

Oliver Bird Database and Application Development

by Ollie Bird

FILE	18246_OLLIE_BIRD_OLIVER_BIRD_DATABASE_AND_APPLICATION_DEVELOPMENT_349706_447829845.DOCX (2.08M)		
TIME SUBMITTED	04-MAY-2021 11:46AM (UTC+0100)	WORD COUNT	2135
SUBMISSION ID	151558653	CHARACTER COUNT	11067

Database Application Development – 1910716

Oliver Bird

Apprentice Software Engineer – BT , Data , Data Services

1910716@student.uwtsd.ac.uk

ABSTRACT

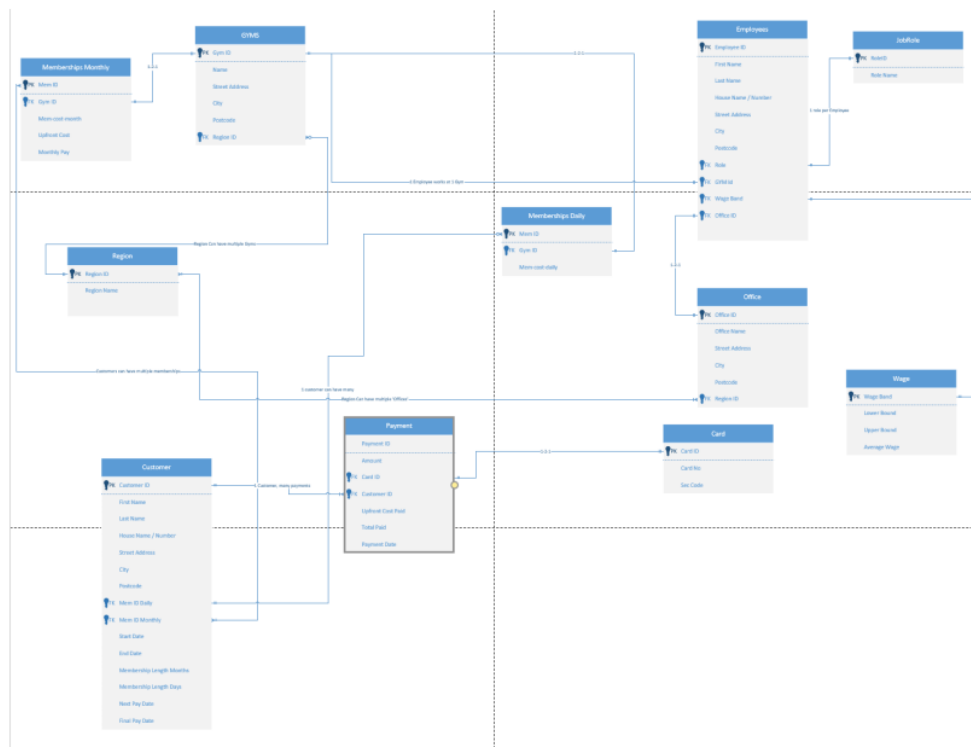
Two database solutions will be presented to cover the needs of Shape Up Gyms. An analysis and test of the database structure will be given prior to an introduction of performance improvements via table indexing and a described distributed database solution. This solution will be broken down by region.

Table of Contents

1. Entity Relationship Diagram – Visio
2. Database Implementation and Testing
 - a. Database Creation
 - i. Table Creation
 - ii. Data Insert
 - b. Referential Integrity
 - i. Integrity between customer and card
 - ii. Integrity between customer and membership
 - iii. Integrity between Payment and customer to prove the above
 - iv. Extreme Integrity between payment cards used vs employees at a given gym.
 - v. Employees by Gym
 - vi. Employee Average Wage at their Gym
 - vii. All customers processed.
 - viii. All current customers
 - ix. Employees by Job Role
 - c. Transactions / Constraints
 - i. Duplicate Id
 - ii. Incorrect Postcode
 - iii. Non-Existent Membership ID
 - iv. Update existing customer with a new membership.
 - v. New Customer
 - vi. New Gym + Customer
 - vii. New Employee
 - viii. Bad Card Number
3. Indexing
 - a. Queries
 - b. Indexes
4. Distributed Database Model

1) Entity Relationship Diagram (ERD)– Visio

An ERD diagram has been created in MS-Visio to display the table structure with their associated references.



2) Database Implementation and Testing

This section will include a brief demonstration of database creation with its original data and parameters. No improvements to the database with regards to performance are made at this stage. Constraints and key references are added during table creation. Cascades for important references have been added to reduce the chance of orphaned rows.

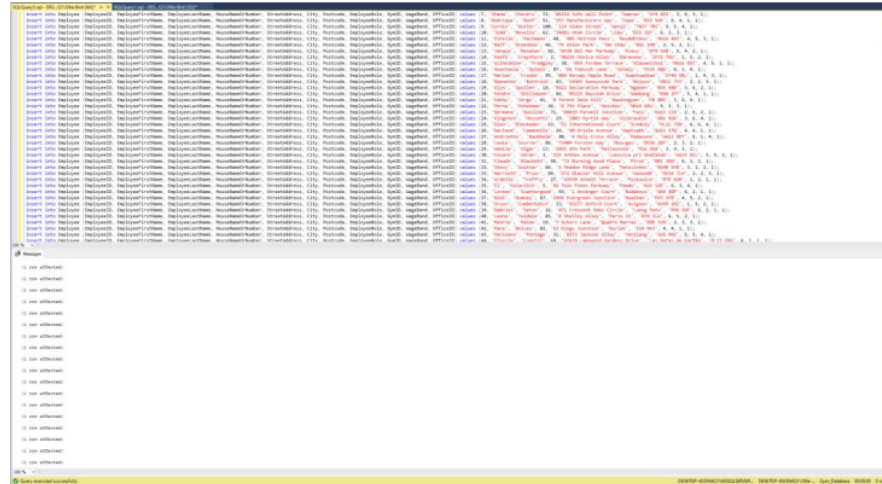
- a) Database Creation
 - i) Table Creation

The create table insert is run directly with no errors as below. The entire script has been omitted and contains a database wipe at the onset to ensure an ‘as-real’ deployment.

