MET CS 669 DATABASE DESIGN AND IMPLEMENTATION FOR BUSINESS

Term Project: Online DVD Rental Business

Nicholas Scala

June 2015

Objective

This document serves as the initial design and implementation of a database for a DVD rental business similar to the DVD rental portion of the business pioneered by Netflix. This document includes a list of business rules, conceptual and logical ERDs, use case executions, and indexes.

TABLE OF CONTENTS

1. Project Description & Business Requirements

This project is to design a database for a DVD rental business such as Netflix.com. The store rents DVDs online. The customer business model is illustrated below:

Image Source: http://www.netflix.com/HowItWorks?lnkctr=nmhhiw

In order to rent a movie, a person must be enrolled at the online store. There are two different membership programs. As quoted from Netflix:

> "With Netflix you can rent as many DVDs from the comfort of your home and have them delivered to your door in about 1 business day. There are no late fees and no due dates, and shipping is free both ways. Programs start at $11.99 plus any applicable tax. With our most popular program, 3-at-a-time, you can rent as many DVDs as you want for just $17.99 a month. You keep a revolving library of up to three DVDs at a time and can exchange them for new available DVDs as often as you like. Our 2-at-a-time program (limit 4 DVDs per month) is $11.99 plus any applicable tax per month."
> (From http://www.netflix.com)

Below is an example use case to aid in understanding the Netflix operation:

1. A customer signs up for the 3-at-a-time program.
2. The same customer adds 10 movies to their queue.
3. Netflix mails to the customer the first three movies in their queue.
4. The customer watches and returns the first movie to Netflix.
5. Netflix mails out the next movie in the queue to the customer, which is the fourth movie added to the queue in Step #3.
6. The customer closes their account, but only returns two of the three movies the customer has at home.
7. Netflix charges the customer $25 for the missing movie.

The database will store membership information for each person, the movies she/he rented, movies in the queue to be rented, when were these movies returned, and so on. The rental history is used for two purposes:

- To give employees a basis to work from when they are asked what movie the customer has rented out or if it was lost in mail
- To determine if the movie was never mailed back by the customer

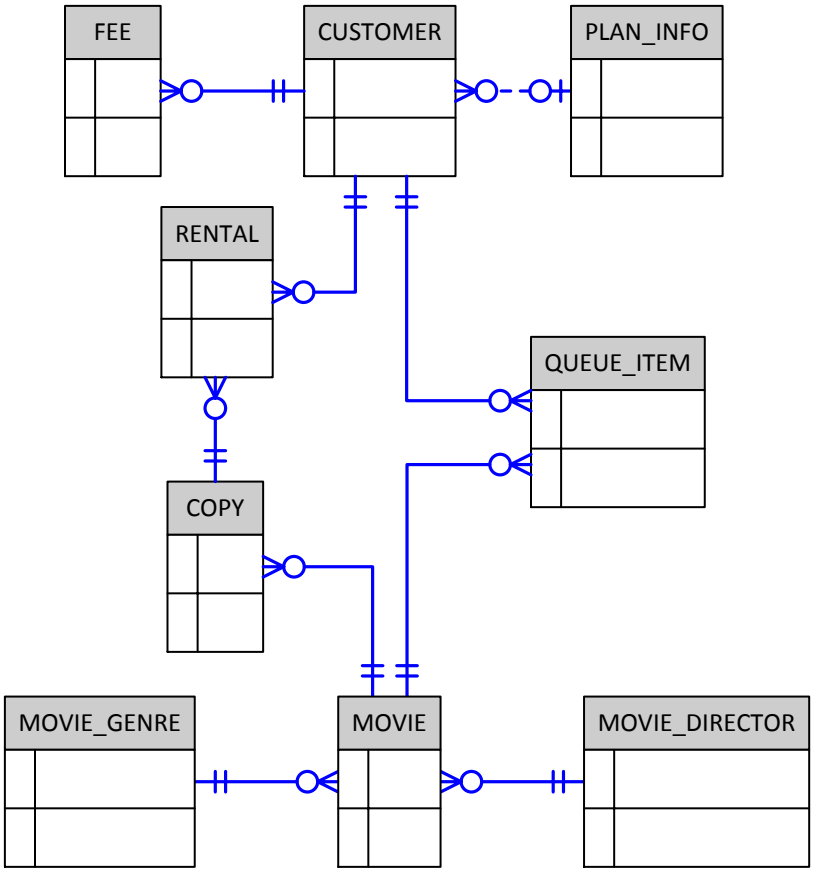
2. Business Rules

Below is a list of business rules used to create the initial database model derived from the problem statement described in Part 1:

