CS 480 Midterm 2

Christian Dominguez

TOTAL POINTS

89.75 / 100

QUESTION 1

1 2 / 2

✓ **+ 0 pts** **Create**

✓ **+ 0.5 pts** **INSERT**

✓ **+ 0 pts** **Read**

✓ **+ 0.5 pts** **SELECT**

✓ **+ 0 pts** **Update**

✓ **+ 0.5 pts** **UPDATE**

✓ **+ 0 pts** **Delete/destroy**

✓ **+ 0.5 pts** **DELETE**

+ 0 pts Operations not described

+ 1 pts CRUD missing SQL keywords

+ 0 pts Modify

+ 0.25 pts CREATE

+ 0.25 pts ALTER

+ 0.25 pts DROP

+ 0 pts Add

+ 0 pts Remove

+ 0 pts Insert

+ 0 pts Retrieve

+ 0 pts Manipulate

+ 0 pts USE

+ 0.5 pts Numeric Functions

+ 0 pts Other Functions

QUESTION 2

2 3 / 3

✓ **+ 1.5 pts** **Unique**

✓ **+ 1.5 pts** **NOT NULL**

+ 1.5 pts INDEXed

+ 1 pts Identifies row

+ 1 pts Used to connect tables together (foreign key, identifies row)

+ 1 pts There wasn't a unique attribute like ID to sufficiently uniquely identify - incorrect, you can have a set of size 1

+ 0 pts Helps with aggregate functions for ordering

+ 0 pts Since there is more than one primary key, it also means that those keys are foreign keys to another table - false on both counts

+ 0 pts Candidate key

+ 0 pts Identifies/Distinguishes table

+ 0 pts Used for SELECT

+ 0 pts No Answer

QUESTION 3

3 2 / 2

+ 2 pts TRANSACTION

✓ **+ 2 pts** **SOURCE**

+ 2 pts FUNCTION

+ 2 pts PROCEDURE

+ 2 pts TRIGGER

+ 2 pts BEGIN...END

+ 2 pts DELIMITER - it always feels like such a hack, but I haven't found a way around it at the command line

+ 1 pts Nested statements or subqueries

+ 1 pts SELECT or functions like SUM(), MAX(), MIN() for example

+ 1 pts GRANT (Authorization) - focused on allows

+ 2 pts Running sql files. (Source)

+ 0 pts SELECT/ Queries

+ 0 pts Aggregate Functions

+ 1 pts Cascade

+ 1 pts Cases. supports conditional statements

+ 2 pts Execute

+ 0 pts USE command

+ 0 pts Schema and Domain of values

+ 0 pts Use database;

+ 0 pts the compiler

+ 1 pts add a semicolon to do a sequence of related

instructions - ?

+ **0 pts** mysql

+ **1 pts** Union

+ **0 pts** Parenthesis

+ **0 pts** No Answer

QUESTION 4

4 3 / 3

+ **0 pts** ----------Answers with one or more of below----------

✓ **+ 1 pts** **CASCADE**

✓ **+ 1 pts** **SET NULL**

✓ **+ 1 pts** **SET DEFAULT**

+ **1 pts** RESTRICT - in MySQL the default behavior, cause the operation to fail

+ **1 pts** DO NOTHING - In other versions of SQL this can be used to defer a check, in my SQL it doesn't do nothing at all, it actually performs the RESTRICT operation

+ **1 pts** Ignore Changes ---DO NOTHING?

+ **1 pts** TRIGGER - you could set up a trigger

+ **1 pts** Updates - I assume you mean CASCADE

+ **1 pts** Does not update/delete/No action - I assume this means RESTRICT?

+ **0 pts** 'Some values are affected' - Partially Committed?

+ **1 pts** 'If T2 deleted set Foreign key to null' - Set NULL?

+ **0 pts** Is Null

+ **0.5 pts** Not Null

+ **0.5 pts** CHECK

+ **0.5 pts** Unique

+ **1 pts** Delete Data Referenced by Keys/ Delete Keys

+ **0.5 pts** Behavior/Constraints related to Primary Key

+ **0.5 pts** Behavior/Constraints related to Foreign Key

+ **0 pts** " not instance of"

+ **0 pts** If Exists/ Not Exists

+ **0 pts** Schedules

+ **0 pts** Control Schemes

+ **0 pts** Error to indicate that keys are invalid

+ **0 pts** Error from violating attributes with unique constraint

+ **1 pts** Error if keys are modified/ Throw Error

+ **0 pts** Corrupted Database/Runtime Logic Error?

+ **0 pts** null pointers

+ **0 pts** T1 Complies with its Integrity Constraints - these changes are the ones I want you to list

+ **0 pts** Save latest value before change occurs

+ **1 pts** There could be an insertion, update, or deletion on Table 1 where Table 2 is dependent on

+ **0.5 pts** warning?

+ **0 pts** No Answer

+ **0 pts** handle(?) via Functions

+ **0 pts** ----------3-Tuple Answers----------

+ **1.5 pts** NOT NULL, UNIQUE, CHECK - wrong answer, these are constraints not the responses to constraints

+ **1.5 pts** NOT NULL, CHECK, REFERENCES

+ **1 pts** NOT NULL, PRIMARY KEY, FOREIGN KEY

+ **1 pts** not null, check, default

+ **1 pts** Not Null, Null, Default

+ **1 pts** not null, unique, primary key unique

+ **1 pts** not null, primary key, exists/not exists

+ **2 pts** default, no action, not null

+ **2 pts** Null, Error, Corrupt

+ **2 pts** Cannot be null, Wrong type, Already Exists

+ **0 pts** Integrity,Referential Integrity,Domain Constraints

+ **2 pts** "Unable To Alter Data", "Key Unavailable", "Error"

+ **2 pts** Failed, Partially Committed, Aborted

+ **2 pts** Alerts, Warnings, Pings

+ **1 pts** IsNull, Not Unique, Not Primary Key

+ **2 pts** Panic, DROP TABLE T1, nothing

+ **0 pts** ----------NOTES ----------

+ **0 pts** CASCADE is DELETE when the operation is ON DELETE, not different from CASCADE on UPDATE

QUESTION 5

5 3 / 3

+ 3 pts CREATE DATABASE Music;
USE Music;

✓ + 3 pts **CREATE DATABASE IF NOT EXISTS Music;**
USE Music;