[illegible]

ii) Data Insert

The generated data [1] [2] is inserted into the database in compliance with all referential and value-based constraints with realistic data for the required scenario of Shape Up Gyms.

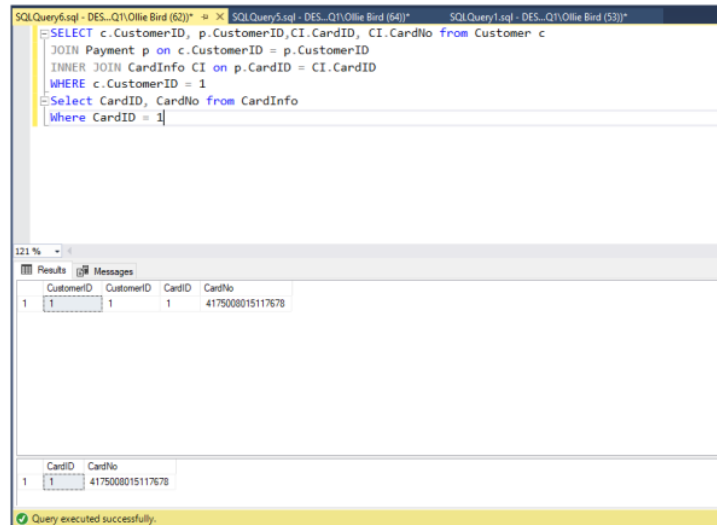


b) Referential Integrity

A number of queries both realistic and complex have been designed to test the referential integrity between tables.

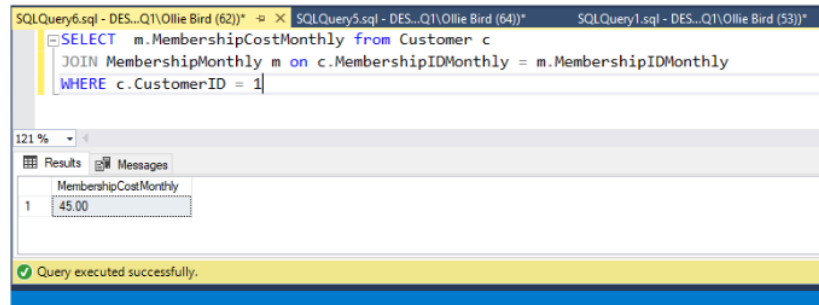
i) Integrity between customer and card

For a given customer, display the correct card information via a join through payments. Two selections are shown to display the matches individually and joined.



ii) Integrity between customer and membership

For a given customer, select the value of their current monthly membership cost via a join between Customer and MembershipMonthly.



The screenshot shows a SQL query window with the following text:

```
SQLQuery6.sql - DES...Q1\Ollie Bird (62)))* - X SQLQuery5.sql - DES...Q1\Ollie Bird (64)))* SQLQuery1.sql - DES...Q1\Ollie Bird (53)))*
SELECT m.MembershipCostMonthly from Customer c
JOIN MembershipMonthly m on c.MembershipIDMonthly = m.MembershipIDMonthly
WHERE c.CustomerID = 1
```

Below the query, the 'Results' tab is active, displaying a table with one row:

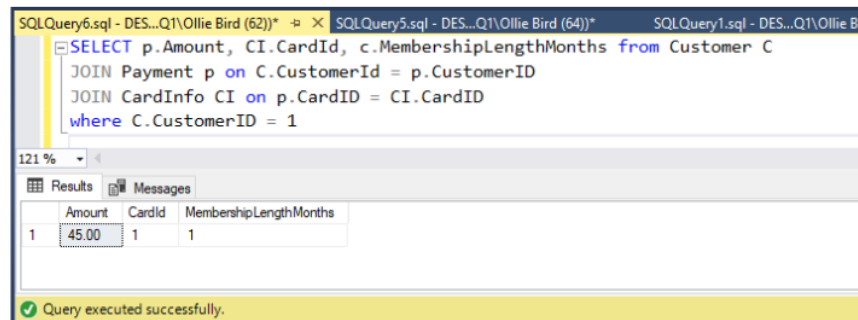
MembershipCostMonthly
45.00

A status bar at the bottom indicates 'Query executed successfully.'

iii) Integrity between Payment and customer to prove the above

This query is designed to test both that the amount paid by the customer matches the value of the membership for its given duration.

As shown below, the paid amount matches the required amount shown above.



The screenshot shows a SQL query window with the following text:

```
SQLQuery6.sql - DES...Q1\Ollie Bird (62)))* - X SQLQuery5.sql - DES...Q1\Ollie Bird (64)))* SQLQuery1.sql - DES...Q1\Ollie Bird (53)))*
SELECT p.Amount, CI.CardId, c.MembershipLengthMonths from Customer C
JOIN Payment p on C.CustomerId = p.CustomerID
JOIN CardInfo CI on p.CardID = CI.CardID
where C.CustomerID = 1
```

Below the query, the 'Results' tab is active, displaying a table with one row:

	Amount	CardId	MembershipLengthMonths
1	45.00	1	1

A status bar at the bottom indicates 'Query executed successfully.'

iv) Extreme Integrity between payment cards used vs employees at a given gym.

This is an extreme integrity check which links almost all tables together via an associated gym. This references a selection of payment cards used at a given gym with the employees that work there. This type of query has no use in reality, however, demonstrates the integrity correctly. A portion of the return has been cut off due to size constraints.