1. Each customer can subscribe to zero or one plans; each plan is subscribed to by zero to many customers.
2. Each customer can make zero to many rentals; each rental must be made by one and only one customer.
3. Each customer can be charged zero to many fees; each fee must be charged to one and only one customer.
4. Each rental must be of one and only one copy of a movie; each copy of a movie can belong to zero to many rentals.
5. Each customer can queue zero to many movies; each movie can be queued by zero to many customers.
6. Each copy is of one and only one movie; each movie can have zero to many copies.
7. Each movie must have one and only one genre; each genre can belong to zero to many movies.
8. Each movie must be directed by one and only one director; each director can direct zero to many movies.

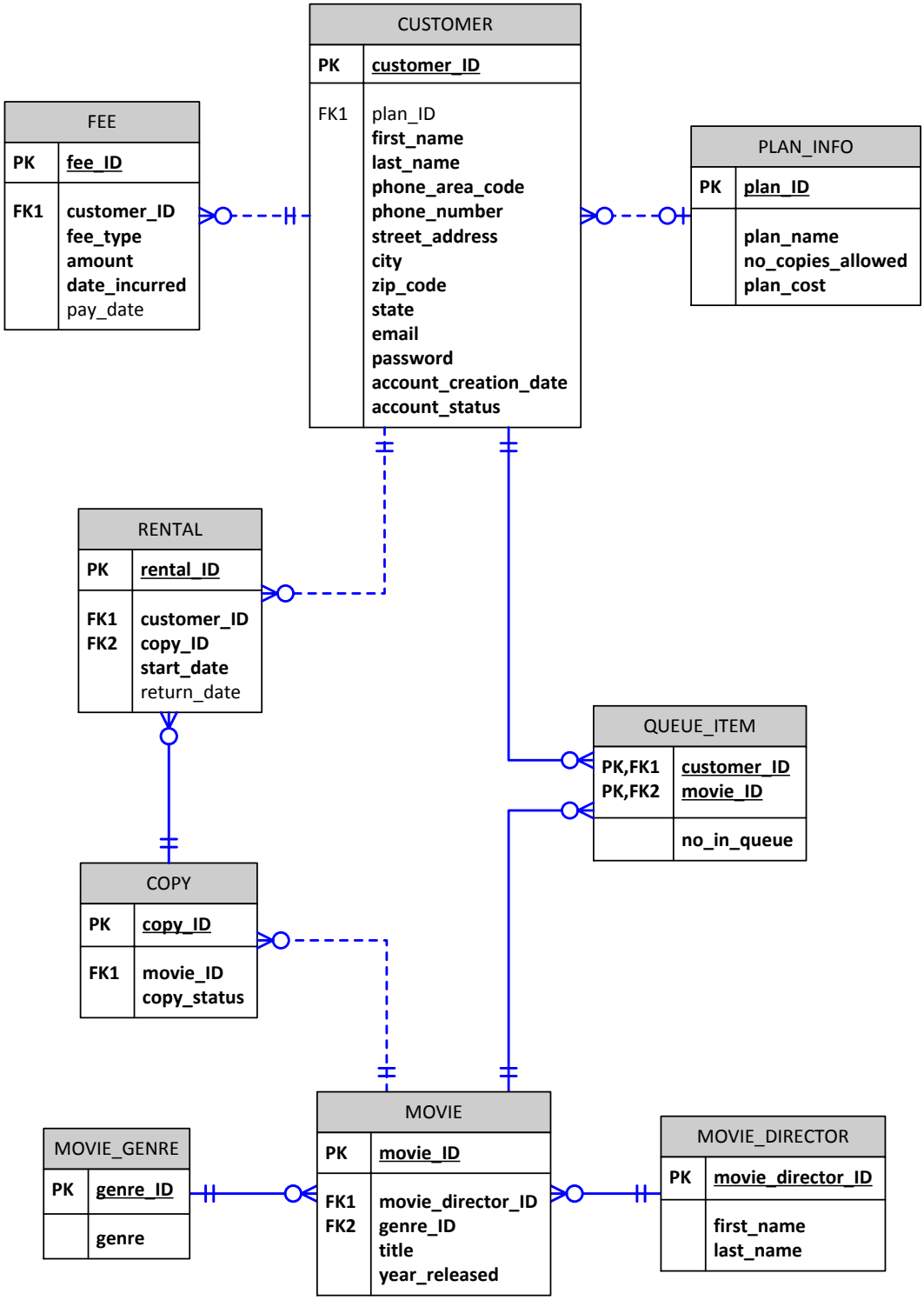
3. Conceptual Entity Relationship Diagram

