 

SECTION A: Database Theory/‘How & Why’ (5 questions | 10 points | 50 minutes)

1. Describe the differences between full, differential and transaction log back-ups and provide an example of a disaster recovery strategy that uses all three types of backups.
2. Describe the differences between Online Transaction Processing (OLTP) databases and those that are supporting Data Warehousing or Online Analytical Processing (OLAP).
3. Explain what is meant by 'Fault-Tolerance' and identify three system component examples.
4. Describe 5 different SQL commands that are considered ‘control of flow’ language.
5. Define the different data warehouse design structures: star schema, snowflake schema, ‘star flake’ schema, fact table, dimension table in addition to a ‘measure’.
6. Compare relational databases with NoSQL databases; what are the primary differences and when is each the preferred technology?
7. Explain what ranking functions are and how they are beneficial to writing complex queries.
8. Compare high availability versus scalability within database environments; when and how will each be implemented?
9. Explain the proper use of error-handling; when, where and how will this be completed?
10. Compare the proper use of temporary objects as explained in class.
11. Explain the differences between a Data Warehouse and a Data Mart.
12. Compare asynchronous communications versus synchronous; which is preferred to reduce risk of data loss?

SECTION B: SQL code: RELATIONAL\_AIR ERD (5 questions | 15 points | 75 minutes)

* Create at least one stored procedure that takes in several parameters of friendly names and INSERTs into multiple tables in an explicit transaction with proper error-handling

/\*

Write the SQL to populate a new row in FLIGHT\_MEAL with the following conditions:

1) Nested stored procedures to GetIDs for FlightID and MealID (these the FKs)

2) Outer stored procedure to execute the whole thing

3) explicit transaction --> we've done this many times

4) proper error-handling --> terminate trx if any variable IS NULL

7:09 --> finish in 12 minutes!! --> 12:01

\*/

-- create the nested 'GetID' stored procedures

CREATE PROCEDURE uspGetFlightID

@Flighty varchar(50),

@FlightID INT OUTPUT

AS

SET @FlightID = (SELECT FlightID

FROM tblFLIGHT

WHERE FlightName = @Flighty)

GO

CREATE PROCEDURE uspGetMealID

@Mealy varchar(50),

@MealID INT OUTPUT

AS

SET @MealID = (SELECT MealID

FROM tblMEAL

WHERE MealName = @Mealy)

GO

--CREATE the outer stored procedure to manage the single row insert into tblFLIGHT\_MEAL

CREATE PROCEDURE uspInsertFlightMeal

@Meal\_name1 varchar(50),

@Flight\_name1 varchar(50),

@TimeServed DateTime

AS

-- quick check for NULL values

IF @Meal\_name1 IS NULL or @Flight\_name1 IS NULL

BEGIN

PRINT ('Hey...the parameters are required')

RAISERROR ('Parameters may not be NULL', 11,1)

RETURN

END

DECLARE @F\_ID INT, @M\_ID INT

EXEC uspGetFlightID

@Flighty = @Flight\_name1,

@FlightID = @F\_ID OUTPUT

IF @F\_ID IS NULL

BEGIN

RAISERROR ('F\_ID cannot be NULL', 11,1)

RETURN

END

EXEC uspGETMealID

@Mealy = @Meal\_name1,

@MealID = @M\_ID OUTPUT

IF @M\_ID IS NULL

BEGIN

RAISERROR ('@M\_ID cannot be NULL', 11,1)

RETURN

END

BEGIN TRAN T1

INSERT INTO tblFIGHT\_MEAL (FlightID, MealID, TimeServed)

VALUES (@F\_ID, @M\_ID, @TimeServed)

IF @@ERROR <> 0

ROLLBACK TRAN T1

ELSE

COMMIT TRAN T1

* Create at least one business rule or computed column leveraging a function

/\*

Write the SQL to enforce the following business rule:

"Customers must be at least 24 take a flight into Dallas, Texas

during December."

-- user-defined function

-- code for the violation

-- alter table to anchor check constraint

-- no parameter required

-- don't forget the schema definition 'dbo' in front of object

~ 6 minutes? 7:26 begin time -- > 7:32

\*/

GO

CREATE FUNCTION fn\_NoOneUnder24DallasDec()

RETURNS INT

AS

BEGIN

DECLARE @RET INT = 0

IF EXISTS (SELECT \*

FROM tblCUSTOMER C

JOIN tblBOOKING B ON C.CustomerID = B.CustomerID

JOIN tblFLIGHT F ON B.FlightID = F.FlightID

JOIN tblAIRPORT A ON F.ArrivalAirportID = A.AirportID

JOIN tblCITY CY ON A.CityID = CY.CityID

WHERE C.BirthDate > (SELECT GetDate() - (365.25 \*24))

AND CY.CityName = 'Dallas'

AND MONTH(B.BookDateTime) = 12)

BEGIN

SET @RET = 1

END

RETURN @RET

END

GO

ALTER TABLE tblBOOKING WITH NOCHECK

ADD CONSTRAINT CK\_NoYoungAdultsDallas

CHECK (dbo.fn\_NoOneUnder24DallasDec()=0)

GO

/\*

Write the SQL to compute the following column:

'TotalFareDollarsPastYear'.

1) What is our parameter?

2) Where does this new column live?

3) don't forget to anchor this column on a column --> and pass the PK!!