SQLQuery6.sql - DES...Q1\Ollie Bird (62))" * SQLQuery5.sql - DES...Q1\Ollie Bird (64))" SQLQuery1.sql - DES...Q1\Ollie Bird (53))"

```

SELECT CI.CardNo, e.EmployeeID, e.EmployeeFirstName, e.EmployeeLastName, GymName from CardInfo CI
JOIN Payment p on CI.CardID = p.CardID
Join Customer c on p.CustomerID = c.CustomerID
JOIN MembershipMonthly mm on c.MembershipIDMonthly = mm.MembershipIDMonthly
JOIN Gym g on mm.GymID = g.GymID
JOIN Employee e on g.GymID = e.GymID
where g.GymID = 1

```

121 %

Results Messages

	CardNo	EmployeeID	EmployeeFirstName	EmployeeLastName	GymName
1	4913530028448060	4	Corie	Gorgietto	Shape Up Cardff South
2	4913530028448060	16	Anastasia	Splain	Shape Up Cardff South
3	4913530028448060	27	Andriette	Raubheim	Shape Up Cardff South
4	4913530028448060	31	Claude	Blackett	Shape Up Cardff South
5	4913530028448060	32	Stevy	Souttar	Shape Up Cardff South
6	4913530028448060	44	Rossie	Coggil	Shape Up Cardff South
7	4913530028448060	45	Herman	Napleton	Shape Up Cardff South
8	417500362875456	4	Corie	Gorgietto	Shape Up Cardff South
9	417500362875456	16	Anastasia	Splain	Shape Up Cardff South

Query executed successfully.

v) Employees by Gym

SQLQuery4.sql - DES...Q1\Ollie Bird (62))" * SQLQuery3.sql - DES...Q1\Ollie Bird (64))" SQLQuery1.sql - DES...Q1\Ollie Bird (53))"

```

SELECT e.EmployeeFirstName, e.EmployeeLastName, g.GymName, g.StreetAddress, g.Postcode from Employee e
JOIN Gym g on e.GymID = g.GymID
Order By e.GymID

```

121 %

Results Messages

	EmployeeFirstName	EmployeeLastName	GymName	StreetAddress	Postcode
1	Corie	Gorgietto	Shape Up Cardff South	07 International Circle	CF47 5JP
2	Anastasia	Splain	Shape Up Cardff South	07 International Circle	CF47 5JP
3	Andriette	Raubheim	Shape Up Cardff South	07 International Circle	CF47 5JP
4	Claude	Blackett	Shape Up Cardff South	07 International Circle	CF47 5JP
5	Stevy	Souttar	Shape Up Cardff South	07 International Circle	CF47 5JP
6	Rossie	Coggil	Shape Up Cardff South	07 International Circle	CF47 5JP
7	Herman	Napleton	Shape Up Cardff South	07 International Circle	CF47 5JP
8	Gabriel	Saton	Shape Up Cardff North	6 Harover Pass	KA10 6PN
9	Hanlett	Pfyr	Shape Up Cardff North	6 Harover Pass	KA10 6PN

Query executed successfully.

vi) Employee Average Wage at their Gym

SQLQuery4.sql - DES...Q1\Ollie Bird (62))" * SQLQuery3.sql - DES...Q1\Ollie Bird (64))" SQLQuery1.sql - DES...Q1\Ollie Bird (53))"

```

SELECT e.EmployeeFirstName, e.EmployeeLastName, g.GymName, g.StreetAddress, g.Postcode, w.AvgWageValue from Employee e
JOIN Wage w on e.GymID = g.GymID
Order By w.AvgWageValue

```

121 %

Results Messages

	EmployeeFirstName	EmployeeLastName	GymName	StreetAddress	Postcode	AvgWageValue
1	Corie	Gorgietto	Shape Up Cardff South	07 International Circle	CF47 5JP	15000.00
2	Tade	Henside	Shape Up Cardff Central	3 Fackens Alley	CY34 6JL	15000.00
3	Rodrique	Ruf	Shape Up Harport	750 Blackbell Pass	T42 7FA	15000.00
4	Wilshire	Redgley	Shape Up Bridgend	729 Lamp Drive	CF11 1JZ	15000.00
5	Henside	Shildner	Shape Up Harport	750 Blackbell Pass	T42 7FA	15000.00
6	Naland	Comenide	Shape Up Harport	750 Blackbell Pass	T42 7FA	15000.00
7	Arvida	Taffy	Shape Up Cardff North	6 Harover Pass	KA10 6PN	15000.00
8	Lowen	Scattergood	Shape Up Cardff North	6 Harover Pass	KA10 6PN	15000.00
9	Gabriel	Saton	Shape Up Cardff North	6 Harover Pass	KA10 6PN	15000.00

Query executed successfully.

vii) All customers processed.

This query unions both membership type tables over all gym IDs to show all customers who have been through the gym, regardless of membership length.

```
SQLQuery6.sql - DES-Q1\Ollie Bird (62)  SQLQuery5.sql - DES-Q1\Ollie Bird (64)  SQLQuery1.sql - DES-Q1\Ollie Bird (53)
SELECT c.CustomerFirstName, c.CustomerLastName, g.GymName, c.customerID, c.MembershipIDMonthly, c.membershipIDDaily from Customer c
JOIN membershipMonthly m on c.membershipIDMonthly = m.membershipIDMonthly
JOIN Gym g on m.GymID = g.GymID

UNION

SELECT c.CustomerFirstName, c.CustomerLastName, g.GymName, c.customerID, c.MembershipIDMonthly, c.membershipIDDaily from Customer c
JOIN membershipIDaily mD on c.membershipIDDaily = mD.membershipIDDaily
JOIN Gym g on mD.GymID = g.GymID
```

121 %

Results Messages

	CustomerFirstName	CustomerLastName	GymName	customerID	MembershipIDMonthly	membershipIDDaily
1	Ab	Goings	Shape Up Newport	110	4	NULL
2	Ab	Stearns	Shape Up Seawasa Bay	71	NULL	7
3	Adam	Stevenson	Shape Up Port Talbot	80	10	NULL
4	Adia	Garnan	Shape Up Seawasa Bay	27	7	NULL
5	Alex	Tebbett	Shape Up Port Talbot	151	10	NULL
6	Alast	Ryle	Shape Up Cardiff Central	75	3	NULL
7	Alvarado	Maksum	Shape Up Seawasa Bay	146	8	NULL
8	Alan	Smith	Shape Up Cardiff Central	62	3	NULL
9	Anabelle	Gourle	Shape Up Bridgend	63	5	NULL