Below is a Conceptual ERD for the DVD rental database:



4. Logical Entity Relationship Diagram

Below is a Logical ERD for the DVD rental database including attributes, primary key and foreign key designations, and normalization:



5. Business Use Cases

5.1 Query for All Movies by a Particular Director

Use Case Description:

A customer requests the titles of all movies that are directed by "George Lucas" or by “Rich Christiano”.

Query Execution:

```
SELECT Movie.title, Movie_Director.first_name, Movie_Director.last_name FROM Movie
JOIN Movie_Director ON Movie.movie_director_ID = Movie_Director.movie_director_ID
WHERE Movie.movie_director_ID IN
    (SELECT movie_director_ID
    FROM Movie_Director
    WHERE (first_name='George' AND last_name='Lucas') OR (first_name='Rich' AND last_name='Christiano'));
```

100 %

Results | Messages

	title	first_name	last_name
1	American Graffiti	George	Lucas
2	Star Wars Episode VI: Return of the Jedi	George	Lucas
3	Second Glance	Rich	Christiano
4	A Matter of Faith	Rich	Christiano

5.2 Query for All Active Customers and Plan Information

Use Case Description:

Management requests the names of all currently active customers, as well as the name of the current plan in which each of these customers is enrolled.

Query Execution:

```sql
SELECT Customer.first_name, Customer.last_name, Plan_Info.plan_name FROM Customer
JOIN Plan_Info ON Plan_Info.plan_ID = Customer.plan_ID
WHERE Customer.account_status = 'Active';
```

100 %

Results | Messages

	first_name	last_name	plan_name
1	Charles	Alexander	3-at-a-time
2	Margaret	Garrett	3-at-a-time
3	Cynthia	Davis	2-at-a-time
4	Jimmy	Stewart	2-at-a-time
5	Norma	Vasquez	2-at-a-time
6	Kevin	Hill	2-at-a-time
7	Bobby	Watson	3-at-a-time

5.3 Stored Procedure for Customer Adding a Movie to the Queue

Use Case Description:

A customer wants to add a movie to their queue so that the newly added movie will be the next movie they receive.

Stored Procedure Creation:

```sql
CREATE PROCEDURE Add_Movie_To_Top_Of_Queue  -- Add Movie to Top of the Queue Procedure (Use Case #3)
    @customer_id_arg DECIMAL,               -- The Customer ID parameter
    @movie_title VARCHAR(255)               -- The Movie Title parameter
AS
BEGIN
    DECLARE @movie_id DECIMAL;  -- Variable to store Movie ID
    SET @movie_id = (SELECT movie_id FROM Movie WHERE Movie.title = @movie_title); -- Set Movie ID variable equal to Movie ID
                                                                                   -- of the Movie Title customer entered

    DECLARE @no_in_queue DECIMAL;   -- Variable to store number of movies currently in the queue

    SET @no_in_queue = (SELECT COUNT(*) FROM Queue_Item WHERE customer_ID = @customer_id_arg);  -- Get the number of items currently in Queue

    IF @no_in_queue = 10    -- If there are 10 items already in the queue
    BEGIN
        DELETE FROM Queue_Item WHERE customer_ID = @customer_id_arg AND no_in_queue = 10;   -- Delete the 10th item in the queue
    END

    UPDATE Queue_Item SET Queue_Item.no_in_queue = Queue_Item.no_in_queue + 1 WHERE Queue_Item.customer_ID = @customer_id_arg;
        -- Update all the currently existing queue items' numbers in the queue + 1

    INSERT INTO Queue_Item  -- Insert new Movie into the top of the Queue
    (
        customer_ID,
        movie_ID,
        no_in_queue
    )
    VALUES
    (
        @customer_id_arg,
        @movie_id,
        1
    )
END;
```