+ 3 pts DROP DATABASE IF EXISTS Music;
CREATE DATABASE Music;
USE Music;

+ 3 pts CREATE DATABASE Music IF NOT ALREADY EXISTS();
USE Music; - bad syntax

+ 1 pts CREATE Music;
USE Music;

+ 1 pts CREATE TABLE Music(name varchar(100), year_founded int, PRIMARY KEY(name));

+ 3 pts CREATE DATABASE Music;
USING Music; - bad syntax

+ 3 pts CREATE DATABASE Music;
USING Music go; - bad syntax

+ 2 pts USE Music;
CREATE DATABASE Music;

+ 3 pts DROP IF EXISTS Music;
CREATE DATABASE Music;
USE Music;

+ 3 pts DROP IF EXISTS DATABASE Music;
CREATE DATABASE Music;
USE Music;

+ 3 pts CREATE DATABASE Music IF NOT EXISTS;
USE Music;

+ 3 pts CREATE DATABASE Music;
USE DATABASE Music;

QUESTION 6

6 5 / 5

✓ + 1 pts **CREATE TABLE Song**

✓ + 1 pts **song_id int**

or

song_id int(x)

or

song_id numeric(x)

or

song_id char(x)

or

song_id varchar(x)

+ 0.5 pts song_id AUTO_INCREMENT

+ 0.5 pts ID but missing type

✓ + 1 pts **song_name varchar(x)**

or

song_name text

+ 0.5 pts song_name string

✓ + 0.5 pts **song_name NOT NULL**

+ 0 pts song_name default NULL

✓ + 1 pts **song_length double**

or

song_length int

or

song_length int(x)

or

song_length timestamp

or

song_length numeric(x, [y])

or

song_length decimal(x, y)

+ 1 pts song_length time

or

song_length time(x, x, x)

+ 0.5 pts song_length varchar(x)

✓ + 1 pts **song_price numeric(x, y)**

or

song_price float(x, [y])

or

song_price decimal(x, y)

+ 1 pts song_price double(x, y)

+ 0.5 pts song_price int

+ 1 pts song_price number(x, y)

+ 0.5 pts song_price varchar(x)

+ 1 pts song_price float()

+ 1 pts artist_name varchar(x)
PRIMARY KEY(artist_name)
or
artist_name varchar(x)
PRIMARY KEY(artist_name, song_name)

+ 0 pts artist_id numeric(x, [y])

or

artist_name varchar(x)

or

artist_id int

FOREIGN KEY(artist_id references Artist)

or

artist_name varchar(x)

FOREIGN KEY(artist_name references Artist)

✓ **+ 1 pts PRIMARY KEY(song_id)**

+ 0.5 pts missing song_id but has PRIMARY KEY(song_id)

+ 0.5 pts PRIMARY KEY(song_name)

+ 1 pts PRIMARY KEY(song_name, length)

+ 0.5 pts PRIMARY KEY(song_length, song_price)

+ 0 pts CREATE DATABASE IF NOT EXISTS Song - Tries to create database instead of table

✓ **- 0.5 pts numeric(x,x), decimal(x, x) or float(x, x) means the number is strictly a decimal value less than 1**

+ 0 pts float(x+1, x), decimal(x+1, x) or numeric(x+1, x) is a decimal value with value less than 10

- 0.5 pts float(2) or decimal(2) is a number with a total of only two digits.

- 0.5 pts float(1,2)?

- 0.5 pts FOREIGN KEY(song_name)

+ 0 pts integer instead of int

+ 0 pts dec instead of decimal

+ 0 pts artist_name being in the song table implies that a song may only be written by a single artist (collaborations not allowed)

- 0.5 pts There is no ON INSERT behavior because there is no FOREIGN KEY TO attach this behavior to

+ 0 pts Sloppy Syntax

+ 0 pts int with range

- 0.5 pts FOREIGN KEY artist_name when no field by that name in table

- 0.5 pts Missing name of field

QUESTION 7

7 5 / 5

✓ **+ 1 pts CREATE TABLE Artist**

✓ **+ 1 pts artist_id int**

or

artist_id int(x)

or

artist_id numeric(x, [y])

or

artist_id varchar(x)

+ 0 pts artist_id char()

+ 0.5 pts artist_id but missing type

+ 0.5 pts artist_id AUTO INCREMENT

✓ **+ 2 pts artist_name varchar(x)**

✓ **+ 0.5 pts artist_name NOT NULL**

+ 2 pts artist_name string

or

artist_name string(x)

or

artist_name text

+ 0 pts artist_name default NULL

✓ **+ 1 pts artist_date_of_entry int**

or

artist_date_of_entry int(x)

or

artist_date_of_entry numeric(x, [y])

or

artist_date_of_entry varchar(x)

+ 1 pts artist_date_of_entry date

or

artist_date_of_entry date()

or

artist_date_of_entry date(x, x, x)

or

artist_date_of_entry date('yyyy')

or

artist_date_of_entry date(x)

or

artist_date_of_entry date(year)

+ 1 pts artist_date_of_entry time(x)

+ 1 pts artist_date_of_entry year

or

artist_date_of_entry year()

or

artist_date_of_entry year(x)

+ 0 pts artist_date_of_entry char(x)

- **0.5 pts** artist_number_of_songs int

or

artist_number_of_songs int(x)

✓ **+ 1 pts** **PRIMARY KEY(artist_id)**

+ **0.5 pts** missing artist_id but has PRIMARY KEY(artist_id)

+ **0.5 pts** PRIMARY KEY(artist_name)

+ **1 pts** PRIMARY KEY(artist_name, artist_date_of_entry)

+ **0 pts** CREATE DATABASE IF NOT EXISTS Artist

- **0.5 pts** There is no ON INSERT behavior because there is no FOREIGN KEY TO attach this behavior to

- **0.5 pts** FOREIGN KEY song_name when no field by that name in table

+ **0 pts** FOREIGN KEY artist_name references Song(name, length) when artist_name is part of the primary key

+ **0 pts** integer instead of int

+ **0 pts** sloppy syntax

QUESTION 8

8 1.75 / 2

✓ **+ 0.5 pts** **CREATE TABLE PerformedBy**

✓ **+ 0.25 pts** **song_id int**

or

song_id int(x)

or

song_id numeric(x, y)

or

song_id text

✓ **+ 0 pts** **song_id NOT NULL**

+ **0.25 pts** artist_id int

or

artist_id numeric(x, y)

or

artist_id text

+ **0 pts** artist_id NOT NULL

+ **0 pts** performance_id int(x)

or

performance_id numeric(x, y)

or

performance_id varchar(x)