~5 minutes? 7:38 --> 7:44

\*/

CREATE FUNCTION fn\_TotalDollarsFares\_Past12Months(@PKID INT)

RETURNS NUMERIC(10,2)

AS

BEGIN

DECLARE @RET NUMERIC(10,2)

SET @RET = (SELECT SUM(Fare)

FROM tblCUSTOMER C

JOIN tblBOOKING B ON C.CustomerID = B.CustomerID

WHERE B.BookDateTime >= (SELECT GetDate() - 366)

AND C.CustomerID = @PKID

)

RETURN @RET

END

GO

ALTER TABLE tblCUSTOMER

ADD TotalFareDollarsPastYear

AS (dbo.fn\_TotalDollarsFares\_Past12Months(CustomerID) )

* Create at least one stored procedure that calls another stored procedure (‘nested’ stored procedures) leveraging OUTPUT parameter (included above)
* Create at least one query that leverages subqueries, CTE, or table variable to answer a complex business question.
* Create at least one complex view (multiple JOINs, GROUP BY, HAVING, CASE, RANKING functions)

/\*

• Create at least one query that leverages subqueries, CTE, or

table variable to answer a complex business question.

Write the SQL to answer the following question using a CTE:

"Which customers have spent more than $10,000 on fares for flights into

South America in the past 5 years

who have also

been involved in more than 3 incidents of inebriated behavior in the

past four years?"

~10 minutes 7:49 --> 8:06

\*/

WITH CTE\_10\_GrandSouthAmerica (CustomerID, Fname, Lastname, FareDollars\_S\_America\_5years)

AS

(SELECT C.CustomerID, C.Fname, C.LastName, SUM(Fare) AS TotalFares

FROM tblCUSTOMER C

JOIN tblBOOKING B ON C.CustomerID = B.CustomerID

JOIN tblFLIGHT F ON B.FlightID = F.FlightID

JOIN tblAIRPORT A ON F.ArrivalAirportID = A.AirportID

JOIN tblCITY CY ON A.CityID = CY.CityID

JOIN tblCOUNTRY CTY ON CY.CountryID = CTY.CountryID

JOIN tblREGION R ON CTY.RegionID = R.RegionID

WHERE F.ScheduledDeparture >= (SELECT GetDate() - (365.25 \* 5))

AND R.RegionName = 'South America'

GROUP BY C.CustomerID

HAVING SUM(Fare) >= 10000)

,

CTE\_Number\_Inebriations\_FourYears (CustomerID, NumberOfInebriated\_Incidents)

AS

(SELECT C2.CustomerID, COUNT(\*) AS NumberIncidentsDrunk

FROM tblCUSTOMER C2

JOIN tblBOOKING B2 ON C2.CustomerID = B2.CustomerID

JOIN tblINCIDENT I2 ON B2.BookingID = I2.BookingID

JOIN tblINCIDENT\_TYPE IT2 ON I2.IncidentTypeID = IT2.IncidentTypeID

WHERE IT2.IncidentTypeName = 'Inebriated Behavior'

AND B2.ScheduledDepart > (SELECT GetDate() - (365.25 \* 4))

GROUP BY C2.CustomerID

HAVING COUNT(\*) >= 3)

SELECT A.CustomerID, B.Fname, B.LastName, NumberOfInebriated\_Incidents, FareDollars\_S\_America\_5years

FROM CTE\_Number\_Inebriations\_FourYears A

JOIN CTE\_10\_GrandSouthAmerica B ON A.CustomerID = B.CustomerID

/\*

--Complex view that uses CASE statement make the query a VIEW!

TotalFees\_Bookings with the following in the past 3 years:

--> greater than $200 = 'expensive yo'

--> greater than $100 but less than $200 = 'slightly expensive yo'

--> greater than $ 50 but less than $100 = 'pretty good'

--> Less than $ 50 but greater than $ 10 = 'cheap'

--> 'nominal'

~6 minutes 8:18 --> 8:27

\*/

CREATE VIEW vwBookingFeeCount

AS

SELECT (

CASE

WHEN TotalFee > 200

THEN 'Expensive yo'

WHEN TotalFee BETWEEN 100 and 200

THEN 'Slightly Expensive yo'

WHEN TotalFee BETWEEN 50 AND 99.99

THEN 'Pretty Good'

WHEN TotalFee BETWEEN 10 AND 49.99

THEN 'Cheap'

ELSE 'nominal'

END) AS Fare\_FeeLable, COUNT(\*) AS NumberOfBookings

FROM

(SELECT B.BookingID, SUM(Fee) AS TotalFee

FROM tblBOOKING B

JOIN tblBOOKING\_FEE BF ON B.BookingID = BF.BookingID

JOIN tblFEE F ON BF.FeeID = F.FeeID

WHERE B.BookDateTime > (SELECT GetDate() - (365.25 \*3))

GROUP BY B.BookingID) AS subquery

GROUP BY --> CASE statement!!

/\*

Write the SQL to answer the following:

'Return all employees divided evenly in 5 buckets (hint: this is NTILE)

based the the totalnumber of flights in the role of 'Attendant Crew' in

the past ten years'

~7 minutes 8:39 --> 8:45

\*/

SELECT E.EmployeeID, E.EmployeeFName, E.EmployeeLname, NTILE(5) OVER (ORDER BY COUNT(\*))

FROM tblEMPLOYEE E

JOIN tblFLIGHT\_EMPLOYEE FE ON E.EmployeeID = FE.EmployeeID

JOIN tblROLE R ON FE.RoleID = R.RoleID

JOIN tblFLIGHT F ON FE.FlightID = F.FlightID

WHERE R.RoleName = 'Attendant Crew'

AND F.ScheduledDepart > (SELECT GetDate() - (365.25 \* 10))

GROUP BY E.EmployeeID, E.EmployeeFName, E.EmployeeLname