100 %

Messages

```
Command(s) completed successfully.
```

Queue_Item Table for Customer "203" – Before Execution:

```sql
SELECT * FROM Queue_Item WHERE customer_ID = 203 ORDER BY no_in_queue;
```

100 %

Results | Messages

	customer_ID	movie_ID	no_in_queue
1	203	610	1
2	203	602	2
3	203	604	3
4	203	606	4
5	203	607	5
6	203	611	6
7	203	601	7
8	203	603	8
9	203	605	9
10	203	608	10

Stored Procedure Execution:

```
EXECUTE Add_Movie_To_Top_Of_Queue 203, 'Star Wars Episode VI: Return of the Jedi'
```

100 %

Messages

```
(1 row(s) affected)

(9 row(s) affected)

(1 row(s) affected)
```

Queue_Item Table for Customer "203" – After Execution:

```
SELECT * FROM Queue_Item WHERE customer_ID = 203 ORDER BY no_in_queue;
```

100 %

Results | Messages

	customer_ID	movie_ID	no_in_queue
1	203	609	1
2	203	610	2
3	203	602	3
4	203	604	4
5	203	606	5
6	203	607	6
7	203	611	7
8	203	601	8
9	203	603	9
10	203	605	10

5.4 Query for All Movies a Customer Has Not Yet Rented

Use Case Description:

A customer requests the titles of all the DVDs that he or she has not rented.

Query Execution:

```
SELECT Movie.title FROM Movie
WHERE Movie.movie_ID NOT IN (SELECT Copy.movie_ID FROM Copy
                             WHERE Copy.copy_ID IN (SELECT Rental.copy_ID FROM Rental
                                                    WHERE Rental.customer_ID = 201));
```

100 %

Results | Messages

	rented_movie_titles
1	Heat
2	Unforgiven
3	Match Point
4	Marie Antoinette
5	Spider-Man
6	Star Wars Episode VI: Return of the Jedi
7	A Matter of Faith

5.5 Stored Procedure for Customer Cancelling Membership

Use Case Description:

A customer cancels their membership and does not return a rented DVD, necessitating that a $25 DVD replacement fee be charged to their account. When a customer cancels their membership, they become inactive, but their DVD queue and rental history remains in the database, in the event they return as a customer.

Stored Procedure Creation:

```sql
CREATE PROCEDURE Cancel_Membership  -- Cancel Membership Procedure (Use Case #5)
    @customer_id_arg DECIMAL        -- The Customer ID parameter
AS
BEGIN
    UPDATE Customer SET account_status = 'inactive' WHERE Customer.customer_ID = @customer_id_arg;  -- Set the account to Inactive

    DECLARE @no_unreturned DECIMAL; -- Variable to hold the number of rentals unreturned
    SET @no_unreturned = (SELECT COUNT(*) FROM Rental WHERE return_date IS NULL AND customer_ID = @customer_id_arg);
        -- Set the number of unreturned rentals equal to the number of rentals where there is no return date for given customer

    DECLARE @new_fee_ID DECIMAL; -- Variable to hold the new FEE UID
    SET @new_fee_ID = (SELECT MAX(Fee.fee_ID) + 1 FROM Fee);    -- Add one to the current maximum fee UID to use a next ID to insert

    DECLARE @new_fee_amount DECIMAL (6,2); -- Variable to hold the new Fee Amount
    SET @new_fee_amount = 25.00 * @no_unreturned -- Multiply the $25 fee times the number of unreturned movies

    INSERT INTO Fee -- Insert into the Fee table
    (
        fee_ID,
        customer_ID,
        fee_type,
        amount,
        date_incurred,
        pay_date
    )
    VALUES
    (
        @new_fee_ID,
        @customer_id_arg,
        'Unreturned Movie Fee',
        @new_fee_amount,
        GETDATE(),
        NULL
    )
END;
```