+ **0 pts** performance_id NOT NULL

✓ **+ 0.25 pts** **song_name varchar(x)**

✓ **+ 0 pts** **song_name NOT NULL**

✓ **+ 0.25 pts** **artist_name varchar(x)**

✓ **+ 0 pts** **artist_name NOT NULL**

✓ **+ 0.5 pts** **PRIMARY KEY(song_id)**

+ **0.5 pts** PRIMARY KEY(song_id, artist_id)

+ **0.5 pts** PRIMARY KEY(song_name, artist_name)

+ **0.5 pts** PRIMARY KEY (performance_id)

+ **0 pts** PRIMARY KEY(artist_name)

+ **0 pts** PRIMARY KEY(song_name)

+ **0 pts** PRIMARY KEY(song_name, artist_name) when they are not defined

+ **0.25 pts** FOREIGN KEY (song_id) REFERENCES Song(song_id)

or

FOREIGN KEY (song_id) REFERENCES Song(song_id) ON DELETE CASCADE

or

FOREIGN KEY (song_id) REFERENCES Song(song_id) ON DELETE SET NULL

+ **0.25 pts** FOREIGN KEY (artist_id) REFERENCES Artist(artist_id)

or

FOREIGN KEY (artist_id) REFERENCES Artist(artist_id) ON DELETE CASCADE

or

FOREIGN KEY (artist_id) REFERENCES Artist(artist_id) ON DELETE SET NULL

+ **0 pts** FOREIGN KEY (song_name) REFERENCES Song(song_name)

+ **0 pts** FOREIGN KEY (artist_name) REFERENCES Artist(artist_name)

- **0.25 pts** FOREIGN KEY reference when missing song_id and artist_id

+ **0.25 pts** FOREIGN KEY name references Song, Artist

- **0.25 pts** FOREIGN KEY reference when missing song_name and artist_name

+ **0.5 pts** REFERENCES Song(song_name), Artist(artist_name)

+ **0.5 pts** FOREIGN KEY(song_id), FOREIGN

KEY(artist_id)
+ **0.5 pts** FOREIGN KEY(song_name), FOREIGN KEY(artist_name)
+ **0.25 pts** FOREIGN KEY(Song, Artist)
+ **0.5 pts** CREATE TABLE PerformedBy(Song, Artist)
or
CREATE TABLE PerformedBy(Song UNION Artist)
or
CREATE TABLE PerformedBy(USE Song, USE Artist)
+ **0 pts** SELECT * FROM Song NATURAL JOIN Artist
or
SELECT * FROM Song NATURAL JOIN Artist ON artist_name = song_name
+ **0 pts** PerformedBy NATURAL JOIN Song NATURAL JOIN Artist
+ **0.25 pts** song_name (SELECT song_name FROM Song)
artist_name (SELECT artist_name FROM Artist)
+ **0 pts** artist_date_of_entry date
song_length numeric(x, y)
song_price numeric(x, y)
+ **0 pts** CREATE DATABASE instead of CREATE TABLE
- **0.25 pts** fields missing types
+ **0 pts** also has artist_date_of_entry to make a composite key with artist_name
+ **0 pts** LINK Song AND Artist AS PerformedBy
+ **0 pts** text description
+ **0 pts** sloppy syntax
+ **0 pts** blank

QUESTION 9

9 5 / 5

✓ **+ 1 pts INSERT INTO Artist**

✓ **+ 1 pts some arrangement of (artist_id, artist_name, artist_date_entry)**

+ **1 pts** some arrangement of (artist_name, artist_date_entry)

✓ **+ 3 pts VALUES (ID, "Journey", 1973)**

+ **3 pts** VALUES ("Journey", 1973)
+ **3 pts** VALUES("Journey", DATE(1973))
or
VALUES("Journey", DATE(1973, 1))
+ **3 pts** VALUES("Journey", MAKEDATE(1973, 1))
+ **0.5 pts** INSERT INTO Music
+ **0.75 pts** INSERT Artist
+ **4 pts** INSERT Artist(
artist_name "Journey"
artist_date_of_entry 1973)
+ **4 pts** (artist_name, group_name, year) VALUES("Journey", "Journey", 1973)
- **0.5 pts** missing VALUES
+ **2 pts** ALTER TABLE Artist add("Journey", 1973)

QUESTION 10

10 5 / 5

✓ **+ 2.5 pts INSERT INTO Song(song_id, song_name, song_length, song_price) VALUES(x, "Open Arms", 199, 1.99)**
or
INSERT INTO Song(song_id, song_name, song_length, song_price) VALUES(x, "Open Arms", 3.19, 1.99)
or
INSERT INTO Song VALUES(x, "Open Arms", 199, 1.99)
or
INSERT INTO Song VALUES(x, "Open Arms", 00:03:19, 1.99)

+ **2.5 pts** INSERT INTO Song(song_name, song_length, song_price) VALUES("Open Arms", 199, 1.99)
or
INSERT INTO Song(song_name, song_length, song_price) VALUES("Open Arms", 00:03:19, 1.99)
or
INSERT INTO Song VALUES("Open Arms", 199, 1.99)
+ **2.5 pts** INSERT INTO Song(song_id, artist_id, song_name, song_length, song_price) VALUES(x, y, "Open Arms", 199, 1.99)
or
INSERT INTO Song VALUES(x, y, "Open Arms", 199, 1.99)
+ **2.5 pts** INSERT INTO Song(song_name,

artist_name, song_length, song_price)
VALUES("Open Arms", "Journey", 199, 1.99)
or
INSERT INTO Song VALUES("Open Arms", "Journey", 199, 1.99)

+ **2 pts** INSERT INTO Song VALUES(x, "Open Arms", "Journey", 199, 1.99)

+ **2 pts** INSERT INTO Song VALUES(x, y, "Open Arms", "Journey", 199, 1.99)

+ **1.5 pts** INSERT INTO Song(song_length, song_price) VALUES(3.19, 1.99)

+ **2.5 pts** INSERT INTO PerformedBy VALUES(SELECT artist_id FROM Artist WHERE artist_name="Journey",
SELECT music_id FROM Song WHERE song_name="Open Arms")

