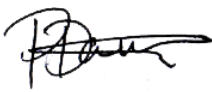




Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

School of Applied Computing

Student Name:	Peter Davies	Student P number:	P079868
Programme:	BSc Web Development		
Module:	Database Management Systems	Lecturer:	Kemi Ademoye
Assignment Title:	Book & Ride Case Study		
Hand Out Date:	October 2017	Submission deadline:	21/03/2018

Disclosure:	I declare that this assignment is all my own work and that I have acknowledged all materials used from the published or unpublished works of other people. All references have been duly cited.		
Student's Signature:		Date:	

Abstract

The following report has been produced to aid Book & Ride solve two problems on their database; Indexes & Fragmentation. The report begins by demonstrating how the database has been created. Example transactions have then been evaluated and appropriate indexes created. Finally, a distributed database solution has been prepared.

Contents

1. Database Creation.....	1
1.1 Database Creation.....	1
1.1.1 Region Table.....	1
1.1.2 City Table.....	1
1.1.3 Office Table.....	2
1.1.4 Driver Table.....	2
1.1.5 Member Table.....	3
1.1.6 Booking Table.....	3
1.1.7 Driver Schedule Table.....	3
1.1.8 ERD Diagram.....	4
1.2 Database Population.....	5
1.3 Primary & Foreign Key Referential Integrity Testing.....	8
1.4 Database Indexes.....	10
1.5 Creating New Users.....	10
1.5.1 Multi-user Access.....	10
1.5.2 Creating Logon & Users.....	11
1.5.3 Granting Select & Update Rights.....	12
1.6 Enforcing 40 Working Hour Business Rule.....	14
1.6.1 Weekly Hours Worked Transaction.....	14
1.6.2 Weekly Hours Worked Trigger.....	15
2. Performance Tuning.....	16
2.1 Example Data Query Requirements.....	16
2.2 Analysis of Examples.....	16
2.2.1 Base Tables.....	16
2.2.2 Additional Access Operations & Actioned Columns.....	17
2.2.3 Completed Queries.....	17
2.3 Secondary Indexes.....	18
2.3.1 Creation of Indexes.....	18
3. Distributed Database Solution.....	20
3.1 Fragmentation Schema & Reconstruction of Fragments.....	20
4. References.....	25
5. Bibliography.....	26

Database Creation

1.1 Database Creation

The following section of the report demonstrates, in the form of screenshots, the creation of the tables in the Book & Ride database. Including Primary Key creation, referential integrity constraints (Foreign Keys) for each table, and the check constraints implemented on certain fields. Full database creation SQL code can be found in accompanying SQL files.

1.1.1 Region Table

- Creation of region table, including Primary Key constraint with check constraint on **RegionCode** column.

```
CREATE TABLE Region (  
    RegionCode CHAR(6) NOT NULL CONSTRAINT PK_Region PRIMARY KEY,  
    CONSTRAINT CHK_Region_RegionCode CHECK (RegionCode LIKE 'R[0-9][0-9][0-9][0-9]'),  
    RegionName VARCHAR(50) NOT NULL  
);
```

Check constraint test below; attempting to insert a **RegionCode** in the wrong format.

```
INSERT INTO Region (RegionCode, RegionName) VALUES ('00001', 'South West Wales')
```

Results in following error;



Msg 547, Level 16, State 0, Line 10
The INSERT statement conflicted with the CHECK constraint "CHK_Region_RegionCode".
The conflict occurred in database "DreamHome_p079869", table "dbo.Region", column 'RegionCode'.
The statement has been terminated.

Confirming that the check constraint implemented is working correctly.

1.1.2 City Table

- Creation of City table with Primary Key Constraint, and check constraint on **CityID** column.
-

Foreign Key reference to Region table also declared.

```
CREATE TABLE City (  
    CityID CHAR(6) NOT NULL CONSTRAINT PK_City PRIMARY KEY,  
    CONSTRAINT CHK_City_CityID CHECK (CityID LIKE 'C[0-9][0-9][0-9][0-9]'),  
    CityName VARCHAR(50) NOT NULL,  
    RegionCode CHAR(6) NOT NULL, FOREIGN KEY (RegionCode) REFERENCES Region(RegionCode)  
);
```

Test on **CityID** check constraint; attempting to insert **CityID** in wrong format.

```
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('D00001', 'Swansea', 'R00001')
```

Results;

```
Messages
Msg 547, Level 16, State 0, Line 26
The INSERT statement conflicted with the CHECK constraint "CHK_City_CityID".
The conflict occurred in database "DreamHome_p079869", table "dbo.City", column 'CityID'.
The statement has been terminated.
```

1.1.3 Office Table

- Creation of Office table, including check constraint on **OfficeID**. To avoid repetition, tests comparable to ones carried out previously in this report will not be repeated as they will yield the same results.

Primary Key constraint can also be seen along with Foreign Key reference to City Table.

```
CREATE TABLE Office (
    OfficeID CHAR(7) NOT NULL CONSTRAINT PK_Office PRIMARY KEY,
    CONSTRAINT CHK_Office_OfficeID CHECK (OfficeID LIKE 'OF[0-9][0-9][0-9][0-9]'),
    CityID CHAR(6) NOT NULL, FOREIGN KEY (CityID) REFERENCES City(CityID),
    Name VARCHAR(60) NOT NULL,
    StreetAddress VARCHAR(50) NOT NULL,
    Postcode CHAR(8) NOT NULL,
    PhoneNo CHAR(12) NOT NULL,
    ManagerName VARCHAR(30) NOT NULL
);
```

1.1.4 Driver Table

- Creation of Driver table, showing Primary Key constraint on **DriverNo** column. Foreign Key references implemented on **OfficeID** from Office table and **CityID** from City table. Five check constraints have been used to check the following important data; Driver Number, License Number, Account Number, Account Sort Code and National Insurance Number.

```
CREATE TABLE Driver (
    DriverNo CHAR(6) NOT NULL CONSTRAINT PK_Driver PRIMARY KEY, CONSTRAINT CHK_Driver_DriverNo CHECK (DriverNo LIKE 'D[0-9][0-9][0-9][0-9]'),
    OfficeID CHAR(7) NOT NULL, FOREIGN KEY (OfficeID) REFERENCES Office(OfficeID),
    CityID CHAR(6) NOT NULL, FOREIGN KEY (CityID) REFERENCES City(CityID),
    FirstName VARCHAR(30) NOT NULL,
    Surname VARCHAR(30) NOT NULL,
    DOB DATETIME NOT NULL,
    StreetAddress VARCHAR(50) NOT NULL,
    PostCode CHAR(8) NOT NULL,
    PhoneNumber VARCHAR(12) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    LicenseNo CHAR(12) NOT NULL CONSTRAINT CHK_Driver_LicenseNo CHECK (LicenseNo LIKE 'LN[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    LicenseExpDate DATETIME NOT NULL,
    VehicleRegNo VARCHAR(8) NOT NULL,
    CarSize INT NOT NULL,
    AccountNo CHAR(8) NOT NULL CONSTRAINT CHK_Driver_AccountNo CHECK (AccountNo LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    AccSortCode CHAR(8) NOT NULL CONSTRAINT CHK_Driver_AccSortCode CHECK (AccSortCode LIKE '[0-9][0-9]-[0-9][0-9]-[0-9][0-9]'),
    NatInsNo CHAR(9) NOT NULL CONSTRAINT CHK_Driver_NatInsNo CHECK (NatInsNo LIKE '[A-Z][A-Z][0-9][0-9][0-9][0-9][0-9][0-9]'),
    DateRegistered DATETIME NOT NULL,
    DriverStatus VARCHAR(20) NOT NULL,
);
```

1.1.5 Member Table

- Member table, Primary Key constraint on **MemberNo** column, including check constraint. Foreign Key reference on **RegionCode** to Region table. Customer's payment details include checks to ensure correct entry for Card Number, Card CVV and Card Expiry Date.

```
CREATE TABLE Member (  
    MemberNo CHAR(6) NOT NULL CONSTRAINT PK_Member PRIMARY KEY, CONSTRAINT CHK_Member_MemberNo CHECK (MemberNo LIKE 'M[0-9]  
        [0-9][0-9][0-9][0-9]'),  
    RegionCode CHAR(6) NOT NULL, FOREIGN KEY (RegionCode) REFERENCES Region(RegionCode),  
    FirstName VARCHAR(30) NOT NULL,  
    Surname VARCHAR(30) NOT NULL,  
    DOB DATETIME NOT NULL,  
    PhoneNo CHAR(12) NOT NULL,  
    Email VARCHAR(100) NOT NULL,  
    CardNo CHAR(16) NOT NULL CONSTRAINT CHK_Member_CardNo CHECK (CardNo LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]  
        [0-9][0-9][0-9][0-9][0-9][0-9]'),  
    CardCVV CHAR(3) NOT NULL CONSTRAINT CHK_Member_CardCVV CHECK (CardCVV LIKE '[0-9][0-9][0-9]'),  
    CardExpDate CHAR(5) NOT NULL CONSTRAINT CHK_Member_CardExpDate CHECK (CardExpDate LIKE '[0-9][0-9]/[0-9][0-9]')  
);
```