100 %

Messages

```
Command(s) completed successfully.
```

Rental Table Showing Three Unreturned Movies for Customer "201":

```sql
SELECT * FROM Rental WHERE customer_ID = 201
```

100 %

Results | Messages

	rental_ID	customer_ID	copy_ID	start_date	return_date
1	801	201	703	2015-02-05 10:10:00.000	NULL
2	806	201	713	2015-01-05 23:00:00.000	NULL
3	813	201	724	2015-04-03 05:44:00.000	NULL
4	818	201	730	2015-01-07 08:49:00.000	2015-01-17 10:22:00.000

Fee Table for Customer "201" – Before Execution:

```
1  SELECT * FROM Fee WHERE customer_ID = 201
```

100 %

Results | Messages

fee_ID	customer_ID	fee_type	amount	date_incurred	pay_date

Stored Procedure Execution:

```
1  EXECUTE Cancel_Membership 201
```

100 %

Messages

```
(1 row(s) affected)

(1 row(s) affected)
```

Fee Table for Customer 201 – After Execution

```
1  SELECT * FROM Fee WHERE customer_ID = 201
```

100 %

Results | Messages

	fee_ID	customer_ID	fee_type	amount	date_incurred	pay_date
1	303	201	Unreturned Movie Fee	75.00	2015-06-17 23:25:28.507	NULL

5.6 Query for Sold Out Movies

Use Case Description:

Management requests the names of all movies that are currently sold out. A movie is sold out if all copies of the movie are currently rented and not yet returned.

Query Execution:

```sql
SELECT Movie.title -- Selects the movie title
FROM Movie
WHERE Movie.movie_ID IN(SELECT DISTINCT Copy.movie_ID -- Selects Distinct Movie IDs so Movies do not repeat
                        FROM Copy
                        WHERE Copy.movie_ID IN (SELECT a.movie_ID
                                                FROM (SELECT Copy.movie_ID, Copy.copy_status, COUNT(Copy.copy_ID) as no_of_copies
                                                      FROM Copy
                                                      GROUP BY movie_ID, copy_status) a -- Groups Movies and Copy Status
                                                GROUP BY a.movie_ID
                                                HAVING COUNT(a.movie_ID) = 1) AND copy_status = 'Rented') -- Selects Movie IDs that are completely rented
```

100 %

Results | Messages

	title
1	Lost in Translation
2	American Graffiti
3	Star Wars Episode VI: Return of the Jedi

5.7 Query for the Plan with the Most Customers Enrolled

Use Case Description:

Management requests identification of the plan with the most customer enrollees, and for that plan, the name, number of DVDs allowed at one time, and the number of customer enrollees.

Query Execution:

```sql
SELECT Plan_info.plan_name, Plan_Info.no_copies_allowed, SS2.number_enrolled
FROM Plan_Info
JOIN
    (SELECT plan_id, COUNT(plan_ID) as number_enrolled
    FROM Customer
    GROUP BY plan_id
    HAVING COUNT(plan_ID) = (SELECT MAX(SS1.plan_count)
                             FROM (SELECT plan_ID, COUNT(plan_ID) as plan_count
                                   FROM Customer
                                   GROUP BY plan_ID) SS1)) SS2
ON Plan_Info.plan_ID = SS2.plan_ID

-- SS1 retrieves the plan_ID and number of customers enrolled in each plan
-- SS2 retrieves the Maximum number of customers enrolled
-- The joined table retrieives the plan ID and the number of enrollees
        -- where the count of enrollees is equal to the maximum found in SS2
```

100 %

Results | Messages

	plan_name	no_copies_allowed	number_enrolled
1	2-at-a-time	2	4

5.8 Query for Customers Who Rented Movie More Than Once

Use Case Description:

Management requests the names of all customers, and for each customer, the titles of the movies that they rented multiple times. For each title, management would like to see the number of times it was rented by the customer, only including titles that the customer rented more than once. If a customer has no rentals, or did not rent any movies multiple times, management does not want to see them in the list.