✓ **+ 2.5 pts INSERT INTO PerformedBy(song_id, song_name, artist_name) VALUES(x, "Open Arms", "Journey")**
or
INSERT INTO PerformedBy(performance_id, song_name, artist_name) VALUES(x, "Open Arms", "Journey")

+ **2.5 pts** INSERT INTO PerformedBy(song_name, artist_name) VALUES("Open Arms", "Journey")
or
INSERT INTO PerformedBy VALUES("Open Arms", "Journey")

+ **2.5 pts** INSERT INTO PerformedBy(song_id, artist_id) VALUES(x, y)
or
INSERT INTO PerformedBy VALUES(x, y)

+ **2.5 pts** INSERT INTO PerformedBy(performance_id, song_id, artist_id) VALUES(x, y, z)

+ **2.5 pts** INSERT INTO PerformedBy VALUES("Get artist_ID of Journey", "Get song_ID of Open Arms") - has text description

+ **2.5 pts** INSERT INTO PerformedBy VALUES("Open Arms", "Journey", 199, 1.99)

- **0.5 pts** INSERT INTO Music instead of INSERT INTO Song

+ **1 pts** INSERT INTO Music(1, 1, 1)

+ **1 pts** UPDATE Artist SET songsreleased=1 WHERE artist_name="Journey"

+ **3 pts** ALTER TABLE Song ADD("Open Arms", 00:03:19, 1.99)
ALTER TABLE Artist ADD("Open Arms", "Journey")

+ **2 pts** INSERT INTO Song VALUES("Journey", 199, 1.99)

+ **2 pts** INSERT INTO PerformedBy VALUES(x, "Journey", 1973)

+ **1 pts** INSERT INTO PerformedBy(Artist Name("Journey"))

+ **2 pts** SELECT artist_id FROM Artist LIMIT 1 UNION INSERT INTO Song VALUES("Open Arms", 199, artist_id)

+ **0.5 pts** text description

+ **0 pts** blank

QUESTION 11

11 4 / 5

+ **5 pts** Correct answer

✓ **+ 1 pts SELECT artist_name, num_songs correctly**

+ **0.5 pts** Selects ArtistName and COUNT without renaming

+ **0.5 pts** SELECT artist_name correctly

+ **0 pts** Incorrect selection of num_songs

+ **3 pts** FROM Artist LEFT JOIN PerformedBy
or
FROM Artist NATURAL JOIN PerformedBy
or
FROM Artist JOIN PerformedBy

+ **2 pts** FROM Artist RIGHT JOIN PerformedBy

+ **3 pts** FROM Artist JOIN PerformedBy JOIN Song

✓ **+ 3 pts Due to alternate design, SELECT from PerformedBy alone**

+ **3 pts** Due to alternate design, SELECT from Artist JOIN Song

+ **0 pts** FROM PerformedBy - no artist_name in this table, need to join with Artist table

+ **1 pts** GROUP BY artist_name
or
GROUP BY artist_id

+ **0 pts** GROUP BY song_id or GROUP BY num_songs - incorrect, should group by artist_name

- **1 pts** INNER JOIN instead of LEFT JOIN

- **1 pts** JOIN missing condition to join on

✓ **+ 0 pts Missing GROUP BY**

+ **0 pts** using SUM instead of COUNT to aggregate

+ **0 pts** ORDER BY artist_name

+ **0 pts** CREATE TABLE PerformedBy with various fields

+ **0 pts** SELECT artist_name, num_songs FROM Music

+ **5 pts** Alternate design, stores and retrieves num_songs field directly without using COUNT; retrieves artist_name as well

+ **0 pts** Alternate design also incorrect

+ **1 pts** Using Artist and Song tables for selection without the info from PerformedBy table - how to determine which artist performed which song?

+ **0 pts** Using Song table which does not contain info about the artist - how to determine which artist performed which song?

or

Using Artist table which does not contain info about the songs - how to determine which artist performed which song?

+ **1 pts** Using JOIN between Artist and Song table which do not have common attributes - Name/id field might be named the same, but should not have the same names/values

+ **0 pts** Union is only possible if both tables have same attribute names

+ **0 pts** FROM Song, Artist - missing condition

+ **0 pts** Unclear

+ **5 pts** Table not defined for PerformedBy - uses this table in the query

QUESTION 12

12 3 / 5

+ **5 pts** Correct Table

✓ **+ 3 pts Missing a row in the table. Have 2 out of 3 rows correct.**

+ **2 pts** Column Description is incorrect. It should be ABC, ABC, GHI. (Rest are correct)

+ **2 pts** RecordNum is incorrect. it should be 1,4,3. (Rest are correct)

+ **1 pts** Missing two rows in the table. Have 1 out of 3 rows correct.

- **1 pts** Did not use Order By Table1.ID ASC (or ordered in DESC order)

- **1 pts** Extra data in the table

- **1 pts** Third row for Description should be GHI.

- **1 pts** missing a row in the table with incorrect column description or record num (ID:10 - RecordNum:4 - Description:GHI)

- **1 pts** incorrect row (35, null and DEF)

- **1 pts** missing recordnum column

- **0.5 pts** incorrect description column, should be ABC, ABC and GHI

QUESTION 13

13 0 / 5

+ **5 pts** Empty Set

✓ **+ 0 pts Incorrect**

QUESTION 14

14 5 / 5

✓ **+ 5 pts Correct Table**

- **1 pts** Table is not in descending order by PartySize

- **3 pts** Did not use GroupBy.

+ **2.5 pts** Partial Credit: Incorrect number for party size.