1.1.6 Booking Table

- Booking table, Primary Key constraint on **BookingNo** column, including check constraint. Foreign Key references on **AssignedDriverNo**, **MemberNo**, **PickUpCityID** & **DestCityId** to Driver, Member and City tables respectively.

```
CREATE TABLE Booking (  
    BookingNo CHAR(8) NOT NULL CONSTRAINT PK_Booking PRIMARY KEY, CONSTRAINT CHK_BookingNo CHECK (BookingNo LIKE 'BK[0-9]  
        [0-9][0-9][0-9][0-9][0-9]'),  
    AssignedDriverNo CHAR(6) NOT NULL FOREIGN KEY (AssignedDriverNo) REFERENCES Driver(DriverNo),  
    MemberNo CHAR(6) NOT NULL, FOREIGN KEY (MemberNo) REFERENCES Member(MemberNo),  
    BookingDateTime DATETIME NOT NULL,  
    PickUpDateTime DATETIME NOT NULL,  
    PickUpCityID CHAR(6) NOT NULL, FOREIGN KEY (PickUpCityID) REFERENCES City(CityID),  
    PickUpAddress VARCHAR(50) NOT NULL,  
    PickUpPostCode CHAR(8) NOT NULL,  
    DestinationAddress VARCHAR(50) NOT NULL,  
    DestCityId CHAR(6) NOT NULL, FOREIGN KEY (DestCityID) REFERENCES City(CityID),  
    DestPostCode CHAR(8) NOT NULL,  
    JourneyMilage INT NOT NULL,  
    CarSize INT,  
    DeliveredDateTime DATETIME,  
    FareAmount DECIMAL NOT NULL,  
    CancellationDateTime DATETIME,  
    CancellationCharge INT,  
    PaymentDetails VARCHAR(10)  
);
```

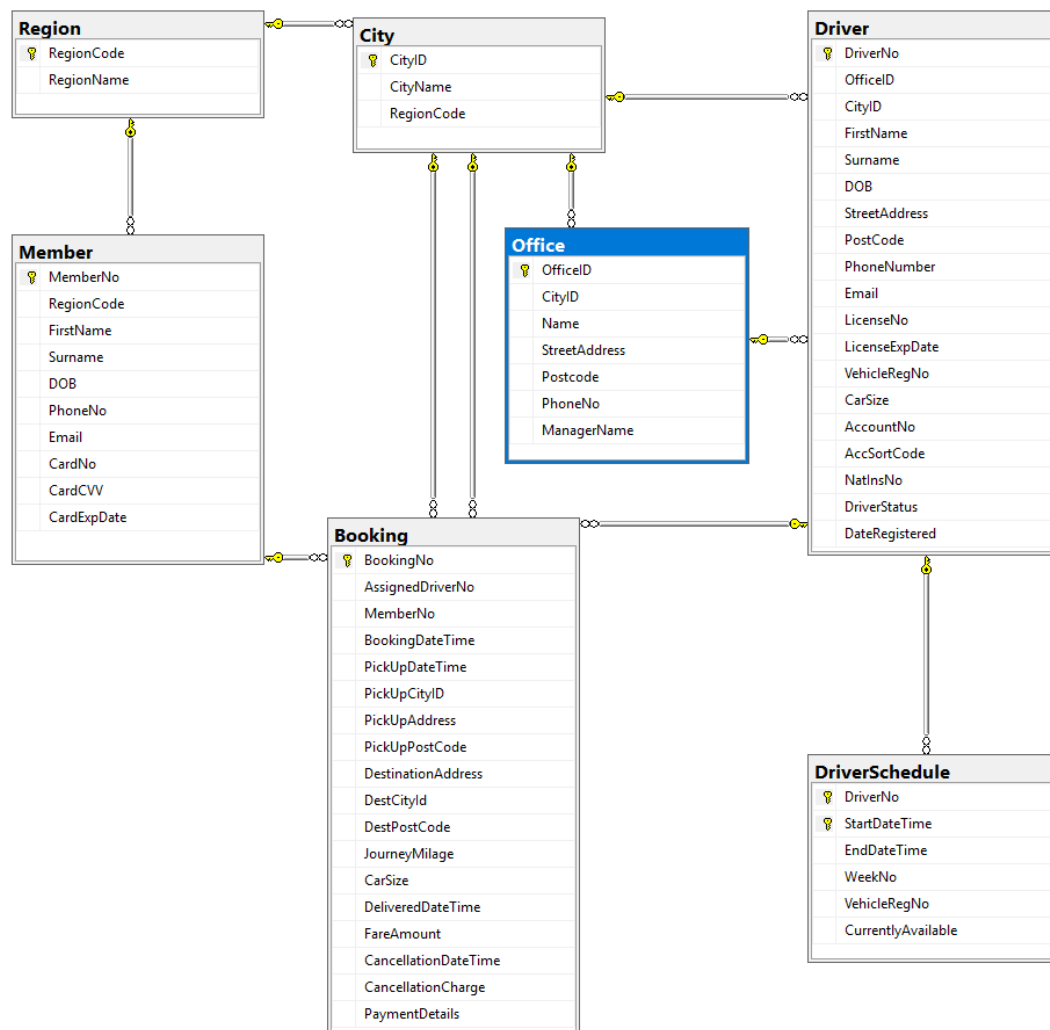
1.1.7 Driver Schedule Table

Driver Schedule table, two Primary Key constraints, **DriverNo** & **StartDateTime**. Foreign Key reference on **DriverNo** from Driver table.

```
CREATE TABLE DriverSchedule (  
    DriverNo CHAR(6) NOT NULL, FOREIGN KEY (DriverNo) REFERENCES Driver(DriverNo),  
    StartDateTime DATETIME NOT NULL,  
    EndDateTime DATETIME NOT NULL,  
    WeekNo INT NOT NULL,  
    VehicleRegNo VARCHAR(8) NOT NULL,  
    CurrentlyAvailable VARCHAR(30) NOT NULL,  
    CONSTRAINT PK_DriverSchedule PRIMARY KEY (DriverNo, StartDateTime)  
);
```

1.1.8 ERD Diagram

ERD Diagram created using Microsoft SQL Server 2017 diagram creator.



1.2 Database Population

Region Table

- For the assignment, the expansion of Book & Ride was focused to five different regions around Wales.

```
INSERT INTO Region (RegionCode, RegionName) VALUES ('R00001', 'South West Wales')
INSERT INTO Region (RegionCode, RegionName) VALUES ('R00002', 'South East Wales')
INSERT INTO Region (RegionCode, RegionName) VALUES ('R00003', 'Mid Wales')
INSERT INTO Region (RegionCode, RegionName) VALUES ('R00004', 'North West Wales')
INSERT INTO Region (RegionCode, RegionName) VALUES ('R00005', 'North East Wales')
```

City Table

- Eleven cities were added to the database, 3 for South Wales and 2 for remaining regions. Wales has 6 recognised cities [1], other larger towns in Wales were used for the assignment.

```
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00001', 'Swansea', 'R00001')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00002', 'St. Davids', 'R00001')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00003', 'Carmarthen', 'R00001')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00004', 'Cardiff', 'R00002')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00005', 'Newport', 'R00002')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00006', 'Aberystwyth', 'R00003')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00007', 'Newtown', 'R00003')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00008', 'Bangor', 'R00004')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00009', 'Conwy', 'R00004')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00010', 'Wrexham', 'R00005')
INSERT INTO City (CityID, CityName, RegionCode) VALUES ('C00011', 'St. Asaph', 'R00005')
```

Office Table

- One office for each region was inserted.

```
INSERT INTO Office (OfficeID, CityID, Name, StreetAddress, PostCode, PhoneNo, ManagerName)
VALUES ('OF00001', 'C00001', 'B&R Swansea', '13 The Kingsway', 'SA1 5HY', '01792665885', 'Peter Davies')
INSERT INTO Office (OfficeID, CityID, Name, StreetAddress, PostCode, PhoneNo, ManagerName)
VALUES ('OF00002', 'C00002', 'B&R Cardiff', '136 Queen Street', 'CF10 2BY', '02920223581', 'Jaydee Harris')
INSERT INTO Office (OfficeID, CityID, Name, StreetAddress, PostCode, PhoneNo, ManagerName)
VALUES ('OF00003', 'C00006', 'B&R Aberystwyth', '27 Great Darkgate St', 'SY23 1DE', '01970611222', 'Daniel Makangu')
INSERT INTO Office (OfficeID, CityID, Name, StreetAddress, PostCode, PhoneNo, ManagerName)
VALUES ('OF00004', 'C00008', 'B&R Bangor', '270 Main St', 'BT20 5AU', '02891456700', 'Timothy Valler')
INSERT INTO Office (OfficeID, CityID, Name, StreetAddress, PostCode, PhoneNo, ManagerName)
VALUES ('OF00005', 'C00011', 'B&R St. Asaph', '70 Mount Rd', 'LL17 0DB', '01745582577', 'Amy Convery')
```