Query Execution:

```sql
SELECT Customer.first_name, Customer.last_name, Movie.title, a.no_times_rented
FROM Customer
JOIN (SELECT Rental.customer_ID, Copy.movie_ID, COUNT(Copy.movie_ID) as no_times_rented
        FROM Rental
        LEFT JOIN Copy ON Rental.copy_ID = Copy.copy_ID
        GROUP BY customer_ID, movie_ID
        HAVING COUNT(Copy.movie_ID) > 1) a
ON Customer.customer_ID = a.customer_ID
JOIN Movie ON Movie.movie_ID = a.movie_ID;
```

100 %

Results | Messages

	first_name	last_name	title	no_times_rented
1	Bobby	Watson	Second Glance	2
2	Margaret	Garrett	A Matter of Faith	2

5.9 Query for All Movies and Number of Unique Renters

Use Case Description:

Management requests the titles of all movies, and for each movie, the number of different customers that rented the movie. They would like the list to be ordered from the highest number of different rentals to the lowest number. Multiple rentals of the same movie by the same customer only count as one unique rental. Management is interested in the number of different customers that rented the movie, but not whether the same customer rented the same movie more than once.

Query Execution:

```sql
SELECT Movie.title, ISNULL(a.no_customers,0) as no_customers
FROM Movie
LEFT JOIN (SELECT Copy.movie_ID, COUNT(DISTINCT customer_ID) as no_customers
           FROM Rental
           LEFT JOIN Copy ON Rental.copy_ID = Copy.copy_ID
           GROUP BY movie_ID) a
ON Movie.movie_ID = a.movie_ID
ORDER BY a.no_customers DESC
```

100 %

Results | Messages

	title	no_customers
1	A History of Violence	5
2	American Graffiti	3
3	Lost in Translation	3
4	Marie Antoinette	3
5	Star Wars Episode VI: Return of the Jedi	2
6	Second Glance	2
7	Heat	2
8	Unforgiven	1
9	A Matter of Faith	1
10	Match Point	0
11	Spider-Man	0

6. Indexes

6.1 Customer ID Index

Index Description:

The first index is on the customer_ID foreign key in the Rental table. It is obvious that Netflix would be very interested in which customers belong to which rentals. It is likely that Netflix leadership and customers will utilize several queries (as evident by the use cases above) that involve tracking the customer to rentals. For example, Netflix leadership would want to track the activeness of each customer to incentivize those that use the system less or reward more frequent customers. Customers, on the other hand, would be very interested in seeing their rental history. Indexing this column would help speed up these queries because instead of scanning the entire table for a particular set of customer_IDs, the query can just look to the index. Besides being a foreign key, customer_ID makes a strong candidate for an index because the Rental table will contain a vast amount of data and the customer_ID data will exhibit high sparsity. It is a non-unique index because customer_ID will repeat many times in the Rental table when the customer rents more than one movie.

Index Execution:

```
CREATE INDEX rental_customer_index
ON Rental (customer_ID)
```

100 %

Messages

```
Command(s) completed successfully.
```

6.2 Rental ID Index

Index Description:

The second index is on the copy_ID foreign key in the Rental table. Similarly, to related customer data, Netflix would be very interested in which copies belong to which rentals. It is likely that Netflix leadership will utilize several queries (as evident by the use cases above) that involve tracking the copy to rentals. For example, Netflix leadership may want to replace a particular copy of a movie if it has been rented extensively and physically worn down. Indexing this column would help speed up these queries because instead of scanning the entire table for a particular set of copy_IDs, the query can just look to the index. Besides being a foreign key, copy_ID makes a strong candidate for an index because the Rental table will contain a vast amount of data and the copy_ID data will exhibit high sparsity. It is a non-unique index because copy_ID will repeat many times in the Rental table when a particular copy is rented by many customers throughout its lifecycle.