+ **0 pts** incorrect

- **1 pts** extra detail/row in table (Only Two rows needed: PartyID: 1,2 and PartySize: 2,1 respectively)

QUESTION 15

15 10 / 10

✓ **+ 2 pts SELECT DishName**

+ **1 pts** SELECT *

+ **0 pts** ----------1. FROM----------

✓ **+ 4 pts FROM Customers,Orders,Dishes**

+ **2 pts** FROM Customers,Dishes - no field to join on

+ **2 pts** FROM Orders,Dishes

+ **1 pts** FROM Dishes

+ **4 pts** FROM Customers,Orders,Dishes, Restaurant

+ **0 pts** ----------2. COND----------

✓ **+ 2 pts** **WHERE CustomerOrders.CustomerName=@C**

+ **2 pts** WHERE CustomerOrders.CustomerName LIKE '@C'

+ **2 pts** WHERE CustomerID=@C (treats @C as a customerID)

+ **2 pts** Retrieve CustomerID by customerName @C, store the result in a variable or subquery, join between orders and dishes

+ **1 pts** WHERE Customer=@C

+ **0 pts** ----------3. order by----------

✓ **+ 1 pts** **Order by DishType**

✓ **+ 1 pts** **ORDER BY Dishes.DishPrice DESC**

+ **0.5 pts** ORDER BY Dishes.DishPrice

+ **0.5 pts** Order by DishType DESC

+ **0.5 pts** ORDER BY Dishes.DishName ASC

- **0.5 pts** ORDER BY Dishes.DishName ASC AND Dishes.DishPrice DESC (should use comma to combine)

+ **0 pts** ORDER BY Dishes.DishName ASC ; Then order by Dishes.DishPrice DESC (should use comma to combine)

+ **0.5 pts** Order by @C

- **0.5 pts** Query to set CustomerID variable after query that uses it

+ **0 pts** ----------4. group by----------

- **0.5 pts** group by DishName ASC, DishPrice DESC

- **0.5 pts** group by DishName

- **0.5 pts** group by DishType

- **0.5 pts** GROUP BY Dishes.DishName ASC GROUP BY Dishes.DishPrice DESC

- **0.5 pts** GROUP BY (Alph(DishName), DESC(DishPrice))

+ **0 pts** ----------feedback ----------

+ **0 pts** should use CD1.CustomerID = CID2.CustomerID

+ **0 pts** @C in quotes

- **0.5 pts** Missing ON/WHERE from join condition

- **0.5 pts** Orders by DishPrice first

- **1 pts** ORDERS BY DishType, then DishName, then DishPrice

+ **0.5 pts** WHERE Dishes.DishID=Orders.DishID when FROM only contains Dishes

+ **1 pts** WHERE Customers.Customer.ID - no actual condition

+ **0 pts** ORDER BY CustomerName

QUESTION 16

16 9 / 10

✓ **+ 1 pts** **SELECT DishType**

+ **0 pts** SELECT DishName

✓ **+ 2 pts** **aggregate func (count)**

+ **1 pts** aggregate func (sum)

- **1 pts** Wrong expression/position of aggregate func

+ **0 pts** ----------Join ----------

✓ **+ 1 pts** **FROM Dishes , Orders**

+ **0 pts** FROM Customer , Orders

+ **0 pts** FROM Dishes

✓ **+ 2 pts** **Dishes LEFT JOIN Orders**

+ **2 pts** Dishes Outer JOIN Orders

+ **2 pts** Orders RIGHT JOIN Dishes

+ **2 pts** Dishes FULL JOIN Orders

+ **0 pts** Dishes RIGHT JOIN Orders

+ **0 pts** Orders LEFT JOIN Dishes

+ **0 pts** use inner join/natural join/join(Cartesian product)

+ **2 pts** Use inner/natural join, then use union to include dishtype that hasn't been ordered

+ **0 pts** ----------Group By----------

+ **2 pts** GROUP BY DishType

+ **1 pts** GROUP BY DishName

✓ **+ 1 pts** **GROUP BY DishId or count() (If using SELECT DishType, then you should group by DishType)**

+ **0 pts** ----------Order

By----------

✓ **+ 2 pts ORDER BY NumOrdered DESC**

+ 1 pts ORDER BY DishType DESC

+ 1 pts ORDER BY DishName DESC

- 0.5 pts sort by NumOrdered DESC

- 1 pts should use DESC

+ 1 pts FROM Dishes , Orders, Customers

+ 0 pts ----------

+ 1 pts Retrieve dishname first, list total num of times each dish has been ordered.

+ 0 pts retrive dishes that have been ordered by customers, then get union with dishes that haven't been ordered

+ 0 pts Having num_order is null set NumOrdered=0

QUESTION 17

17 5 / 5

✓ **+ 1 pts SELECT TableID**

+ 0 pts ----------FROM ----------

✓ **+ 2 pts FROM Waiters, Service**

+ 2 pts FROM Table, Waiters, Service

+ 1 pts FROM Waiters

+ 0 pts From Service

+ 1 pts FROM Tables INNER JOIN @W ON @W = Waiters.WaiterName

+ 0 pts ----------WHERE ----------

✓ **+ 2 pts Where Waiters.WaiterName=@W**

+ 2 pts Where Waiters.WaiterName LIKE @W

+ 0 pts ----------Deduction ----------

- 1 pts should use proper syntax in where

+ 0 pts Order by EmployeeID

+ 0 pts Order by TableID

+ 1 pts select WaiterName FROM Waiters Where Waiters.WaiterName = Waiters.WaiterName

+ 1 pts WHERE W.ID=Service.EmployeeID

QUESTION 18

18 10 / 10

✓ **+ 3 pts SELECT SUM(DishPrice)**

✓ **+ 5 pts FROM Dishes, Orders, CustomersInParty, Service**

+ 5 pts FROM Dishes, Orders, CustomersInParty, Service, Tables

+ 5 pts FROM Dishes, Orders, CustomersInParty, Service, Tables,Customers

+ 5 pts FROM Dishes, Orders, CustomersInParty, Service, Customers

✓ **+ 2 pts where TableId=@T**

+ 2 pts where TableId LIKE @T

+ 2 pts GROUP BY TableID Having TableID = @T

+ 0 pts ----------

+ 1 pts SELECT COUNT(DishPrice) -- inappropriate aggregate function

+ 1 pts SELECT DishPrice, COUNT(*)

+ 0 pts Select TableID

+ 0 pts FROM Dishes, @T

+ 2 pts FROM Dishes,Service, Table

+ 2 pts FROM Service, Orders, CustomersInParty

+ 2 pts FROM Dishes, Orders, CustomersInParty

+ 2 pts FROM Dishes, Orders, Customers

+ 2 pts FROM Dishes, Orders, Tables

+ 2 pts FROM Dishes, CustomersInParty, Service,Tables(NO Orders)

+ 2 pts FROM Dishes,Service

+ 1 pts FROM Dishes, Orders

+ 0 pts GROUP BY TableID

+ 0 pts GROUP BY CustomerName

+ 0 pts GROUP BY @T

+ 0 pts GROUP BY sum (disprice)

+ 0 pts GROUP BY Dishprice

+ 0 pts extract partyID related with table @T first, use it in the following step

+ 0 pts Retrive the total price paid by each customer (unclear)

+ 0 pts Retrive partyID for table @T, then calculate sum(dishprice) for those parties.