Driver Table

- One driver for every city is added for each table, except for 'C00010', where three drivers exist for testing purposes. The screenshot below shows three example insert statements, full data insertion can be seen in the accompanying SQL file.

```
INSERT INTO Driver (DriverNo, OfficeID, CityID, FirstName, Surname, DOB, StreetAddress, PostCode, PhoneNumber, Email, LicenseNo, LicenseExpDate, VehicleRegNo, CarSize, AccountNo, AccSortCode, NatInsNo, DateRegistered, DriverStatus)
VALUES ('D00001', 'OF00001', 'C00001', 'Johnathan', 'James', '22-Jun-1976', '12 Fern Fawr Road', 'SA2 7YH', '07855225663', 'jjames@hotmail.com', 'LN0145891265', '31-Dec-2025', 'CF58 NUU', 7, '00120499', '20-84-41', 'JJ417079D', '13-Feb-2011', 'Available')
INSERT INTO Driver (DriverNo, OfficeID, CityID, FirstName, Surname, DOB, StreetAddress, PostCode, PhoneNumber, Email, LicenseNo, LicenseExpDate, VehicleRegNo, CarSize, AccountNo, AccSortCode, NatInsNo, DateRegistered, DriverStatus)
VALUES ('D00002', 'OF00001', 'C00002', 'Rhian', 'Mainwaring', '01-Aug-2000', '1 St. Davids House', 'SA67 2DV', '07980002245', 'mainwaringr@hssemail.com', 'LN7854126354', '30-Sep-2028', 'CF66 HAS', 5, '78941025', '30-30-65', 'RM0987656', '02-01-2018', 'Available')
INSERT INTO Driver (DriverNo, OfficeID, CityID, FirstName, Surname, DOB, StreetAddress, PostCode, PhoneNumber, Email, LicenseNo, LicenseExpDate, VehicleRegNo, CarSize, AccountNo, AccSortCode, NatInsNo, DateRegistered, DriverStatus)
VALUES ('D00003', 'OF00001', 'C00003', 'Susan', 'Montgomery', '12-Jan-1960', '120 Townhill Road', 'SA21 8IH', '07855898988', 'mongomerysusan@live.co.uk', 'LN1025874574', '01-Feb-2019', 'HB64 HKS', 4, '78545685', '20-84-41', 'CC152452K', '01-Mar-2013', 'Available')
```

Member Table

- Three members are added for each region to ensure accurate testing on the database. Four examples are shown below.

```
INSERT INTO Member (MemberNo, RegionCode, FirstName, Surname, DOB, PhoneNo, Email, CardNo, CardCVV, CardExpDate)
VALUES ('M00001', 'R00001', 'Adam', 'Jenkins', '10-May-1986', '07894511223', 'bentos87@hotmail.com', '1458789652321012', '663', '10/19')
INSERT INTO Member (MemberNo, RegionCode, FirstName, Surname, DOB, PhoneNo, Email, CardNo, CardCVV, CardExpDate)
VALUES ('M00002', 'R00001', 'Steve', 'Davies', '14-Feb-1970', '07985456321', 'stevedavies@snooker.com', '7458414123202525', '102', '01/22')
INSERT INTO Member (MemberNo, RegionCode, FirstName, Surname, DOB, PhoneNo, Email, CardNo, CardCVV, CardExpDate)
VALUES ('M00003', 'R00001', 'Stephanie', 'Whitlow', '11-Dec-1988', '07778966332', 'swhitlow@hotmail.com', '5363656520311014', '556', '04/18')
INSERT INTO Member (MemberNo, RegionCode, FirstName, Surname, DOB, PhoneNo, Email, CardNo, CardCVV, CardExpDate)
VALUES ('M00004', 'R00002', 'Georgie', 'Wells', '04-Apr-1990', '07706666136', 'gwells@email.co.uk', '7894561230124785', '885', '12/25')
INSERT INTO Member (MemberNo, RegionCode, FirstName, Surname, DOB, PhoneNo, Email, CardNo, CardCVV, CardExpDate) VALUES ('M00005', 'R00002', 'Thomas', 'Rees', '16-Apr-1988', '07854178456', 'braino88@hotmail.co.uk', '3214656598987878', '119', '10/19')
```

Booking Table

- Six bookings have been inserted. Care has been taken to ensure most of the cities are included for accurate testing. Three insert examples are shown below.

```
INSERT INTO Booking (BookingNo, AssignedDriverNo, MemberNo, BookingDateTime, PickupDateTime, PickupCityID, PickupAddress,
PickUpPostCode, DestinationAddress, DestCityID, DestPostCode, JourneyMilage, CarSize, DeliveredDateTime, FareAmount,
CancellationDateTime, CancellationCharge, PaymentDetails)
VALUES ('BK000001', 'D00001', 'M00001', '12-Feb-2018', '12-Feb-2018', 'C00001', '12 Glenville Road', 'SA3 5TE', '70 Alder
Way', 'C00001', 'SA3 4AD', 3, 5, '12-02-2018', 2.99, NULL, NULL, 'Card')
INSERT INTO Booking (BookingNo, AssignedDriverNo, MemberNo, BookingDateTime, PickupDateTime, PickupCityID, PickupAddress,
PickUpPostCode, DestinationAddress, DestCityID, DestPostCode, JourneyMilage, CarSize, DeliveredDateTime, FareAmount,
CancellationDateTime, CancellationCharge, PaymentDetails)
VALUES ('BK000002', 'D00011', 'M00013', '01-02-2018', '22-Feb-2018', 'C00011', '100 St. Asaph St', 'LL16 5UT', '700 Jolly
Way', 'C00011', 'LL10 4AD', 15, 5, '22-Feb-2018', 16.99, NULL, NULL, 'Card')
INSERT INTO Booking (BookingNo, AssignedDriverNo, MemberNo, BookingDateTime, PickupDateTime, PickupCityID, PickupAddress,
PickUpPostCode, DestinationAddress, DestCityID, DestPostCode, JourneyMilage, CarSize, DeliveredDateTime, FareAmount,
CancellationDateTime, CancellationCharge, PaymentDetails)
VALUES ('BK000003', 'D00001', 'M00003', '11-02-2018', '28-Feb-2018', 'C00003', '8 Carmarthen Road', 'SA45 8HJ', '200
Cardiff Road', 'C00003', 'SA16 9HA', 7, 7, '28-Feb-2018', 8.99, NULL, NULL, 'Cash')
```

Driver Schedule

- Data for 7 drivers has been added to the Driver Schedule table, each driver working different hours on defined weeks. Data has been entered to check the rule that no driver can work more than 40 hours each week. Triggers and Transactions have been implemented later in the report to enforce this rule on the database.

```
INSERT INTO DriverSchedule (DriverNo, StartDateTime, EndDateTime, WeekNo, VehicleRegNo, CurrentlyAvailable)
VALUES ('D00002', '01-Jan-2018 00:00:00', '01-Jan-2018 08:00:00', 1, 'CF66 HAS', 'Yes')
INSERT INTO DriverSchedule (DriverNo, StartDateTime, EndDateTime, WeekNo, VehicleRegNo, CurrentlyAvailable)
VALUES ('D00002', '03-Jan-2018 00:00:00', '03-Jan-2018 08:00:00', 1, 'CF66 HAS', 'Yes')
INSERT INTO DriverSchedule (DriverNo, StartDateTime, EndDateTime, WeekNo, VehicleRegNo, CurrentlyAvailable)
VALUES ('D00002', '05-Jan-2018 00:00:00', '05-Jan-2018 08:00:00', 1, 'CF66 HAS', 'Yes')

INSERT INTO DriverSchedule (DriverNo, StartDateTime, EndDateTime, WeekNo, VehicleRegNo, CurrentlyAvailable)
VALUES ('D00003', '6-Jan-2018 18:00:00', '7-Jan-2018 06:00:00', 1, 'HB64 HKS', 'Yes')
INSERT INTO DriverSchedule (DriverNo, StartDateTime, EndDateTime, WeekNo, VehicleRegNo, CurrentlyAvailable)
VALUES ('D00004', '7-Jan-2018 18:00:00', '8-Jan-2018 06:00:00', 2, 'BF65 GHF', 'Yes')
```

1.3 Primary & Foreign Key Referential Integrity Testing

To test the existence of the primary key in the Book & Ride database, the following code was modified and executed [2] . To avoid test repetition, testing will be carried out on the City table. Further testing is evident in SQL files accompanying this report.

```
SELECT COLUMN_NAME AS PrimaryKeys
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE OBJECTPROPERTY(OBJECT_ID(CONSTRAINT_SCHEMA + '.' + QUOTENAME(CONSTRAINT_NAME)), 'IsPrimaryKey') = 1
AND TABLE_NAME = 'City'
```

When executed, the query displays a view containing a column of all the primary keys. To confirm accurate results, a screenshot of results is also provided when the same query is run on the Driver Schedule table, which has two primary keys.

City Table

Results	Messages
PrimaryKeys	
1 RegionCode	

Driver Schedule Table

Results	Messages
PrimaryKeys	
1 DriverNo	
2 StartDateTime	

The query is then further modified to search for foreign keys in the table.

```
SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE CONSTRAINT_TYPE = 'FOREIGN KEY'
AND TABLE_NAME = 'City'
```

Once executed, the query displays the database, the foreign key constraint column and constraint type. Again, a screenshot is provided for the same query run on the City table and the Driver table which contains two foreign key constraints.

City Table

Results		Messages					
	CONSTRAINT_CATALOG	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	CONSTRAINT_TYPE
1	Book & Ride	dbo	FK_City_RegionCode_3B75D760	Book & Ride	dbo	City	FOREIGN KEY

Driver Table

Results Messages

	CONSTRAINT_CATALOG	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	CONSTRAINT_TYPE
1	Book & Ride	dbo	FK_Driver_OfficeID_0E6E26BF	Book & Ride	dbo	Driver	FOREIGN KEY
2	Book & Ride	dbo	FK_Driver_CityID_0F624AF8	Book & Ride	dbo	Driver	FOREIGN KEY

To test the relationship integrity of the primary in the city table, join queries from the Primary Key **CityID** in the City table ON **CityID** as a foreign key in the office, driver and booking table have been written and executed.

- Join **CityID** PK in City ON **CityID** FK in Office

```
SELECT c.CityID AS CityTable, o.CityID AS OfficeTable
FROM City c
JOIN Office o ON c.CityID = o.CityID
```

- Join **CityID** PK in City ON **CityID** in Driver

```
SELECT c.CityID AS CityTable, d.CityID AS DriverTable
FROM City c
JOIN Driver d ON c.CityID = d.CityID
```

- Join **CityID** PK in City ON **PickUpCityID** AND **DestCityId** in Booking

```
SELECT c.CityID AS CityTable, b.PickUpCityID AS BookingTable, b.DestCityID AS BookingTable1
FROM City c
JOIN Booking b ON c.CityID = b.PickUpCityID AND c.CityID = b.DestCityId
```

When executed the above queries display the matching PK **CityID** in tables Office, Driver and Booking which contain foreign key references to **CityID** in the city table.

