many databases in the past, but before this project creating an ERD was almost an afterthought. I now understand having an ERD with a properly laid out database design is a crucial step before you begin writing code. I referenced my ERD during all parts of this project and it proved to make my coding more efficient.
5. My Instr\_Classes table is a weak entity. With that I was unsure what would happen if I deleted a referenced entity instance in my parent tables. During this project I learned about the “ON DELETE CASCADE” operator. This is a very important operator as it will ensure any entity instance in the child table is deleted if their associated parent table entity instance is deleted.