- 2 pts Do several seperate queries on Table Service, CustomerInParty, Dishes, Tables.

+ 0 pts Should use SELECT...INTO syntax

- 1 pts (Select partyID From Service where

tableID=@T) as A;
(Select CustomerID from CustomerInParty Where partyID = A.partyID)as B; (Select DishID from Orders where B.CustomerID = Orders.OrderID;) AS C;
Select SUM(DishPrice) as Total from Dishes where Dishes.dishID = C.DishID

QUESTION 19

19 7 / 10

+ **1 pts** Final result is renamed as Money
+ **1 pts** Final result is renamed as Total/Amount/Amt-paid

✓ **+ 0 pts Final result isn't renamed as Money**

+ **0 pts** Select sum(CustomerInParty)
+ **0 pts** Select count(Dishprice)
+ **0 pts** Select sum(Gratuity)
+ **0 pts** miss sum() to calculate dish price for party @P
+ **0 pts** ----------SUM(Dishprice)----------

✓ **+ 3 pts Gets sum of dish prices for party @P (CustomersInParty,Order, Dishes)**

+ **1 pts** Gets sum of dish prices for party @P (CustomersInParty, Dishes)
+ **1 pts** Gets sum of dish prices for party @P (Service)
+ **1 pts** Gets sum of dish prices for party @P (CustomersInParty)
+ **1 pts** Gets sum of dish prices for party @P (Order, Dishes)
+ **0.5 pts** Get sum of dish prices ordered by each customer
+ **0.5 pts** Get sum of dish prices for each party
- **0.5 pts** Get sum of dish prices from Question 18 directly(But Q18 retrieve sum of dish price by tableID)
+ **0 pts** ----------gratuity----------

✓ **+ 3 pts Get gratuity (Service,Bills) for party @P**

+ **1 pts** Get gratuity (Bills) for party @P
+ **1 pts** Get gratuity (Service) for party @P

✓ **+ 1 pts WHERE CustomerInParty.PartyID = @P**

+ **1 pts** WHERE CustomerInParty.PartyID LIKE @P
+ **0 pts** ----------SUM UP----------
+ **2 pts** Sum up sum(dish prices) and gratuity
+ **0 pts** Multiplies by 1+Gratuity/100 (Added to Applies gratuity (Service,Bills))-----CORRECT
- **0.5 pts** Multiplies by 1+Gratuity (Added to Applies gratuity (Service,Bills))
- **1 pts** Multiplies by Gratuity (Added to Applies gratuity (Service,Bills))
- **0.5 pts** Multiplies by Gratuity/100 (Added to Applies gratuity (Service,Bills))
- **0.5 pts** Multiplies by 1.Gratuity(Added to Applies gratuity (Service,Bills))
- **0.5 pts** ADD Gratuity
- **0.5 pts** SUM(Dishprice, Gratuity) - SUM() can only take one input parameter
- **0.5 pts** Use percent(Gratuity) instead of Gratuity/100
- **1 pts** Use Multiply keyword after FROM instead of SELECT TO sum up SUM(Dishprice) and Gratuity
+ **0 pts** Select sum(Dishprice) as S; Select Gratuity as G; Set @m =S*G; SELECT @m+S as Money
+ **0 pts** (Select sum(Dishprice) AS TotalDish From...)as A; (Select Gratuity AS Gra)as B;Select A.TotalDish * B.Gra from A Natural JOIN T;
+ **0 pts** Select sum(Dishprice)* Gratuity as temp from (CustomersInParty,Order, Dishes, Bills, Orders); Select Dishprice + temp from Dish where partyID LIKE @p
+ **0 pts** @COST = Select sum(Dishprice) From...; @Gratuity = Select Gratuity FROM .... ; @Money = @Cost(1+@Gratuity)
- **0.5 pts** Use Multiply(a,b) to compute a*b
+ **0 pts** return sum(dishprice) + (sum(dishprice)* tip)
+ **0 pts** ----------Deduction ----------
+ **2 pts** Multiply the results of two seperate select statements
- **1 pts** Retrieve SUM(dishprice) and Gratuityfrom (Orders, Dishes, Services)-- Not enough tables
- **1 pts** Retrieve sum of dish prices for each party, but miss GROUP BY
- **1 pts** get union of sum(dishprices) and gratuity
- **0.5 pts** should use 'CREATE TABLE new_tbl AS SELECT * FROM orig_tbl; ' to create temporary table.

+ **0 pts** ----------Some cases----------

+ **0 pts** JOIN (Orders, Dishes, CustomersInParty, Service, Bills, Waiters)

+ **0 pts** Only JOIN (Orders, Dishes, CustomersInParty, Service, Bills)

+ **0 pts** SELECT SUM(Dishprice) + SUM(Gratuity) From Dishes, Bills

+ **0 pts** No answer

QUESTION 20

20 Extra Credit 2 / 0

✓ **+ 1 pts** **Definition : A piece of data is atomic if it can not be divided into one or more significant pieces of data.**

✓ **+ 1 pts** **Correct Example**

+ **0 pts** Wrong Definition

+ **0 pts** Incorrect Example

+ **0 pts** Blank

+ **0 pts** ---------******Question 2*******----------

+ **1 pts** Defintion - An operation is atomic if a partial execution could leave the database in an invalid state.

+ **1 pts** Correct Example

✓ **+ 0 pts** **Wrong Definition**

+ **0 pts** Incorrect Example

✓ **+ 0 pts** **Example matches definition (though wrong)**

+ **0 pts** Blank

CS 480
Database Systems
Spring 2019

Midterm #2: Thursday, March 21st

Name: Christian Dominguez **LAST 4 DIGITS OF ID:** 2330

Short Answer (10 points)

1) There are 4 primary operations that we can perform on the data within a table in SQL. What are these four operations, and what SQL keywords do we invoke to perform these operations? (2 points)

CREATE -> INSERT
READ -> SELECT
UPDATE -> UPDATE
DELETE -> DELETE

2) What does it mean for a set of attributes to be a primary key? Use definitions specific to the DBMS, not in general terms. (3 point)

- The set of attributes are guaranteed to be unique
- These attributes cannot be NULL

3) What mechanism can a you invoke in SQL which allows you to execute a sequence of statements? (2 points)

The "source" mechanism executes all the statements in a file.