City ON Office

	CityTable	Office Table
1	C00001	C00001
2	C00002	C00002
3	C00006	C00006
4	C00008	C00008
5	C00011	C00011

City ON Driver

	CityTable	DriverTable
1	C00001	C00001
2	C00002	C00002
3	C00003	C00003
4	C00004	C00004
5	C00005	C00005
6	C00006	C00006
7	C00007	C00007
8	C00008	C00008
9	C00009	C00009
10	C00010	C00010
11	C00011	C00011

City ON Booking

	CityTable	BookingTable	BookingTable 1
1	C00001	C00001	C00001
2	C00011	C00011	C00011
3	C00003	C00003	C00003

1.4 Database Indexes

Database indexes are similar to indexes in a book, they allow a query to retrieve data from a database efficiently, and indexes drastically speed up retrieval of data. Indexes relate to tables in the DB and involve one or keys established on each tables columns. Small databases will not see the value of indexes, however, when the amount of data stored in a database grows, queries take longer, thus the need for correct indexes [3]. The indexes for this database are investigated and implemented in the section 2, page 16 of this report.

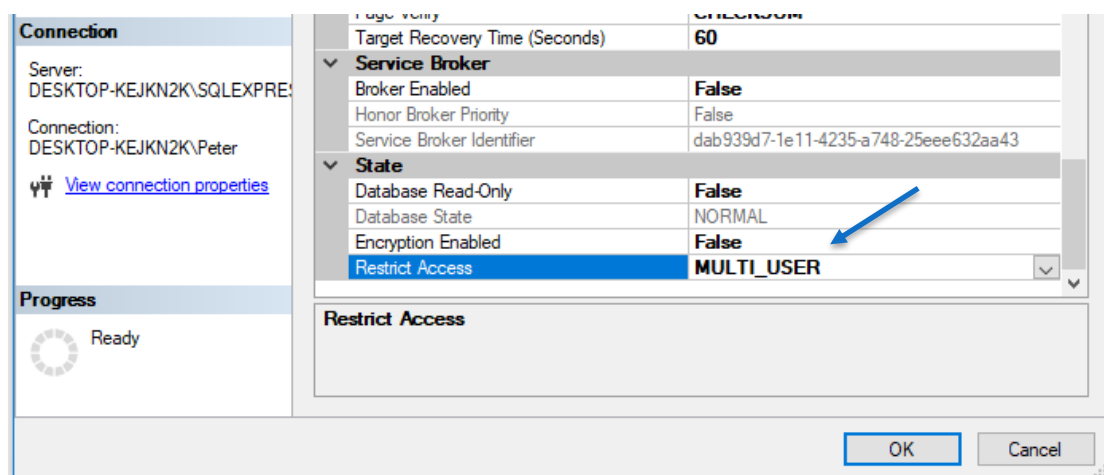
1.5 Creating New Users

1.5.1 Multi-user Access to Database

Multi-user access can also be authorised on the database by two different methods.

Method one – Using the GUI

Right-click on the Book & Ride database and selecting properties > options > state and selecting MULTI_USER from the drop-down menu.



Method Two - SQL

```
ALTER DATABASE [Book & Ride]
SET MULTI_USER
GO
```

1.5.2 Creating Logon & Users

In a real world scenario it would be beneficial to create logons and users for each member of staff in the office, each one with varying levels of rights on the database. In the Book & Ride assignment scenario it has been requested to create a logon and user account for one other database user, for this assignment, a logon and user has been created for the manager of the Swansea office. Joe Bloggs.

To create access for multiple users the following steps are required.

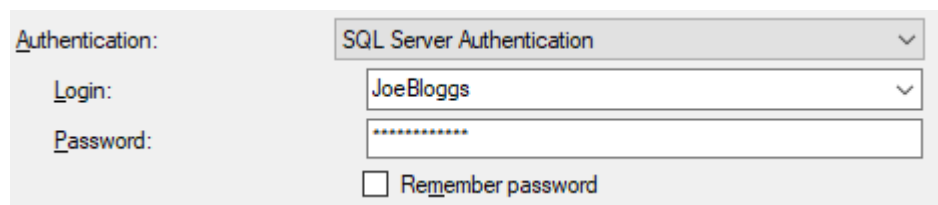
First, a login must be created.

```
-- Create login to server with UserName JoeBloggs
CREATE LOGIN JoeBloggs
    WITH PASSWORD = 'otherUser123';
GO
```

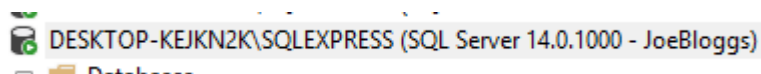
Second, a user is created for the login.

```
-- Creates a database user for the login created above.
CREATE USER JoeBloggs FOR LOGIN JoeBloggs;
GO
```

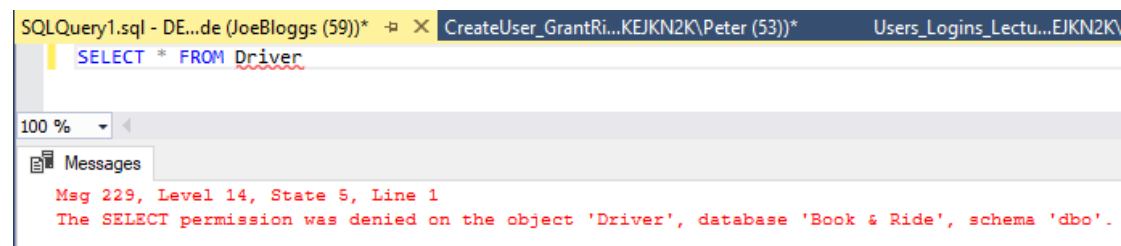
It is now possible to connect to the database a new user by using 'SQL Server Authentication' at the login prompt.



The new user can then be seen in the database explorer bar.



However, the newly created user needs to be granted rights on the database to perform queries. To test this, a simple SELECT query is run.



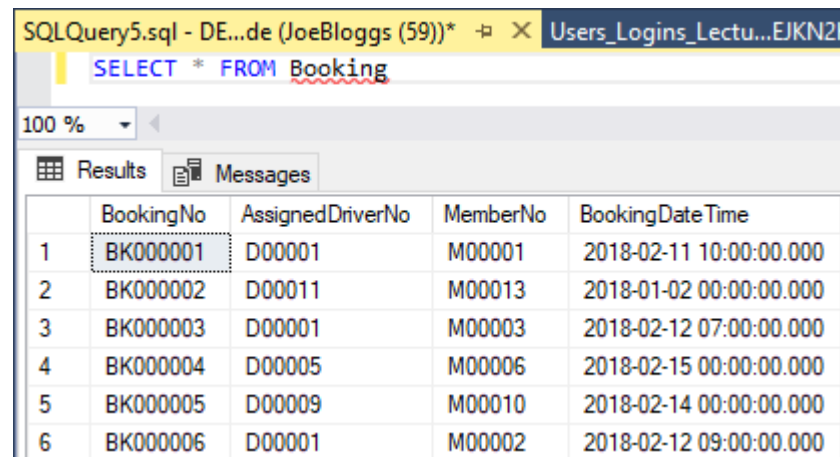
The user can be seen in the tab as **(JoeBloggs (59))***, the above error message occurs when running the query.

1.5.3 Granting Select and Update Rights

Once the new user created, it is then possible to grant the new user rights. For this assignment one other user is required to possess SELECT and UPDATE rights, to implement these rights, the following code is used.

```
-- GRANT SELECT AND UPDATE RIGHTS ACROSS THE DATABASE TO USER 'JoeBloggs' --  
  
GRANT SELECT, UPDATE ON Booking TO JoeBloggs;  
GRANT SELECT, UPDATE ON City TO JoeBloggs;  
GRANT SELECT, UPDATE ON Driver TO JoeBloggs;  
GRANT SELECT, UPDATE ON DriverSchedule TO JoeBloggs;  
GRANT SELECT, UPDATE ON Member TO JoeBloggs;  
GRANT SELECT, UPDATE ON Office TO JoeBloggs;  
GRANT SELECT, UPDATE ON Region TO JoeBloggs;
```

Test with SELECT statement

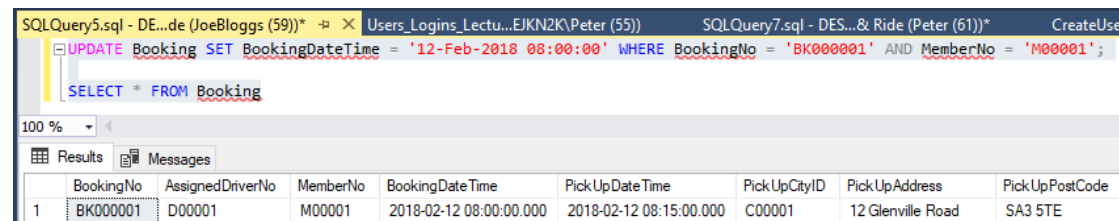


The screenshot shows a SQL query window with the query `SELECT * FROM Booking`. The results are displayed in a table with the following columns: BookingNo, AssignedDriverNo, MemberNo, and BookingDateTime.