Query executed successfully. DESKTOP-4GONKQ1\MSQLS

viii) All current customers

This will show all current customers, selected by their end of contract date being greater than the current date. It returns no values for daily memberships as at the time of writing no customers posses a day pass at any gym.

```
SQLQuery6.sql - DES-Q1\Ollie Bird (62)  SQLQuery5.sql - DES-Q1\Ollie Bird (64)  SQLQuery1.sql - DES-Q1\Ollie Bird (53)
SELECT c.CustomerFirstName, c.CustomerLastName, g.GymName, c.customerID, c.EndDate, FinalPayDate from Customer c
JOIN membershipMonthly m on c.membershipIDMonthly = m.membershipIDMonthly
JOIN Gym g on m.GymID = g.GymID
Where EndDate >= GETDATE()

SELECT c.CustomerFirstName, c.CustomerLastName, g.GymName, c.customerID, c.EndDate, FinalPayDate from Customer c
JOIN membershipIDaily mD on c.membershipIDDaily = mD.membershipIDDaily
JOIN Gym g on mD.GymID = g.GymID
Where EndDate >= GETDATE()
```

121 %

Results Messages

	CustomerFirstName	CustomerLastName	GymName	customerID	EndDate	FinalPayDate
1	Riley	Topper	Shape Up Newport	29	2021-05-25	2021-05-25
2	Zachery	Mather	Shape Up Cardiff North	47	2021-05-16	2021-05-16
3	Toby	Pembrett	Shape Up Bridgend	98	2021-06-08	2021-06-08
4	Jeromy	Tomczakowski	Shape Up Cardiff South	160	2021-05-05	2021-05-05
5	German	Rittmer	Shape Up Cardiff South	177	2021-05-08	2021-05-08

Query executed successfully.

ix) Employees by Job Role

SQL Query 5.sql - DES-Q1\Ole Bed (640) SQL Query 5.sql - DES-Q1\Ole Bed (640) SQL Query 5.sql - DES-Q1\Ole Bed (640)

```
--SELECT e.EmployeeFirstName, e.EmployeeLastName, g.GymName, g.StreetAddress, g.Postcode, j.RoleName from Employee e
JOIN Gym g on e.GymID = g.GymID
Join JobRole j on e.EmployeeRole = j.RoleID
Order By e.EmployeeRole
```

Results Messages

	EmployeeFirstName	EmployeeLastName	GymName	StreetAddress	Postcode	RoleName
1	Helen	Crooke	Shape Up Newcourt	750 Backland Pass	TA2 7PA	Manager
2	Kevle	Craythorn	Shape Up Bridgend	729 Lenny Drive	CF11 7JZ	Manager
3	Arabella	Trefly	Shape Up Cardiff North	6 Hanover Pass	KA10 6PN	Manager
4	Oreon	Cumberbatch	Shape Up Bridgend	729 Lenny Drive	CF11 7JZ	Manager
5	Tina	Beddie	Shape Up Cardiff Central	3 Packers Alley	CV34 6JL	Manager
6	Lynard	Gladium	Shape Up Swansea Bay	589 Cabarg Court	NRS 9PF	Manager
7	Bard	Rodburth	Shape Up Swansea, Port Head	84 Temyon Alley	BN14 8LY	Manager
8	Aley	Batholin	Shape Up Swansea, Gletty	30017 Washington Alley	FY3 7RU	Manager
9	Latis	Vafolomeev	Shape Up Swansea Bay	0496 Eastfawn Park	WC1E 7LE	Manager
10	Loi	Luplen	Shape Up Port Talbot	17052 Hoffman Trail	WS12 4SB	Manager
11	Myah	Culley	Shape Up Swansea Bay	0496 Eastfawn Park	WC1E 7LE	Manager
12	Marvin	Bullery	Shape Up Swansea, Port Head	84 Temyon Alley	BN14 8LY	Financial Analyst
13	Jany	Matyshev	Shape Up Swansea, Gletty	30017 Washington Alley	FY3 7RU	Financial Analyst
14	Kik	MacLeod	Shape Up Swansea Bay	0496 Eastfawn Park	WC1E 7LE	Financial Analyst
15	Henne	Pane	Shape Up Cardiff Central	3 Packers Alley	CV34 6JL	Financial Analyst
16	Carl	Neames	Shape Up Bridgend	729 Lenny Drive	CF11 7JZ	Financial Analyst
17	Givi	Housman	Shape Up Cardiff Central	3 Packers Alley	CV34 6JL	Financial Analyst

Query executed successfully.

c) Transactions / Constraints

Transactions have been designed to test the integrity between tables via value inputs. This will test both the values directly inserted and their referential equivalents. The transactions are dynamic and select current values for ID when inserting records.

i. Duplicate ID

This transaction will attempt to add a new customer however the ID has not been inserted correctly. The transaction will rollback.

```
--BEGIN TRANSACTION NewCustomerAddID
BEGIN TRY
    --Get the last ID in the payments table
    DECLARE @PaymentID AS INT
    SELECT TOP(1) @PaymentID = PaymentID FROM Payment
    ORDER BY PaymentID DESC
    --Get the last Card ID
    DECLARE @CardID AS INT
    SELECT TOP(1) @CardID = CardID FROM CardInfo
    ORDER BY CardID DESC
    INSERT INTO Customer (CustomerID, CustomerFirstName, CustomerLastName, HouseNameOrNumber, StreetAddress, City, Postcode, MembershipIDMonthly, MembershipIDDaily, StartDate, EndDate, FinalPayment)
    values (3, 'Randi', 'Tetter', 45, '58 South Road', 'Neitan', 'CO2 430', 5, null, '11/11/2020', '12/11/2021', '12/11/2021');
    INSERT INTO CardInfo(CardID, CardNo, SecCode)
    values(@CardID + 1, 402085947012345, 397);
    INSERT INTO Payment(PaymentID, Amount, CardID, CustomerID, UpFrontCostPaid) values(@PaymentID + 1, 64.5, (@CardID + 1), 5, 0);
    COMMIT TRANSACTION NewCustomerAddID
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION NewCustomerAddID;
    PRINT('Rolling back');
    THROW;
END CATCH;
```

Results Messages

(1 row affected)

Rolling back

Msg 2527, Level 16, State 1, Line 11
Violation of PRIMARY KEY constraint 'PK_CustomerID'. Cannot insert duplicate key in object 'dbo.Customer'. The duplicate key value is (3).

