**Principal Analytics Prep**

**Certified Data Analytics Practitioner (CDAP)**

**Sprint 2018**

**Syllabus**

**Module Title: Databases I**

**Instructor: Sam Sultan**

4 sessions, 3 hours per session

**Instructor Bio**

Over 25 years of experience in Information Technology. Expertise in all aspects of software engineering and construction lifecycle including requirement gathering, project management and costing, analysis, design, coding, testing and client training. Led many successful small, medium and large custom development projects, as well as full ERP implementation. Knowledgeable in traditional, object-oriented and agile development methodologies. NYU Assistant Professor, teaches graduate courses in Management and Information Technology with skills in Java, web development, SQL, data warehousing and many others.

<http://samsultan.com/>

**Learning Objectives**

At the end of this course, you will be able to:

* Understand what databases are and the difference between the various database models
* Appreciate the power and flexibility of the relational database model
* Understand the various terms used in conjunction with SQL
* Work with SQL as it related to both Oracle and MySql
* Master how to join tables to access data from multiple database tables
* Differentiate between inner and outer joins and when to use each
* Obtain a working knowledge of the various SQL functions
* Become proficient with queries and sub-queries
* Design databases and how to build databases tables and indexes
* Execute insert, update and delete data from database tables

**Course Requirements**

This course uses a Pass/Fail grading system. However to obtain a passing grade, students must complete and submit all assigned homeworks by next session. Students must also demonstrate full understanding of the subject matter by asking relevant/informative questions, participating in class by offering their opinions on the subject, sharing ideas or discussion, and commenting on others.

**Weekly Schedule**

|  |  |  |
| --- | --- | --- |
| **Week #** | **Topics** | **Assignment Due** |
| 1 | Introduction to databases  What is a database?  Relational databases  Other database models  Introduction to SQL  Flavors of SQL  DDL - Data Definition Language  DML - Data Manipulation Language  The SELECT statement  Choosing distinct values  The WHERE clause  Comparison operators  Comparing with LIKE  Logical operators, AND, OR, NOT  Numeric operators  Creating computational columns | Reading: *see blow* |
| 2 | Selecting data from multiple tables  The join construct  Old vs. new join syntax  Normal or Inner join  Cross join - Cartesian product  Outer join vs. Inner join  What is a Self-Join?  Set operators, UNION, INTERSECT, MINUS  Combining Join with UNION  SQL built-in Functions  Numeric functions  String functions  The CASE expression, 2 flavors  Date functions - MySQL and Oracle  Current date, date manipulation & formatting | Reading: *see below*  Homework: *see below*  Homework due next week |
| 3 | Aggregating and Grouping  Aggregate functions  The GROUP BY clause  The HAVING clause  Finding Duplicate Records  GROUP BY with ROLLUP feature  The ORDER BY clause  Pivoting rows into columns  Using SELECT Sub-Queries  Subqueries as filters  Subqueries as inline views  Correlated Subqueries  Where [NOT} EXISTS in Subquery | Reading: *see below*  Homework: *see below*  Homework due next week |
| 4 | Database Design  The Logical and Physical Model  Understanding data normalization  First, Second and Third normal form  Pros & cons of data normalization  Denormalizing data  Entity relationships  One-to-one relationship  One-to-many relationship  Many-to-many relationship  Designing for an ODS (Reporting Database)  Designing for a Data Mart/Warehouse  What is a primary key?  What is a foreign key?  What is an index?  Creating database tables  What is a view?  Creating a database view | Reading: *see below*  Homework: *see below*  Homework due next week |

**Reading**

**Book:** **Title:** Teach Yourself SQL in One Hour a Day (5th Edition)

**Author:** Ryan Stephens, Ronald Plew & Arie Jones

**Publisher:** SAMS

Session 1 - Reading: chapters 1, 2, 3

Session 2 - Reading: chapters 5, 7 (skip aggregate functions), 12

Session 3 - Reading: chapters 4, 6, 7 (aggregate functions)

Session 4 - Reading: chapters 8, 9, 10

**Homework for Session 2**

Create SELECT statements that would output

1. List all students and course ids of classes they are currently taking   
   *(expected number of rows - 50)*
2. List all students, course ids and course descriptions for courses they are currently taking   
   *(expected number of rows - 50)*
3. List all instructors, and their current students   
   *(expected number of rows - 50)*
4. List all students, course ids and course descriptions. Include all student regardless of whether they are currently taking a class or not (think outer join)   
   *(expected number of rows - 53)*
5. List all instructors (currently teaching or not), classes they are teaching, and all their students (as well as students not taking any classes)   
   - Data from all 3 tables (instructor, class, student) must be all on the same line.  
   - Hint: in Oracle it is easy. How would you do this in MySql? Think UNION  
   *(expected number of rows - 55)*  
     
   Partial output for 5 should look like

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instructor fname** | **Instructor lname** | **Course Id** | **Student fname** | **Student lname** |
| Sam | Sultan | X52-9272 | Vincent | Cambria |
| Sam | Sultan | X52-9272 | David | Smith |
| George | Pefanis | X52-9740 | Barbara | Burns |
| Eric | Katz | X52-9755 | Duncan | Davidson |
| Marc | Paller |  |  |  |
|  |  |  | David | Chan |

* Use both **old join syntax** (when possible) and **new join syntax** to write the above queries
* Write **1-4 in either** MySql or Oracle. Write **5 in both** MySql and Oracle

**Homework for Session 3**

Perform the following query in either **MySql**, **Oracle** or both

1. Count, sum and average the prices of all courses
2. Aggregate by summing the amount and counting the number of payments per vendor and per description.   
   Rollup this data up to the vendor level and grand total level
3. Verify to make sure that no student is registered for the same course twice.
4. Display all students (including students taking no classes) and the number of classes they are taking.   
   Sort the output by descending number of classes taken, then by last name and first name.
5. Display all vendors, descriptions and amounts from the payment table,   
   and a row at the bottom that displays the total amount of all payments made   
   PS. Do not display sub-total lines - i.e. do not use 'rollup' feature.
6. List all instructors that are currently not teaching any classes.   
   (I am not looking for a list of active/inactive status from instructor\_info table)   
   Here's one way to do this, can you think or another way?   
     
   SELECT fname, lname   
   FROM instructor LEFT JOIN class on ssn=inst\_ssn   
   WHERE class\_id IS NULL
7. From the payment table  
   List all vendors whose "total" amount of payments exceed the average amount for the entire payment table   
   (i.e. you must total all payments for each vendor, and then compare that total to the average payment for the entire table).   
   - HINT: Use subquery as a filter
8. List all students and courses they are taking.   
   Produce the output where the list of courses is displayed horizontally   
   (i.e. courses are listed as column oriented not as row oriented).   
   - HINT: Use inline views

**Homework for Session 4**

1 - Create a **logical database model** that includes the following tables.

* Customer table
* Customer address table
* Customer order table
* Customer order address table
* Customer order line detail table
* Product table
* Vendor table

- Add whatever columns you feel are necessary to support an order entry process.  
- Relate the tables using 1:1, 1:M or N:M relationships as see fit.   
- Do not worry about column types, or primary/foreign keys yet.   
- Diagram should show the various columns and relationships between the tables.

2 – Use the above logical model to create the **physical** objects:

* Create tables that contain the data elements you have chosen for each table
* Create primary keys for each table
* Primary keys should use auto-increment (MySql) or sequences (Oracle)
* Create addition indexes on some columns as you see fit (example on customer name)
* Optionally, create foreign key constraint between tables