	BookingNo	AssignedDriverNo	MemberNo	BookingDateTime
1	BK000001	D00001	M00001	2018-02-11 10:00:00.000
2	BK000002	D00011	M00013	2018-01-02 00:00:00.000
3	BK000003	D00001	M00003	2018-02-12 07:00:00.000
4	BK000004	D00005	M00006	2018-02-15 00:00:00.000
5	BK000005	D00009	M00010	2018-02-14 00:00:00.000
6	BK000006	D00001	M00002	2018-02-12 09:00:00.000

User JoeBloggs is now able to perform SELECT commands.

Test with UPDATE statement



The screenshot shows a SQL query window with the query `UPDATE Booking SET BookingDateTime = '12-Feb-2018 08:00:00' WHERE BookingNo = 'BK000001' AND MemberNo = 'M00001';`. The results are displayed in a table with the following columns: BookingNo, AssignedDriverNo, MemberNo, BookingDateTime, PickUpDateTime, PickUpCityID, PickUpAddress, and PickUpPostCode.

	BookingNo	AssignedDriverNo	MemberNo	BookingDateTime	PickUpDateTime	PickUpCityID	PickUpAddress	PickUpPostCode
1	BK000001	D00001	M00001	2018-02-12 08:00:00.000	2018-02-12 08:15:00.000	C00001	12 Glenville Road	SA3 5TE

BookingDateTime column has been updated from 10:00am to 8:00am.

To test user **JoeBloggs** only has SELECT and UPDATE rights, an INSERT query is be used.

```
SQLQuery5.sql - DE...de (JoeBloggs (59))* X BookAndRide_DataE...e (JoeBloggs (58)) Users_Logins_Lectu...EJKN2K\Peter
INSERT INTO Booking (BookingNo, AssignedDriverNo, MemberNo, BookingDateTime, PickupDateTime, Pic
DeliveredDateTime, FareAmount, CancellationDateTime, CancellationCharge, PaymentDetails) VALUE
'17 Dog Street', 'C00002', 'SA22 6RE', 6, 5, '12-02-2018 13:30:00', 5.99, NULL, NULL, 'Card')

100 %
Messages
Msg 229, Level 14, State 5, Line 1
The INSERT permission was denied on the object 'Booking', database 'Book & Ride', schema 'dbo'.
```

And error message is shown when the above INSERT query is executed as user **JoeBloggs** has not been granted rights to INSERT.

It is also possible to grant a user rights to the database using 'DataReader' and 'DataWriter' commands. This command can grant rights across all tables in the database, thus eliminating code repetition.

```
USE [Book & Ride]
GO
EXEC sp_addrolemember N'db_datawriter', N'JoeBloggs'
GO
EXEC sp_addrolemember N'db_datareader', N'JoeBloggs'
GO
```

However, the 'db_datawriter' command permits the selected user INSERT, UPDATE & SELECT rights and therefore obsolete for the task required in this assignment.

The rights can be reversed with the following SQL query.

```
USE [Book & Ride]
GO
EXEC sp_addrolemember N'db_denydatawriter', N'JoeBloggs'
GO
EXEC sp_addrolemember N'db_denydatareader', N'JoeBloggs'
GO
```

1.6 Enforcing 40 Working Hour Business Rule

Book & Ride enforces a policy of where drivers cannot work over 40 hours in one week. To enforce this rule in the database two methods have been provided.

1.6.1 Weekly Hours Worked Transaction

Firstly, a transaction has been created which would act as a stored query, to be used to enter new bookings which checks a driver's working schedule.

```
BEGIN TRANSACTION AddNewBooking

-- Insert Statement for adding new booking.
INSERT INTO Booking (BookingNo, AssignedDriverNo, MemberNo, BookingDateTime, PickupDateTime, PickupCityID, PickupAddress,
    PickupPostCode, DestinationAddress, DestCityID, DestPostCode, JourneyMilage, CarSize, DeliveredDateTime, FareAmount,
    CancellationDateTime, CancellationCharge, PaymentDetails)
VALUES ('BK000006', 'D00009', 'M00012', '21-Feb-2018 11:15:00', '21-Feb-2018 11:30:00', 'C00010', '69 Mold Road,
    Wrexham', 'LL11 2AH', '11 Benjamin Road', 'C00010', 'LL13 8EE', 0.9, 5, NULL, 0.99, NULL, NULL, 'Card')

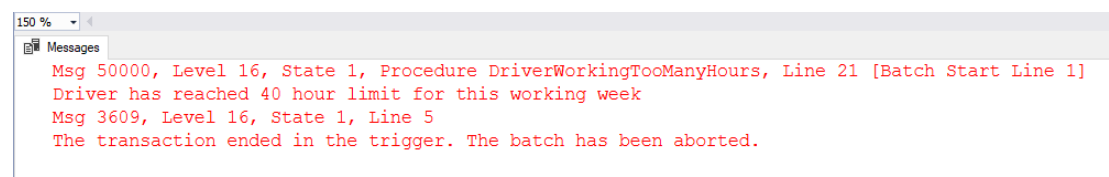
-- Get hours worked for specified driver on specified week & store in variable @HoursWorked.
DECLARE @HoursWorked DATETIME;
SET @HoursWorked = (SELECT SUM(DATEDIFF(MINUTE, [StartDateTime], [EndDateTime]) / 60) FROM DriverSchedule WHERE DriverNo
    = 'D00009' AND WeekNo = 9)

-- Use IF Statement to check if resulting hours are greater or equal to 40.
IF (@HoursWorked) >= 40

BEGIN
    ROLLBACK TRANSACTION AddNewBooking
    RAISERROR('Driver worked too many hours!', 16, 1);
END
ELSE

COMMIT TRANSACTION AddNewBooking
```

The transaction attempts to insert a new record into the Booking table, then using an if-statement, the driver's hours are checked, if greater than or equal to 40 the transaction is rolled back and an error message displayed.



It is possible to test the rollback has performed correctly by running the following query and ensuring a new record has not been entered.

```
SELECT * FROM Booking
```

It can be seen from the query results below that a record has not been inserted and the rollback has been performed.

	BookingNo	AssignedDriverNo	MemberNo	BookingDate Time	Pick Up Date Time	Pick Up City ID	Pick Up Add
1	BK000001	D00001	M00001	2018-02-12 00:00:00.000	2018-02-12 00:00:00.000	C00001	12 Glenville
2	BK000002	D00011	M00013	2018-01-02 00:00:00.000	2018-02-22 00:00:00.000	C00011	100 St. As
3	BK000003	D00001	M00003	2018-11-02 00:00:00.000	2018-02-28 00:00:00.000	C00003	8 Camarth
4	BK000004	D00005	M00006	2018-02-15 00:00:00.000	2018-02-15 00:00:00.000	C00005	77 Newpor
5	BK000005	D00009	M00010	2018-02-14 00:00:00.000	2018-03-01 00:00:00.000	C00009	Drews Anti

1.6.2 Weekly Hours Worked Trigger

The second method uses a trigger which is stored on the database and called if an INSERT or UPDATE is attempted on the Booking table.

```
-- Create Trigger on Booking Table when insert or update is being actioned on the Booking Table
CREATE TRIGGER DriverWorkingTooManyHours ON Booking

-- Trigger Event after insert or update is attempted.
AFTER INSERT, UPDATE AS

-- Declare variables to store a warning message and hours worked
DECLARE @WarningMsg VARCHAR(100), @HoursWorked DATETIME;

-- Get hours worked from Driver Scehdule Table and Store in Variable
SET @HoursWorked = (SELECT SUM(DATEDIFF(MINUTE, [StartDateTime], [EndDateTime])) / 60 FROM DriverSchedule ds
LEFT JOIN inserted i ON ds.DriverNo = i.AssignedDriverNo WHERE DriverNo = 'D00009' AND WeekNo = 9)

-- IF Statement add condition to trigger; Check how many hours a driver has worked.
IF @HoursWorked >= 40
BEGIN

-- SET warning message variable value.
SELECT @WarningMsg = 'Driver ' + inserted.AssignedDriverNo + ' has reached 40 hour limit for this working week' FROM
inserted

-- Trigger action
RAISERROR(@WarningMsg, 16, 1);
ROLLBACK TRANSACTION;
END;
```

The trigger is tested by attempting to insert a record to the Booking table.

```
-- Test trigger by attempting insert into Booking Table.
INSERT INTO Booking (BookingNo, AssignedDriverNo, MemberNo, BookingDate Time, PickUpDate Time, PickUpCityID, PickUpAddress, PickUpPostCode,
DestinationAddress, DestCityID, DestPostCode, JourneyMilage, CarSize, DeliveredDate Time, FareAmount, CancellationDate Time,
CancellationCharge, PaymentDetails)
VALUES ('BK000006', 'D00009', 'M00012', '21-Feb-2018 11:15:00', '21-Feb-2018 11:30:00', 'C00010', '69 Mold Road, Wrexham', 'LL11 2AH', '11
Benjamin Road', 'C00010', 'LL13 8EE', 0.9, 5, NULL, 0.99, NULL, NULL, 'Card')
```

Results of the test shown below.

```
Messages
Msg 50000, Level 16, State 1, Procedure DriverWorkingTooManyHours, Line 21 [Batch Start Line 29]
Driver D00009 has reached 40 hour limit for this working week
Msg 3609, Level 16, State 1, Line 30
The transaction ended in the trigger. The batch has been aborted.
```

Performance Tuning

2.1 Example Data Query Requirements

The following queries have been identified as **important** to the Book & Ride database;

1. Identifying drivers available for hire in specific regions and cities.
 - Busiest at weekends when customers may be out and unable to drive due to alcohol.
2. Drivers listing their bookings for the day.
 - Most searched during the start of each working day due to most drivers working day shifts, however, as many drivers start at different times throughout the day this transaction will be frequent throughout.
3. Swansea office reading customer payment details, based on their bookings in specific regions. (Finance remains in Swansea office).
 - Busy transaction throughout the day as bookings come in, payment will need to be processed as soon as possible.
4. Swansea office searching driver payment details, total earned and to be paid.
 - Weekly transaction, busiest at the end of the Book & Ride working week (Saturday) when drivers will be paid for their bookings from the week.