Completion time: 2021-05-07T16:12:11.0000000+01:00

ii. Incorrect Postcode

This transaction will attempt to add a new customer with an invalid postcode. The maximum length accepted is 9 characters.

```
--BEGIN TRANSACTION NewCustomerAddPost
BEGIN TRY
    DECLARE @CustomerID AS INT
    SELECT TOP(1) @CustomerID = CustomerID FROM Customer
    ORDER BY CustomerID DESC
    --Get the last ID in the payments table
    DECLARE @PaymentID AS INT
    SELECT TOP(1) @PaymentID = PaymentID FROM Payment
    ORDER BY PaymentID DESC
    --Get the last Card ID
    DECLARE @CardID AS INT
    SELECT TOP(1) @CardID = CardID FROM CardInfo
    ORDER BY CardID DESC
    Insert INTO Customer (CustomerID, CustomerFirstName, CustomerLastName, HouseNameOrNumber, StreetAddress, City, Postcode, MembershipIDMonthly, MembershipIDDaily, StartDate, EndDate, FinalPayment)
    values (@CustomerID + 1, 'Randi', 'Tetter', 45, '58 South Road', 'Neitan', 'CO2 4300000', 5, null, '11/11/2020', '12/11/2021', '12/11/2021');
    INSERT INTO CardInfo(CardID, CardNo, SecCode)
    values(@CardID + 1, 402085947012345, 397);
    INSERT INTO Payment(PaymentID, Amount, CardID, CustomerID, UpFrontCostPaid) values(@PaymentID + 1, 64.5, (@CardID + 1), (@CustomerID + 1), 0);
    COMMIT TRANSACTION NewCustomerAddPost
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION NewCustomerAddPost;
    PRINT('Rolling back');
    THROW;
END CATCH;
```

Results Messages

(1 row affected)

Rolling back

Msg 2527, Level 16, State 1, Line 14
Message or priority data would be violated in table 'dbo.Customer-NewCustomer', because 'Postcode'. The message value is 'CO2 43000'.

Completion time: 2021-05-07T16:13:04.2170000+01:00

iii. Non-Existent Membership ID

The below transaction attempts to add a customer with a membership ID which does not exist in the membership table.

```
GO
DECLARE @SQL NVARCHAR(4000) = N''
-- Transaction: NewCustomerMembership
-- BEGIN TRY
-- DECLARE @CustomerID AS INT
-- SELECT TOP(1) @CustomerID = CustomerID FROM Customer
-- ORDER BY CustomerID DESC
-- -- Gets the last ID in the payments table
-- DECLARE @PaymentID AS INT
-- SELECT TOP(1) @PaymentID = PaymentID FROM Payment
-- ORDER BY PaymentID DESC
-- -- Gets the last Card ID
-- DECLARE @CardID AS INT
-- SELECT TOP(1) @CardID = CardID FROM CardInfo
-- ORDER BY CardID DESC
-- Insert INTO Customer (CustomerID, CustomerFirstName, CustomerLastName, HouseNumber, StreetAddress, City, Postcode, MembershipMonthly, MembershipMonthly, StartDate, EndDate, FinalPayDate)
-- values (@CustomerID + 1, 'Randi', 'Tettery', 45, '58 Walth Road', 'Wilton', 'CO2 430', '58', null, '11/23/2008', '1/23/2009', '1/23/2009')
-- Insert INTO CardInfo (CardID, CardNo, SecCode)
-- values (@CardID + 1, 46068047813667, 197)
-- Insert INTO Payment (PaymentID, Amount, CardID, CustomerID, UpfrontCostPaid) values (@PaymentID + 1, 64.5, @CardID + 1, @CustomerID + 1, 0)
-- COMMIT TRANSACTION NewCustomerMembership
-- END TRY
-- BEGIN CATCH
-- ROLLBACK TRANSACTION NewCustomerMembership;
-- PRINT 'Rolling back';
-- THROW;
-- END CATCH
```

Messages

(1 row affected)
Rolling back
Msg 141, Level 16, State 6, Line 14
The INSERT statement conflicted with the FOREIGN KEY constraint "FK_Customer_MembershipMonthly". The conflict occurred in database "TpmDatabase", table "dbo.MembershipMonthly", column "MembershipMonthly".
Completion time: 2021-05-07T14:10:47.1087103+01:00

iv. Update existing customer with a new membership.

This is a complex transaction as outlined in the project spec. It will take in an existing customer and modify their membership to an equivalent monthly one once their current membership has finished.

For example, Customer ID 1 wishes to update her membership. Her current membership was due to expire on 2020-08-18 as shown below.

Messages

(1 row affected)
Rolling back
Msg 141, Level 16, State 6, Line 14
The INSERT statement conflicted with the FOREIGN KEY constraint "FK_Customer_MembershipMonthly". The conflict occurred in database "TpmDatabase", table "dbo.MembershipMonthly", column "MembershipMonthly".
Completion time: 2021-05-07T14:10:47.1087103+01:00

CustomerID	CustomerFirstName	CustomerLastName	HouseNumber	StreetAddress	City	Postcode	MembershipMonthly	MembershipCode	StartDate	EndDate	MembershipLengthMonths	MembershipLengthDays	NextPayDate	FinalPayDate
1	Randi	Tettery	45	58 Walth Road	Wilton	CO2 430	58	197	2020-07-18	2020-08-18	1	31	2020-08-18	2020-08-18