4) What are the three types of responses you can define in a database schema to violations of integrity constraints?
Phrased another way, what automated responses can occur in table T1 when a change or delete happens in table T2, if T1 contains a foreign key referring to the entities of T2? (3 points)

- Cascade
- Set null
- Set default

These are some of the responses that can occur

SQL Schema Definition (30 points)

Write a sequence of SQL statements to do the following:

5) Create a database named Music, and begin using that Database. (3 points)

```
CREATE DATABASE IF NOT EXISTS Music;

USE Music;
```

Each table must contain these elements as a minimum, you may add additional fields as you see fit.
Do not assume that names are unique.
Each table should contain a primary key, and fields with appropriate data types.

6) Write a sequence of SQL statements to build the table
Song, containing a name, length in seconds, and price in dollars. (cents included) (5 points)

```
CREATE TABLE Song (id INT(3) UNSIGNED NOT NULL,
                   name VARCHAR(255) Not NULL,
                   length DOUBLE Not NULL,
                   price NUMERIC(2,2) default NULL,
                   PRIMARY KEY (id));
```

7) Write a sequence of SQL statements to build the table
Artist, containing a name, date of entry into the music industry. (5 points)

```
CREATE TABLE Artist (id INT(3) UNSIGNED NOT NULL,
                     name VARCHAR(255) NOT NULL,
                     date_of_entry INT(4) default NULL,
                     PRIMARY KEY (id));
```

8) Write a sequence of SQL statements to build the table PerformedBy, linking the Song and Artist Tables. (2 points)

```
CREATE TABLE PerformedBy ( song_id INT(3) NOT NULL,
                           song_name VARCHAR(256) NOT NULL,
                           artist_name VARCHAR(255) NOT NULL,
                           PRIMARY KEY (song_id) );
```

9) Add the artist Journey to your Music Database. The group Journey was founded in 1973. (5 points)

```
INSERT INTO Artist (id, name, date_of_entry) VALUES
(100, "Journey", "1973");
```

10) Add the song Open Arms to your Music Database. The song is 3 minutes and 19 seconds long, and is listed at $1.99. This song is performed by Journey, your Database should represent this information in addition to the song details. (5 points)

```
INSERT INTO Song (id, name, length, price) VALUES
(201, "Open Arms", 3.19, 1.99);

INSERT INTO PerFormed By (song_id, song_name, artist_name) VALUES
(201, "Open Arms", "Journey");
```

11) Write a query to return a table containing data about every artist. The data should be in two columns, the artist's name in a column named artist_name and the number of songs released by that artist in a column named num_songs. If an artist has no songs in the database, return 0 for the num_songs attribute for that row. (5 points)

```
SELECT Artist.name AS artist_name, COUNT(*) AS num_songs
FROM PerFormedBy;
```

SQL queries (60 points)

12) What is the result of the following query based on the Database SimpleDB tables? Show the exact answer; include attribute names. (5 points)

```
SELECT      Table1.ID, RecordNum, Table1.Description
FROM        Table1
INNER JOIN Table2 ON Table1.ID = Table2.ID
ORDER BY    Table1.ID ASC;
```

Table1.ID	RecordNum	Table1.Description
10	1	ABC
21	3	GHI

13) What is the result of the following query based on the Database SimpleDB tables? Show the exact answer; include attribute names. (5 points)

```
SELECT      Table1.ID, Table1.Description, Table2.Description
FROM        Table1
NATURAL JOIN Table2
ORDER BY    Table2.Description DESC;
```

Table1.ID	Table1.Description	Table2.Description
21	GHI	CCC
10	ABC	AAA

14) I What is the result of the following query based on the Database Restaurant tables? Show the exact answer; include attribute names. (5 points)

```
SELECT      PartyID, COUNT(*) AS PartySize
FROM        CustomersInParty
GROUP BY    PartyID
ORDER BY    PartySize DESC;
```

PartyID	PartySize
1	2
2	1

For this page use the Restaurant Schema defined on the attached worksheet. These queries should be written relative to the schema defined, not the specific data that is shown as a sample.

15) Suppose the SQL variable @C contains a customer name. Write a SQL query to retrieve the names of the dishes ordered by customer @C. Dishes should be ordered alphabetically (A-Z) by Type, then from most expensive to least expensive. (10 points)

```
SELECT DishName FROM Dishes
INNER JOIN Orders ON
Dishes.DishID = Orders.DishID
INNER JOIN Customers ON
Orders.CustomerID = Customers.CustomerID
WHERE Customers.CustomerName = @C
ORDER BY Dishes.DishType ASC, Dishes.DishPrice DESC;
```

16) How many times has each dish type been ordered? Write a SQL query to retrieve the name of each dish type (Appetizer, Main, Dessert in the data listed), alongside the total number of times each dish type has been ordered. Dish types that have not been ordered should appear with 0 (or NULL if you prefer) for the number of times that they have been ordered. Dish types should be listed at most once, from the most ordered dish type to the least ordered dish type. (10 points)

```
SELECT DISTINCT DishType, COUNT(Orders.DishID) AS num_times FROM Dishes
LEFT OUTER JOIN Orders On
Dishes.DishID = Orders.DishID
GROUP BY num_times
ORDER BY num_times DESC;
```

17) Suppose that the table Service only contains data for one day.
Suppose the SQL variable @W contains a waiter name.
Write a SQL query to retrieve the tables waiter @W waited upon that day. (5 points)

```
SELECT TableID FROM Service
RIGHT OUTER JOIN Waiters ON
Service.EmployeeID = Waiters.EmployeeID
WHERE WaiterName = @W;
```

18) Suppose the SQL variable @T contains a tableID.
Write a SQL query to retrieve the sum of the prices of dishes delivered to that table. (Hint: Question 15) (10 points)

```
SELECT SUM(DishPrice) FROM Dishes
INNER JOIN Orders ON Dishes.DishID = Orders.DishID
INNER JOIN CustomersInParty ON Orders.CustomerID = CustomersInParty.CustomerID
INNER JOIN Service ON CustomersInParty.PartyID = Service.PartyID
INNER JOIN Tables ON Service.TableID = Tables.TableID
WHERE Tables.TableID = @T;
```