2.2 Analysis of Examples

The following analysis of the transactions identified follows an example found in the book Database Systems by Connolly & Begg [4].

2.1.2 Base Tables Identified

To identify the base tables used in the important example queries for Book & Ride a transaction/relation cross-reference matrix is used to determine the tables on which to create secondary indexes.

Tables	1				2				3				4			
	C	R	U	D	C	R	U	D	C	R	U	D	C	R	U	D
Region																
City						X				X						
Office																
Driver		X				X								X		
Member						X				X						
Booking						X				X				X		
DriverSchedule		X												X		

Analysis of the table shows the base tables; Driver, Booking, Driver Schedule, Member, City are used with Driver and Booking the most frequent.

2.2.2 Additional Access Operations & Actioned Columns

From results found in 2.1.2, it is possible to identify the interactions between the base tables and query transactions, thus identifying which columns the query will act on and which operations should be added to the SELECT statement.

*Frequency figures estimated based on current small amount of database entries.

Table	Transaction	Field	Frequency (Per Day)
Driver	(1)	Predicate: CityID Join: DriverSchedule ON DriverNo	100's
	(2)	Join: Booking ON AssignedDriverNo	10 - 50
	(4)	Join: Booking ON AssignedDriverNo	1 - 5
Booking	(2)	Predicate: AssignedDriverNo, PickUpDateTime	10 - 50
	(3)	Join: Member ON MemberNo	100's
	(4)	Predicate: AssignedDriverNo	1 - 5
Driver Schedule	(1)	Predicate: StartDateTime	100's
	(4)	Predicate: WeekNO Join: Driver ON DriverNo	1 - 5
Member	(2)	Join: Booking ON MemberNo	10 - 50
	(3)	Join: City ON RegionCode	100's
City	(2)	Join: Booking ON PickUpCityID Join: Booking ON DestCityID	10 - 50
	(3)	Predicate: RegionCode	100's

2.2.3 Completed Queries

1. Identifying drivers available for hire in specific regions and cities.

```
SELECT d.DriverNo, d.CityID, d.FirstName, d.Surname, d.CarSize, d.VehicleRegNo, ds.StartDateTime, ds.EndDateTime, ds.CurrentlyAvailable
FROM Driver d
JOIN DriverSchedule ds ON d.DriverNo = ds.DriverNo
WHERE d.CityID = 'C00010' AND ds.StartDateTime BETWEEN '28-Feb-2018' AND '01-Mar-2018'
```

2. Drivers listing their bookings for the day.

```
SELECT b.BookingNo, b.AssignedDriverNo, b.PickUpDateTime, b.PickUpAddress, c.CityName, b.PickUpPostCode, b.DestinationAddress, c1.CityName,
b.DestPostCode, b.JourneyMilage, m.FirstName, m.Surname, m.PhoneNo, b.FareAmount, b.PaymentDetails
FROM Booking b
INNER JOIN Driver d ON d.DriverNo = b.AssignedDriverNo
INNER JOIN Member m ON m.MemberNo = b.MemberNo
INNER JOIN City c ON c.CityID = b.PickUpCityID
INNER JOIN City c1 ON c1.CityID = b.DestCityID
WHERE b.AssignedDriverNo = 'D000001' AND b.PickUpDateTime BETWEEN '12-Feb-2018' AND '13-FEB-2018'
```

3. Swansea office reading customer payment details, based on their bookings in specific regions. (Finance remains in Swansea office).

```
SELECT DISTINCT c.RegionCode, b.BookingNo, m.MemberNo, m.FirstName, m.Surname, m.CardNo, m.CardExpDate, m.CardCVV, b.FareAmount, b.CancellationCharge,
b.PaymentDetails
FROM Member m
JOIN Booking b ON b.MemberNo = m.MemberNo
JOIN City c ON c.RegionCode = m.RegionCode
WHERE c.RegionCode = 'R00001'
```

4. Swansea office searching for driver payment details, their total earnings for a given week and calculating how much the driver is to be paid. (Based on the scenario that Book & Ride pays 50% commission to drivers for each fare.)

```
SELECT d.DriverNo, d.FirstName, d.Surname, d.AccountNo, d.AccSortCode, b.BookingNo, b.FareAmount, SUM((b.FareAmount / 4) / 2) AS TotalOwed
FROM Booking b
JOIN Driver d ON b.AssignedDriverNo = d.DriverNo
JOIN DriverSchedule ds ON ds.DriverNo = d.DriverNo
WHERE b.AssignedDriverNo = 'D00001' AND ds.WeekNo = 7
GROUP BY d.DriverNo, d.FirstName, d.Surname, d.AccountNo, d.AccSortCode, b.BookingNo, b.FareAmount
```

The aim of the above query was to sum the total of the of the **FareAmount** column and divide by two to allow the Swansea office to calculate the 50% commission for each fare, then SUM the total of the alias **TotalOwed** column to show total owed to each driver at the end of each week. However, as there are for records for driver 'D0001' in week 7, the SUM function is multiplying by 4, giving inaccurate results. As a workaround, the summed column **FareAmount** is first divided by 4 to give original **FareAmount**, then divided by 2 to result in the drivers 50% commission figure. More research into this type of transaction problem is required. The query above has been taken further in the SQL files, including comments. As it is not possible to reference research in Microsoft SQL Server, references are included in the report [5]–[7].

2.3 Secondary Indexes

The table below shows the additional indexes to be created. The Member table is indexed on the **MemberNo** column which is also a primary key. It is understood that Microsoft SQL Server automatically creates a clustered index on all primary key fields [8] and therefore an index will not need to be created for this table. To demonstrate that it can be created it has been included in the interest of thoroughness.

Table	Index
Driver	CityID, DriverNo
Booking	AssignedDriverNo, PickupDateTime, MemberNo, PickupCityID, DestCityID
Driver Schedule	DriverNo, StartDateTime, WeekNo
Member	MemberNo
City	RegionCode

2.3.1 Creation of Indexes

Creation of indexes in SQL for section 1.3

An index is shown for the Member table, however, Microsoft SQL Server indexes all primary key fields automatically and therefore this index is not needed.

```
/* SECONDRY INDEX CREATION */

-- Driver Table --

CREATE INDEX DriverIndex
ON Driver (CityID, DriverNo);

-----

-- Booking Table --

CREATE INDEX BookingIndex
ON Booking (AssignedDriverNo, PickupDateTime, MemberNo, PickupCityID, DestCityID);

-----

-- Driver Schedule Table --

CREATE INDEX DriverScheduleIndex
ON DriverSchedule (DriverNo, StartDateTime, WeekNo);

-----

-- Member Table --
CREATE INDEX MemberIndex
ON Member (MemberNo);

-----

-- City Table --

CREATE INDEX CityIndex
ON City (RegionCode)

-----
```

Distributed Database Solution

3.1 Fragmentation Schema & Reconstruction of Fragments

It is requested that a fragmentation schema is derived to distribute Book & Rides data at a regional level.

As tables Region, Office and City contain a nominal amount of data which is unlikely to be modified regularly, these tables will be left out of the fragmentation schema. Due to this replication across all sites on these tables, it can be said that the fragmentation schema used for Book & Ride will be that of selective replication; where some data will be fragmented to achieve good locality of reference and tables which are not regularly updated will be replicated [4].

Driver Table

As driver information is stored in the driver table it is desirable to have this information stored regionally, therefore, horizontal fragmentation is utilised, with the fragment based on the **OfficeID** as each region has one office. However, it could be considered that the Swansea office retains a complete replication of the Driver table to retain all payment and HR details, as per assignment brief.

D₁: $\sigma_{\text{OfficeID} = \text{'OF00001'}}$ (Driver) / Swansea Office / South West Wales Region

	DriverNo	OfficeID	CityID	FirstName	Surname	DOB	StreetAddress	PostCode	PhoneNumber	Email
1	D00001	OF00001	C00001	Johnathan	James	1976-06-22 00:00:00.000	12 Fem Fawr Road	SA2 7YH	07855225663	jjames@hotmail.
2	D00002	OF00001	C00002	Rhian	Mainwaring	2000-08-01 00:00:00.000	1 St. Davids House	SA67 2DV	07980002245	mainwaringr@hs
3	D00003	OF00001	C00003	Susan	Montgomery	1960-01-12 00:00:00.000	120 Townhill Road	SA21 8IH	07855898988	mongomerysusa

D₂: $\sigma_{\text{OfficeID} = \text{'OF00002'}}$ (Driver) / Cardiff Office / South East Wales Region

	DriverNo	OfficeID	CityID	FirstName	Surname	DOB	StreetAddress	PostCode	PhoneNumber	Email
1	D00004	OF00002	C00004	Micheal	Stewart	1989-03-19 00:00:00.000	65 Bute Road	CF10 6TB	07656898985	mikestewart@yahoo
2	D00005	OF00002	C00005	Mary	Jane	1999-09-21 00:00:00.000	165 Malpharant Street	WX03 CSV	0789541425	mary_jane@icloud

D₃: $\sigma_{\text{OfficeID} = \text{'OF00003'}}$ (Driver) / Aberystwyth Office / Mid Wales Region