After running the supplied transaction, the customer now has an extra month added on to her payment cycle (EndDate) and the full 5 months added to her final payment date.

```
GO
DECLARE @SQL NVARCHAR(4000) = N''
-- Transaction: CurrentCustomerUpdate
-- BEGIN TRY
-- DECLARE @PaymentID AS INT
-- DECLARE @DesiredLength AS INT
-- DECLARE @CustomerID AS INT
-- SET @CustomerID = 1
-- SET @DesiredLength = 5
-- -- Gets the card ID for a customer
-- DECLARE @CardID AS INT
-- SELECT @CardID = CI.CardID FROM Customer C
-- JOIN Payment P ON C.CustomerID = P.CustomerID
-- JOIN CardInfo CI ON P.CardID = CI.CardID
-- WHERE C.CustomerID = @CustomerID
-- -- Gets the monthly cost for a given user
-- DECLARE @MonthCost AS DECIMAL(5,2)
-- SELECT @MonthCost = m.MembershipCostMonthly FROM Customer c
-- JOIN MembershipMonthly m ON c.MembershipIDMonthly = m.MembershipIDMonthly
-- WHERE c.CustomerID = @CustomerID
-- -- Gets the last ID in the payments table
-- SELECT TOP(1) @PaymentID = PaymentID FROM Payment
-- ORDER BY PaymentID DESC
-- IF (SELECT EndDate FROM Customer WHERE CustomerID=@CustomerID) <= GETDATE()
-- BEGIN
-- -- We already have a membership, add on to the end of his current
-- -- Add 18 to ID to convert to monthly equivalent of membership
-- SET MembershipIDMonthly=MembershipIDMonthly+18, EndDate=@DATEADD(month,1,EndDate), FinalPayDate = @DATEADD(month,@DesiredLength,EndDate)
-- UPDATE Customer
-- SET MembershipIDMonthly=13, StartDate=@DATEADD(month,1,StartDate), EndDate=@DATEADD(month,1,StartDate), FinalPayDate=@DATEADD(month,@DesiredLength,StartDate)
-- WHERE CustomerID=1
-- INSERT INTO Payment (PaymentID, Amount, CardID, CustomerID, UpfrontCostPaid, PaymentDate) values (@PaymentID + 1, @MonthCost, @CardID, @CustomerID, 0, @DATEADD(month,1,StartDate))
-- COMMIT TRANSACTION CurrentCustomerUpdate;
-- END
-- ELSE
-- BEGIN
-- -- Insert Customer
-- SET MembershipIDMonthly=13, StartDate=@DATEADD(month,1,StartDate), EndDate=@DATEADD(month,1,StartDate), FinalPayDate=@DATEADD(month,@DesiredLength,StartDate)
-- WHERE CustomerID=1
-- COMMIT TRANSACTION CurrentCustomerUpdate;
-- END
-- END TRY
-- BEGIN CATCH
-- ROLLBACK TRANSACTION CurrentCustomerUpdate;
-- PRINT 'Rolling back';
-- THROW;
-- END CATCH
```

Messages

(1 row affected)
(1 row affected)
Completion time: 2021-05-07T14:10:21.8144404+01:00

The results are as expected given the scenario: A 5month membership after her current single month membership.

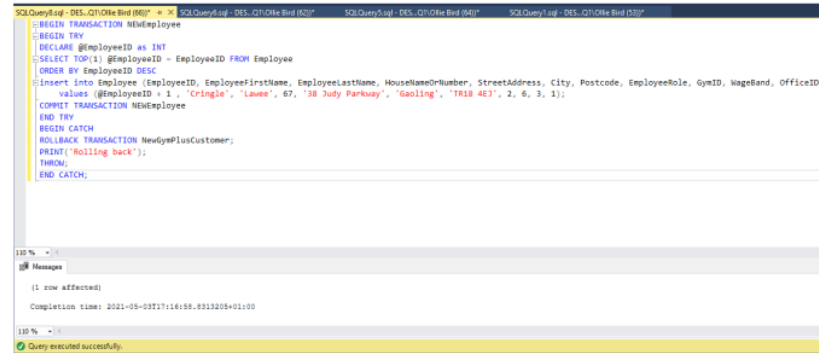
This transaction creates a new customer who is purchasing a fixed length membership.

The below displays the creation of a new gym and the addition of a customer.

[illegible]

vii. New Employee

This will add a new employee.

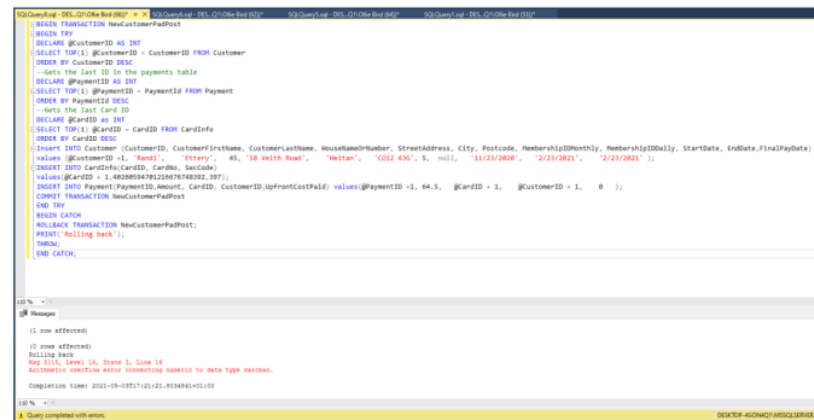


```
SQLQuery1.sql - DES-QT-01a Red (60) | SQLQuery1.sql - DES-QT-01a Red (60) | SQLQuery1.sql - DES-QT-01a Red (60) | SQLQuery1.sql - DES-QT-01a Red (60)
--BEGIN TRANSACTION NewEmployee
--BEGIN TRY
--DECLARE @EmployeeID as INT
--SELECT TOP(1) @EmployeeID = EmployeeID FROM Employee
--ORDER BY EmployeeID DESC
--Insert into Employee (EmployeeID, EmployeeFirstName, EmployeeLastName, HouseNameOrNumber, StreetAddress, City, Postcode, EmployeeRole, GymID, WageBand, OfficeID)
--values (@EmployeeID + 1, 'Cringle', 'Lawee', 67, '38 Judy Parkway', 'Gaoiling', 'TRIS 4E3', 2, 6, 3, 1);
--COMMIT TRANSACTION NewEmployee
--END TRY
--BEGIN CATCH
--ROLLBACK TRANSACTION NewEmployeeCustomer;
--PRINT('Rolling back');
--THROW;
--END CATCH;
```