19) Suppose the SQL variable @P contains a partyID.
Write a SQL query calculating the amount paid by the party @P. This value is calculated from the prices of the dishes ordered by the members of that party, multiplied to add the percentage value of the gratuity field from the Bills table on the bill assigned to that party. (Hint: separate the pieces of this calculation into multiple queries, use scratch paper to write out the pieces then assemble them with joins). The output should contain a single row table composed of a single column with the amount paid labelled Money. (10 points)

```
SELECT SUM(DishPrice) FROM Dishes
INNER JOIN Orders ON Dishes.DishID = Orders.DishID
INNER JOIN CustomersInParty ON Orders.CustomerID = CustomersInParty.CustomerID
WHERE PartyID = @P

- Multiplied By 2
SELECT SUM(Gratuity) FROM Bills
INNER JOIN Service ON Bills.BillID = Service.BillID
INNER JOIN CustomersInParty ON Service.PartyID = CustomersInParty.PartyID
WHERE CustomersInParty.PartyID = @P;
```

EXTRA CREDIT: You can earn extra credit of 4 points by answering the following two questions:

1. What does it mean for a piece of data to be atomic? Give an example of a piece of data that is not atomic.

An atomic piece of data is indivisible, it cannot be reduced further.

An example of data that is not atomic would be an equation or formula.

2. What does it mean for an operation to be atomic? Give an example of something that can go wrong with your database when an operation that is normally atomic is not performed atomically.

For an operation to be atomic it needs to be done efficiently. If not done efficiently, it will be costly in several regards.

Database **Restaurant**

Customers

CustomerID	CustomerName
1	Wilma
2	Fred
3	Barney

Waiters

EmployeeID	WaiterName
1	Greg
2	Amy
3	Brian

Dishes

DishID	DishName	DishPrice	DishType
1	Ham	16	Main
2	Eggs	13	Main
3	Spam	5	Main
4	Mozzarella Sticks	11	Appetizer
5	Pudding	7	Dessert

Service

PartyID	TableID	EmployeeID	BillID
2	3	3	2
1	1	2	1

Tables

TableID	Capacity	Type	Window
1	6	Booth	True
2	1	Bar	False
3	4	Table	False

Bills

BillID	Gratuity	PaymentMethod
1	20	Check
2	15	Cash

Orders

CustomerID	DishID
1	1
1	3
2	2
2	5
3	4

CustomersInParty

PartyID	CustomerID
1	1
1	2
2	3

Database SimpleDB

Table1

ID	Description
10	ABC
35	DEF
21	GHI

Table 2

RecordNum	ID	Description
1	10	AAA
2	55	BBB
3	21	CCC
4	10	DDD
5	55	EEE

Page from Restaurant Menu

FOOD EATERY

Appetizers

Mozzerella Sticks Cheese wrapped in fried bread	11.00
Nachos Fried cheese for dried up tortillas	9.95
Apple Rumored to have doctor repelling properties	10.50
Pineapple Not to be confused with Apple	10.70

Main Dishes

Ham Fresh off the hog	16.00
Eggs Fried with delicious potential	13.00
Spam It's not ham, but some people are totally fine	5.00
Wall Chicken Whip the wall for a secret	10.70

Dessert

Pudding It's jiggy	7.00
Fried Ice Cream Fire and Ice	16.70
After dinner mint It's wafer thin	3.75
Flan Superior to pudding	13.50

DDL - Data Definition Language: Schema for each relation, domain of values associated with each attribute, integrity (not null, primary) constraints

Char(n) - Fixed, varchar(n) - max length, text - really long, int(n), tinyint(n), numeric(p,d) - p: left of decimal, d: right, double, float(n) - n digits, date: yyyy-m-dd, time: hh:mm:ss, time stamp - combination of previous two, interval - period of time

CRUD: CREATE, READ, UPDATE, DELETE ->
Data Manipulation Language (DML), INSERT, SELECT, UPDATE, DELETE

USE "database", SHOW DATABASES, SHOW TABLES, SHOW COLUMNS FROM "table", LIMIT "#"

<> = not equal

CREATE, DROP, SHOW, ALTER

CREATE DATABASE IF NOT EXISTS db_name;

```
CREATE TABLE comments
(id INT(3) UNSIGNED NOT NULL
AUTO_INCREMENT, body TEXT default
NULL, user_id INT(7) UNSIGNED NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY(user_id) REFERENCES
user(ID)) AUTO_INCREMENT = 1;
```

```
UPDATE trips, SET trip_date_time_end
= '2015-04-15 23:15:33', WHERE id = 7;
```

Omitting WHERE changes all the table values.

```
ALTER TABLE drivers CHANGE old_col
new_col TINYINT(1)
```

```
ALTER TABLE tbl_name ADD col_name
INT(10) NOT NULL
```

RENAME TABLE cars TO vehicles

```
INSERT INTO tbl_name (col1, col2, col3)
VALUES (val1, val2, val3).
```

- If adding values for all columns, no need to specify columns

```
DELETE FROM table_name,
WHERE col_name = something
```

- OMITTING WHERE will delete entire tables

ALTER TABLE tbl_name DROP col

DROP TABLE tbl_name

TRUNCATE TABLE tblname
- Deletes DATA, not table

DROP DATABASE db

```
SELECT * FROM classrooms
JOIN courses ON
classrooms.course_id = courses.id;
```

```
room_num INT(4) UNSIGNED NOT NULL UNIQUE KEY
```

INNER JOIN - Natural Join, No NULL (Full Outer)

LEFT (OUTER) JOIN - Everything from Left Table and common things from the right table

RIGHT OUTER JOIN - Everything FROM Right Table and common things from the left

```
ALTER TABLE students
ADD CONSTRAINT FOREIGN KEY (classroom_id)
RERENCES classrooms (id)
```

```
SELECT MIN(price) AS min_price FROM products
```

MIN, MAX, SUM, COUNT, AVG

GROUP BY groups the items by requested column and lists them in numerical / alphabetical order.

WHERE cannot be used on aliases, instead use having ↓

```
SELECT products.id, products.name,
AVG(stars) AS avg_stars
FROM products
JOIN reviews ON
products.id = reviews.product_id
WHERE products.id < 6
GROUP BY product.name HAVING avg_stars < 4
```

```
CRETE VIEW name AS
SELECT ID, name, dept_name
FROM instructor
```

Primary key - set of attributes is guaranteed to be unique.
- Cannot be NULL

Transaction - user invokes this to allow them to execute a sequence of statements without worrying about the values in the database changing between statements

Responses to Integrity constraints:
Cascade
set null
set default

Christian Dominguez