	DriverNo	OfficeID	CityID	FirstName	Surname	DOB	StreetAddress	PostCode	PhoneNumber	Email
1	D00006	OF00003	C00006	Dafydd	Jones	1998-10-31 00:00:00.000	Flat 56a Main Avenue	SY87 1XX	07414752565	welshdafydd@g
2	D00007	OF00003	C00007	Cai	Denty	1983-08-19 00:00:00.000	10 Powys House	LD1 8UH	07845413625	denty1@ntlwork

D₄: $\sigma_{\text{OfficeID} = \text{'OF00004'}}$ (Driver) / Bangor Office / North West Wales Region

	DriverNo	OfficeID	CityID	FirstName	Surname	DOB	StreetAddress	PostCode	PhoneNumber	Email
1	D00008	OF00004	C00008	Daniel	James	1970-12-01 00:00:00.000	19 Bangor Way	BT20 1OH	07412536654	dj@gmail.com
2	D00009	OF00004	C00009	Lucy	Hughes	1986-08-03 00:00:00.000	78 Newton Road	LL8 3BN	07864142321	little_lucy@btoper

D₅: $\sigma_{\text{OfficeID} = \text{'OF00005'}}$ (Driver) / St. Asaph Office / North East Wales Region

	DriverNo	OfficeID	CityID	FirstName	Surname	DOB	StreetAddress	PostCode	PhoneNumber	Email
1	D00010	OF00005	C00010	Lee	Doran	1980-02-14 00:00:00.000	16 Baglan View	LL8 8UT	07451212414	dorans@hsbc.co.uk
2	D00011	OF00005	C00011	Alex	Lakkis	1987-02-07 00:00:00.000	87 West Cross Avenue	SY99 9GG	07895687456	lakkis_alex@gmail.com
3	D00012	OF00005	C00010	Lawrence	Merrigan	1988-08-16 00:00:00.000	16 Vivian Terrace	LL77 9UT	07895658574	merrigan@civicresure
4	D00013	OF00005	C00010	Mark	Bond	1990-01-16 00:00:00.000	60 Castle Road	LL11 6YT	079865321456	007@bondcarsales.co.

Reconstruction is achieved with the use of a UNION.

$$D1 \cup D2 \cup D3 \cup D4 \cup D5$$

Member Table

Mixed fragmentation is used to fragment the Member table. As learnt from the assignment brief, Swansea retains the finance department for Book & Ride, therefore the table is first split vertically to separate members payment details.

$M_1: \Pi_{\text{MemberNo, CardNo, CardCVV, CardExpDate}} (\text{Member})$

	MemberNo	CardNo	CardCVV	CardExpDate
1	M00001	1458789652321012	663	10/19
2	M00002	7458414123202525	102	01/22
3	M00003	5363656520311014	556	04/18
4	M00004	7894561230124785	885	12/25
5	M00005	3214656598987878	119	10/19
6	M00006	2020323256458541	321	11/21
7	M00007	6565454521213232	888	06/20
8	M00008	7845259584123652	108	09/19
9	M00009	2032147865412365	083	04/22
10	M00010	9898656578784545	001	01/21
11	M00011	7789456612339987	100	05/20
12	M00012	5555666841112222	900	02/23
13	M00013	0233012214445856	789	09/25
14	M00014	1245986532120332	147	10/20
15	M00015	9696565654545252	723	12/20

$M_2: \Pi_{\text{MemberNo, RegionCode, FirstName, Surname, DOB, PhoneNo, Email}} (\text{Member})$

	MemberNo	RegionCode	FirstName	Surname	DOB	PhoneNo	Email
1	M00001	R00001	Adam	Jenkins	1986-05-10 00:00:00.000	07894511223	bentos87@hotmail.com
2	M00002	R00001	Steve	Davies	1970-02-14 00:00:00.000	07985456321	stevedavies@snooker.com
3	M00003	R00001	Stephanie	Whitlow	1988-12-11 00:00:00.000	07778966332	swhitlow@hotmail.com
4	M00004	R00002	Georgie	Wells	1990-04-04 00:00:00.000	07706666136	gwells@email.co.uk
5	M00005	R00002	Thomas	Rees	1988-04-16 00:00:00.000	07854178456	braino88@hotmail.co.uk
6	M00006	R00002	Will	Goss	1985-03-28 00:00:00.000	07878087070	wgboot@rock.com
7	M00007	R00003	Summer	Kent	2000-12-27 00:00:00.000	07789456123	bailingonthereefhurts@mail.com
8	M00008	R00003	Thomas	Purton	1984-01-05 00:00:00.000	07765124545	purton1@jpsurfboards.co.uk
9	M00009	R00003	John	Evans	1956-09-10 00:00:00.000	07894565636	evans_j@uwtsd.ac.uk
10	M00010	R00004	Jeremy	Evans	1973-10-25 00:00:00.000	07765223444	jem@onhetube.com
11	M00011	R00004	Sarah	O'Shea	1969-07-16 00:00:00.000	07777897890	sarash@su.co.uk
12	M00012	R00004	Ana	Siviatre	1991-01-31 00:00:00.000	07789477711	anas@studentunion.com
13	M00013	R00005	Daniela	Tucker	1965-09-03 00:00:00.000	07999654458	tucker_daniela@nhs.gov
14	M00014	R00005	Martin	Wedlake	1978-01-16 00:00:00.000	07563123456	wedlaker@livingspace.com
15	M00015	R00005	Jordan	Carter	1995-06-17 00:00:00.000	07123456789	jordan@tooters.co.uk

Fragment M_2 is then split horizontally based on **RegionCode** to allocate members to their specific region.

M_{21} : $\sigma_{\text{RegionCode} = \text{'R00001'}}(M_2)$

	MemberNo	RegionCode	FirstName	Surname	DOB	PhoneNo	Email
1	M00001	R00001	Adam	Jenkins	1986-05-10 00:00:00.000	07894511223	bentos87@hotmail.com
2	M00002	R00001	Steve	Davies	1970-02-14 00:00:00.000	07985456321	stevedavies@snooker.com
3	M00003	R00001	Stephanie	Whitlow	1988-12-11 00:00:00.000	07778966332	swhitlow@hotmail.com

M_{22} : $\sigma_{\text{RegionCode} = \text{'R00002'}}(M_2)$

	MemberNo	RegionCode	FirstName	Surname	DOB	PhoneNo	Email
1	M00004	R00002	Georgie	Wells	1990-04-04 00:00:00.000	07706666136	gwells@email.co.uk
2	M00005	R00002	Thomas	Rees	1988-04-16 00:00:00.000	07854178456	braino88@hotmail.co.uk
3	M00006	R00002	Will	Goss	1985-03-28 00:00:00.000	07878087070	wgboot@rock.com

M_{23} : $\sigma_{\text{RegionCode} = \text{'R00003'}}(M_2)$

	MemberNo	RegionCode	FirstName	Surname	DOB	PhoneNo	Email
1	M00007	R00003	Summer	Kent	2000-12-27 00:00:00.000	07789456123	bailingonthereefhurts@mail.com
2	M00008	R00003	Thomas	Purton	1984-01-05 00:00:00.000	07765124545	purton1@jpsurfboards.co.uk
3	M00009	R00003	John	Evans	1956-09-10 00:00:00.000	07894565636	evans_j@uwtsd.ac.uk

M_{24} : $\sigma_{\text{RegionCode} = \text{'R00004'}}(M_2)$

	MemberNo	RegionCode	FirstName	Surname	DOB	PhoneNo	Email
1	M00010	R00004	Jeremy	Evans	1973-10-25 00:00:00.000	07765223444	jem@onthe tube.com
2	M00011	R00004	Sarah	O'Shea	1969-07-16 00:00:00.000	07777897890	sarash@su.co.uk
3	M00012	R00004	Ana	Siviatre	1991-01-31 00:00:00.000	07789477711	anas@studentunion.com

M_{25} : $\sigma_{\text{RegionCode} = \text{'R00005'}}(M_2)$

	MemberNo	RegionCode	FirstName	Surname	DOB	PhoneNo	Email
1	M00013	R00005	Daniela	Tucker	1965-09-03 00:00:00.000	07999654458	tucker_daniela@nhs.gov
2	M00014	R00005	Martin	Wedlake	1978-01-16 00:00:00.000	07563123456	wedlaker@livingspace.com
3	M00015	R00005	Jordan	Carter	1995-06-17 00:00:00.000	07123456789	jordan@tooters.co.uk

Reconstruction is achieved first by a UNION to restore the Regional fragment, then a NATURAL JOIN is used to restore the full Member table.

$$M_1 \bowtie (M_{21} \cup M_{22} \cup M_{23} \cup M_{24} \cup M_{25})$$

Booking Table

Considering the rule stated in the brief that 'If drivers have not been allocated the next passenger, they wait within close vicinity of their last drop off point until they are allocated another passenger' no fragmentation is implemented on the Booking table, complete replication is preferred. This is because all regions will wish to see all drivers operating at any

given time, especially if said driver picks up from one region, drops to another, and waits in that new region for another passenger.

Vertical fragmentation was considered to fragment the payment details from the Booking table to the Swansea office finance department, however, a VIEW or TRANSACTION for these details would be preferential to avoid compromising database performance.

Driver Schedule Table

Derived fragmentation is used on the Driver Schedule table. The Driver table is a parent relation to the Driver Schedule table and has already been fragmented horizontally by region (**OfficeID**); therefore, it is useful to store the Driver Schedule data by using the same fragmentation as the parent, whilst also horizontally fragmenting the Driver Schedule Table by the **CurrentlyAvailable** column to show which drivers are or are not available in a specific region.