100 %
Messages
(1 row affected)
Completion time: 2021-05-03T17:16:59.8313205+01:00
100 %
Query executed successfully.

viii. Bad Card Number

This transaction simulates a failure in inputting the card number. The overflow error seen is due to the card number inserted being too large for the constraint applied to that column.



```
SQLQuery1.sql - DES-QT-01a Red (60) | SQLQuery1.sql - DES-QT-01a Red (60) | SQLQuery1.sql - DES-QT-01a Red (60) | SQLQuery1.sql - DES-QT-01a Red (60)
--BEGIN TRANSACTION NewCustomerPadPost
--BEGIN TRY
--DECLARE @CustomerID as INT
--SELECT TOP(1) @CustomerID = CustomerID FROM Customer
--ORDER BY CustomerID DESC
--Gets the last ID in the payments table
--DECLARE @PaymentID as INT
--SELECT TOP(1) @PaymentID = PaymentID FROM Payment
--ORDER BY PaymentID DESC
--Gets the last Card ID
--DECLARE @CardID as INT
--SELECT TOP(1) @CardID = CardID FROM CardInfo
--ORDER BY CardID DESC
--Insert INTO Customer (CustomerID, CustomerFirstName, CustomerLastName, HouseNameOrNumber, StreetAddress, City, Postcode, MembershipMonthly, MembershipDaily, StartDate, EndDate, FinalPayDate)
--values (@CustomerID + 1, 'Randi', 'Stacey', 45, '38 South Road', 'Wilton', 'CO2 43W', 5, null, '11/23/2020', '1/23/2021', '1/23/2021');
--Insert INTO CardInfo (CardID, CardNo, SecCode)
--values (@CardID + 1, 46280547012367676392, 307);
--Insert INTO Payment (PaymentID, Amount, CardID, CustomerID, UpfrontCostPaid) values (@PaymentID + 1, 64.5, @CardID + 1, @CustomerID + 1, 0);
--COMMIT TRANSACTION NewCustomerPadPost
--END TRY
--BEGIN CATCH
--ROLLBACK TRANSACTION NewCustomerPadPost;
--PRINT('Rolling back');
--THROW;
--END CATCH;
```

100 %
Messages
(1 row affected)
(1 row affected)
Rolling back
Msg 8102, Level 16, State 1, Line 14
Arithmetic overflow error converting numeric to data type varchar.
Completion time: 2021-05-03T17:12:12.8534961+01:00
100 %
Query completed with errors.

3) Indexing

The database and use cases for it have been analysed to generate performance improvements via indexing. Considerations have been made regarding frequent similar queries from the various departments. The proposed below solution will improve performance of many common queries. The indexes proposed only apply to tables which would contain large amounts of data, in this case Employee and Customer. These tables could ultimately contain 100s to 1000s of rows of data which are frequently required to be searched and displayed. The payments table has been chosen not to be indexed as

it is inherently with payment ID. Payments are logged incrementally as they are made, hence the date is purely an identifier of when it was paid and hence should not be indexed.

a) Customer Indexing

It is proposed that four additional secondary indexes are applied to the customer table to speed up transactions.

The first is to index last name alphabetically. This is done as both by convention and for query speed. It is a common query of this type of database to return customer's names, whether that be from a receptionist, manager or financial expert. In the case of a receptionist, indexing by last name improves return speed when searching for a specific name as opposed to a customer ID. When viewing the data as a larger cluster, or when selecting all rows, logical sense dictates that there be an order other than an arbitrary ID value, in which alphabetical makes the most sense.

The second proposed index is of the value of the customers 'end date'. This value corresponds to the end of their current payment cycle, for example each month of payment would have an end date of the end of that month, however an upfront customer of a fixed term would have an 'end date' corresponding to the final date their current paid for membership is valid. Indexing then by this field is of value primarily to the finance and membership management employees. For example, this specific index aids in viewing current customers, by selecting a value of 'end date' to be greater than or equal to the current date as shown below:

```
Select CustomerFirstName, CustomerLastName, CustomerID, StartDate, EndDate from Customer
Where EndDate >= GETDATE()
Order by EndDate
```

This is an extremely common transaction and will be used by both of the above departments, in some cases with further selections imposed (such as by gym as below).

Much the same as indexing by 'End Date', it is also proposed to index by Final Payment Date. In much the same scenario in which the group wish to identify longer term customers, or to exclude them in the case of finding customers who are coming to the end of their membership. The below is an example of such.

```
Select CustomerFirstName, CustomerLastName, CustomerID, StartDate, EndDate from Customer
Where EndDate >= GETDATE()
AND
Where Enddate < FinalPaymentDate
Order by EndDate
```

The final payment date of a customer also aids to identify longer term profits as the customer is guaranteed to remain paying for that length of time, hence the reason for this indexing.

The final index chosen for the customer table is that of MembershipIDMonthly, and by extension gym ID. When querying the database in bulk with regards to all customers, the indirect join between gymID and customer provides an extremely useful logical order. If this kind of query were to be run each day, it makes sense to index via this field.

b) Employee Indexing

The second proposed table for indexing is the employee records table as it also contains a large number of records, with many foreign key references.

As with customer name, there is solid logical reasoning behind ordering the names alphabetically, as well as the gain in search speed. The same is proposed for the employee table in which the last name is indexed alphabetically.

The final proposed indexing is via the Employee Role field. Both employee role and gym ID will be indexed to order the roles into gyms. Queries looking up employees by role for both finance and by region gyms are improved by this index, such as a search of all employees when comparing wages via finance, or a potential extension to this database in which employees would have rotas and jobs as part of newly created tables.

4) **Distributed Database Model**

The second potential improvement to the current design of the database is to use a distributed database model in which the original database is fragmented into various smaller sections and physically located in various areas.