This fragmentation system is based on the scenario where customers and employees will need to see information from the Driver Schedule table based on their office region (in case of employees), or GPS location (in case of customers booking through mobile app and accessing regionally fragmented database data based on their location).

$DS_1: \sigma_{\text{CurrentlyAvailable}} = DS \triangleright_{\text{OfficeID}} D_1$ (D_1 is a horizontal fragment $\text{OfficeID} = \text{OF00001}$)

	DriverNo	StartDateTime	EndDateTime	WeekNo	VehicleRegNo	CurrentlyAvailable
1	D00001	2018-02-12 08:00:00.000	2018-02-12 18:00:00.000	7	CF58 NUU	Yes
2	D00001	2018-02-13 08:00:00.000	2018-02-13 18:00:00.000	7	CF58 NUU	Yes
3	D00001	2018-02-14 08:00:00.000	2018-02-14 18:00:00.000	7	CF58 NUU	Yes
4	D00001	2018-02-15 08:00:00.000	2018-02-15 18:00:00.000	7	CF58 NUU	Yes
5	D00002	2018-01-01 00:00:00.000	2018-01-01 08:00:00.000	1	CF66 HAS	Yes
6	D00002	2018-01-03 00:00:00.000	2018-01-03 08:00:00.000	1	CF66 HAS	Yes
7	D00002	2018-01-05 00:00:00.000	2018-01-05 08:00:00.000	1	CF66 HAS	Yes
8	D00003	2018-01-06 18:00:00.000	2018-01-07 06:00:00.000	1	HB64 HKS	Yes

$DS_2: \sigma_{\text{CurrentlyAvailable}} = DS \triangleright_{\text{OfficeID}} D_2$ (D_2 is a horizontal fragment $\text{OfficeID} = \text{OF00002}$)

	DriverNo	StartDateTime	EndDateTime	WeekNo	VehicleRegNo	CurrentlyAvailable
1	D00004	2018-01-07 18:00:00.000	2018-01-08 06:00:00.000	2	BF65 GHF	Yes

$DS_3: \sigma_{\text{CurrentlyAvailable}} = DS \triangleright_{\text{OfficeID}} D_3$ (D_3 is a horizontal fragment $\text{OfficeID} = \text{OF00003}$)

	DriverNo	StartDateTime	EndDateTime	WeekNo	VehicleRegNo	CurrentlyAvailable
1	D00006	2018-02-19 09:00:00.000	2018-02-19 17:00:00.000	9	BN51 FMM	Yes
2	D00006	2018-02-20 09:00:00.000	2018-02-20 17:00:00.000	9	BN51 FMM	Yes
3	D00006	2018-02-21 09:00:00.000	2018-02-21 17:00:00.000	9	BN51 FMM	Yes
4	D00006	2018-02-22 09:00:00.000	2018-02-22 17:00:00.000	9	BN51 FMM	Yes
5	D00006	2018-02-23 09:00:00.000	2018-02-23 17:00:00.000	9	BN51 FMM	Yes
6	D00007	2018-02-19 08:00:00.000	2018-02-19 14:00:00.000	9	CP51 ZBC	Yes
7	D00007	2018-02-20 08:00:00.000	2018-02-20 14:00:00.000	9	CP51 ZBC	Yes
8	D00007	2018-02-21 08:00:00.000	2018-02-21 14:00:00.000	9	CP51 ZBC	Yes
9	D00007	2018-02-22 08:00:00.000	2018-02-22 14:00:00.000	9	CP51 ZBC	Yes
10	D00007	2018-02-23 08:00:00.000	2018-02-23 14:00:00.000	9	CP51 ZBC	Yes

DS₄: $\sigma_{\text{CurrentlyAvailable}} = \text{DS} \triangleright_{\text{OfficeID}} D_4$ (D_4 is a horizontal fragment OfficeID = OF00004)

	DriverNo	StartDateTime	EndDateTime	WeekNo	VehicleRegNo	CurrentlyAvailable
1	D00009	2018-02-17 10:00:00.000	2018-02-17 16:00:00.000	8	FC68 GFD	Yes
2	D00009	2018-02-18 10:00:00.000	2018-02-18 16:00:00.000	9	FC68 GFD	Yes
3	D00009	2018-02-19 10:00:00.000	2018-02-19 16:00:00.000	9	FC68 GFD	Yes
4	D00009	2018-02-20 10:00:00.000	2018-02-20 16:00:00.000	9	FC68 GFD	Yes
5	D00009	2018-02-21 10:00:00.000	2018-02-21 16:00:00.000	9	FC68 GFD	Yes
6	D00009	2018-02-22 10:00:00.000	2018-02-22 16:00:00.000	9	FC68 GFD	Yes
7	D00009	2018-02-23 10:00:00.000	2018-02-23 16:00:00.000	9	FC68 GFD	Yes
8	D00009	2018-02-24 10:00:00.000	2018-02-24 16:00:00.000	9	FC68 GFD	Yes

DS₅: $\sigma_{\text{CurrentlyAvailable}} = \text{DS} \triangleright_{\text{OfficeID}} D_5$ (D_5 is a horizontal fragment OfficeID = OF00005)

	DriverNo	StartDateTime	EndDateTime	WeekNo	VehicleRegNo	CurrentlyAvailable
1	D00010	2018-02-25 00:00:00.000	2018-02-25 12:00:00.000	10	FA07 OAJ	Yes
2	D00010	2018-02-26 00:00:00.000	2018-02-26 12:00:00.000	10	FA07 OAJ	Yes
3	D00010	2018-02-27 00:00:00.000	2018-02-27 12:00:00.000	10	FA07 OAJ	Yes
4	D00011	2018-02-26 07:00:00.000	2018-03-26 15:00:00.000	11	W11 BGG	Yes
5	D00011	2018-02-27 07:00:00.000	2018-03-27 15:00:00.000	11	W11 BGG	Yes

	DriverNo	StartDateTime	EndDateTime	WeekNo	VehicleRegNo	CurrentlyAvailable
1	D00010	2018-02-28 00:00:00.000	2018-02-28 12:00:00.000	10	FA07 OAJ	No
2	D00011	2018-02-28 07:00:00.000	2018-03-28 15:00:00.000	11	W11 BGG	No

Reconstruction is achieved with the use of a UNION.

$$\text{DS1} \cup \text{DS2} \cup \text{DS3} \cup \text{DS4} \cup \text{DS5}$$

References

- [1] "Frequently asked questions | wales.com." [Online]. Available: http://www.wales.com/about-wales/frequently-asked-questions#Question_6. [Accessed: 13-Mar-2018].
- [2] "SQL Server: Get table primary key using sql query - Stack Overflow." [Online]. Available: <https://stackoverflow.com/questions/3930338/sql-server-get-table-primary-key-using-sql-query>. [Accessed: 02-Mar-2018].
- [3] K. Wenzel, "Database Indexes Explained - Essential SQL." [Online]. Available: <https://www.essentialsql.com/what-is-a-database-index/>. [Accessed: 13-Mar-2018].
- [4] T. Connolly and C. Begg, *Database Systems*. 2016.
- [5] "To calculate sum() two alias named columns - in sql - Stack Overflow." [Online]. Available: <https://stackoverflow.com/questions/3932205/to-calculate-sum-two-alias-named-columns-in-sql?rq=1>. [Accessed: 08-Mar-2018].
- [6] "mysql - How to sum alias column table sql - Stack Overflow." [Online]. Available: <https://stackoverflow.com/questions/31743103/how-to-sum-alias-column-table-sql>. [Accessed: 08-Mar-2018].
- [7] "Getting duplicate results from query... but only on some records - Microsoft Community." [Online]. Available: https://answers.microsoft.com/en-us/msoffice/forum/msoffice_access-mso_other/getting-duplicate-results-from-query-but-only-on/97c6265f-1f81-48be-99a3-345e21882929. [Accessed: 01-Mar-2018].
- [8] "SQL Server Index Design Guide." [Online]. Available: [https://technet.microsoft.com/en-us/library/jj835095\(v=sql.110\).aspx](https://technet.microsoft.com/en-us/library/jj835095(v=sql.110).aspx). [Accessed: 10-Mar-2018].

Bibliography

- SQL Server 2008: how do I grant privileges to a username? - Stack Overflow. (n.d.). Retrieved March 7, 2018, from <https://stackoverflow.com/questions/3998634/sql-server-2008-how-do-i-grant-privileges-to-a-username#3998649>
- GRANT (Transact-SQL) | Microsoft Docs. (n.d.). Retrieved March 7, 2018, from <https://docs.microsoft.com/en-us/sql/t-sql/statements/grant-transact-sql>
- SQL Permissions: DataReader & DataWriter Roles — DatabaseJournal.com. (n.d.). Retrieved March 7, 2018, from <https://www.databasejournal.com/features/mssql/article.php/1479511/SQL-Permissions-DataReader--DataWriter-Roles.htm>
- Create a Database User | Microsoft Docs. (n.d.). Retrieved February 26, 2018, from <https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/create-a-database-user>
- sql - Set database from SINGLE USER mode to MULTI USER - Stack Overflow. (n.d.). Retrieved February 26, 2018, from <https://stackoverflow.com/questions/14652923/set-database-from-single-user-mode-to-multi-user>
- SQL Indexes. (n.d.). Retrieved February 26, 2018, from <https://www.tutorialspoint.com/sql/sql-indexes.htm>
- 'Username: Curt'. (2011). sql server - How can I get the sum of multiple datetime values? - Stack Overflow. Retrieved February 24, 2018, from <https://stackoverflow.com/questions/7793531/how-can-i-get-the-sum-of-multiple-datetime-values>