The below design will split the database both horizontally and vertically via views and select queries on those views to fragment the database initially by region.

In the case of the finance and management teams overseeing all of the gyms, no distributed model is implemented as all of the available tables are useful to those in some capacity.

a) **Horizontal fragmentation**

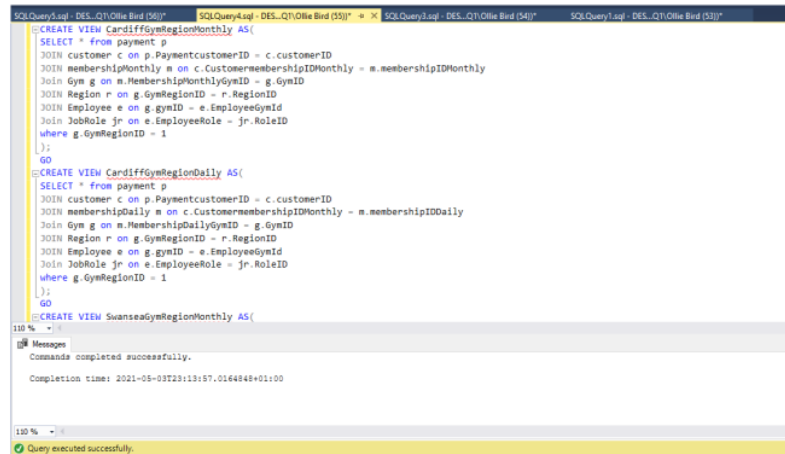
The database will be fragmented using the where clauses associated with the respective regions for each gym. This will be done inclusive of both daily and monthly membership tables for allow for a complete view. These are detailed as below:

- V1- All tables, excluding wages and card information, including monthly membership where region ID =1
- V2- All tables, excluding wages and card information, including Daily membership where region ID =1
- V3- All table, excluding wages and card information, s including monthly membership where region ID =2
- V4- All tables, excluding wages and card information, including Daily membership where region ID =2
- V5- All tables, excluding wages and card information, including monthly membership where region ID =3
- V6- All tables, excluding wages and card information, including Daily membership where region ID =3
- V7- All tables, excluding wages and card information, including monthly membership where region ID =4
- V8- All tables, excluding wages and card information, including Daily membership where region ID =4
- V9- All tables, excluding wages and card information, including monthly membership where region ID =5
- V10- All tables, excluding wages and card information, including Daily membership where region ID =5

The database is reconstructed via the union of all of these fragments via UNION ALL.

The view for the finance department is created via the union of all of the above fragments, with the addition of the wages and card information table supplied via an inner join to the employee and payments table. This also has the effect of global reconstruction as the original database is re-formed as required for the finance department.

The created views can be found below:



```
SQLQuery1.sql - DES...Q1\Ole Bed (56)
SQLQuery4.sql - DES...Q1\Ole Bed (53)
SQLQuery1.sql - DES...Q1\Ole Bed (54)
SQLQuery1.sql - DES...Q1\Ole Bed (53)

CREATE VIEW CardiffGymRegionMonthly AS(
SELECT * from payment p
JOIN customer c on p.PaymentcustomerID = c.customerID
JOIN membershipMonthly m on c.CustomermembershipIDMonthly = m.membershipIDMonthly
JOIN Gym g on m.MembershipMonthlyGymID = g.gymID
JOIN Region r on g.GymRegionID = r.RegionID
JOIN Employee e on g.gymID = e.EmployeeGymID
JOIN JobRole jr on e.EmployeeRole = jr.RoleID
where g.GymRegionID = 1
);
GO

CREATE VIEW CardiffGymRegionDaily AS(
SELECT * from payment p
JOIN customer c on p.PaymentcustomerID = c.customerID
JOIN membershipDaily m on c.CustomermembershipIDMonthly = m.membershipIDDaily
JOIN Gym g on m.MembershipDailyGymID = g.gymID
JOIN Region r on g.GymRegionID = r.RegionID
JOIN Employee e on g.gymID = e.EmployeeGymID
JOIN JobRole jr on e.EmployeeRole = jr.RoleID
where g.GymRegionID = 1
);
GO

CREATE VIEW SwanseaGymRegionMonthly AS(
SELECT * from payment p
JOIN customer c on p.PaymentcustomerID = c.customerID
JOIN membershipMonthly m on c.CustomermembershipIDMonthly = m.membershipIDMonthly
JOIN Gym g on m.MembershipMonthlyGymID = g.gymID
JOIN Region r on g.GymRegionID = r.RegionID
JOIN Employee e on g.gymID = e.EmployeeGymID
JOIN JobRole jr on e.EmployeeRole = jr.RoleID
where g.GymRegionID = 1
);
GO

Messages
Commands completed successfully.
Completion time: 2021-05-03T23:13:57.0164848+01:00

Query executed successfully.
```

A potential argument could be made for further vertical fragmentation of each of these views, for example between customer related information and employee, however in the case of each region, all of this information is necessary and can be stored as such in the horizontal fragment and selected from directly which acts as a vertical fragment. The reconstruction is then simply then outer join of each vertical fragment of each view, and the union of each of those views as listed above.

References

- [1] Mockaroo LLC, "Mockaroo Data Generator," Mockaroo, 8 14 2019. [Online]. Available: <https://www.mockaroo.com/>. [Accessed 20 04 2021].
- [2] UK Postcode , "LIST OF RANDOM POSTCODES," 2021. [Online]. Available: <https://www.ukpostcode.co.uk/random.htm>. [Accessed 21 04 2021].

Oliver Bird Database and Application Development

GRADEMARK REPORT

FINAL GRADE

83/100

GENERAL COMMENTS

Instructor

An excellent solution. You demonstrate an in-depth understanding of database design and you have produced a well-planned solution. You have a good command of the SQL environment excellent implementation of test data and subsequent testing. You have produced and documented thoughtful transactions again to test out the structure. Good analysis of the distributed data requirements and the production of appropriate views to support your theories. Try to improve you report layout a little place page number in your TOC and add figure legends to your images. However, an excellent submission.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13