Index Execution:

```
CREATE INDEX rental_copy_index
ON Rental (copy_ID)
```

100 %

Messages

```
Command(s) completed successfully.
```

6.3 Customer State Index

Index Description:

The third index is on the state attribute of the Customer table. Netflix leadership and marketing teams would be interested in which states their customers live in. It is likely that Netflix leadership will utilize several queries that involve customer demographics. For example, Netflix leadership may want to advertise more in regions where there are less customers in order to gain customers for that region. Indexing this column would help speed up these queries because instead of scanning the entire table for a particular set of states, the query can just look to the index. State makes a strong candidate for an index because the Customer table will contain a vast amount of data and state data will exhibit high sparsity. It is a non-unique index because states will repeat many times in the Customer table when there are multiple customers from the same state.

Index Execution:

```
1  CREATE INDEX customer_state_index
2   ON Customer (state)
```

100 %

Messages

```
Command(s) completed successfully.
```

7. Appendix

7.1 Plan_Info Table Data

```
1  SELECT * FROM Plan_Info
```

100 %

Results | Messages

	plan_ID	plan_name	no_copies_allowed	plan_cost
1	101	3-at-a-time	3	17.99
2	102	2-at-a-time	2	11.99

7.2 Customer Table Data

```
1  SELECT * FROM Customer
```

100 %

Results | Messages

	customer_ID	plan_ID	first_name	last_name	phone_area_code	phone_number	street_address	city	zip_code	state	email	password	account_creation_date	account_status
1	201	101	Charles	Alexander	063	919-9581	5 Mevlin Road	Pittsburgh	10015	Pennsylvania	calexander@blog.com	nE5xQ4wiH	2015-01-03 22:10:00.000	active
2	202	101	Margaret	Garrett	105	934-5729	24 Grover Drive	Spokane	35846	Washington	mgarrett1@csmonitor.com	Hq62ljW	2014-09-12 03:50:00.000	active
3	203	102	Cynthia	Davis	927	950-8683	942 Hudson Trail	Cumming	18657	Georgia	cdavis2@tinypic.com	uzM9Np1uAE3J	2015-03-07 18:25:00.000	active
4	204	102	Jimmy	Stewart	057	842-9003	46534 Cardinal Hill	Jacksonville	95641	Florida	jstewart3@slate.com	AdqJsD	2014-07-21 07:44:00.000	active
5	205	102	Norma	Vasquez	069	637-3519	5045 Dawn Alley	Minneapolis	65474	Minnesota	nvasquez4@dion.ne.jp	3qHiH8FFXH	2013-05-19 13:00:00.000	active
6	206	NULL	Andrea	Ellis	732	835-8705	61972 Scofield Circle	Phoenix	16875	Arizona	aellis5@live.com	fcXD3BjeB	2013-06-01 06:28:00.000	inactive
7	207	NULL	Lillian	Hunter	554	037-7436	93057 Carberry Avenue	Las Vegas	48631	Nevada	lhunter6@marriott.com	pjjxa6	2015-04-30 15:15:00.000	inactive
8	208	102	Kevin	Hill	109	804-0679	64 David Plaza	Bradenton	29884	Florida	khill7@hhs.gov	KKn5jH	2014-08-10 05:08:00.000	active
9	209	101	Bobby	Watson	633	772-9101	28 Memorial Street	Scranton	67324	Pennsylvania	bwatson8@behance.net	DcX6aann	2015-02-26 20:33:00.000	active
10	210	NULL	Deborah	Mccoy	409	843-0781	09538 Dottie Trail	Norfolk	59812	Virginia	dmccoy9@spiegel.de	Pm1krq8s	2014-10-16 01:22:00.000	inactive

7.3 Fee Table Data

```
1  SELECT * FROM Fee
```

100 %

Results | Messages

	fee_ID	customer_ID	fee_type	amount	date_incurred	pay_date
1	301	205	Lost Movie Fee	25.00	2015-06-01 12:25:00.000	NULL
2	302	203	Lost Movie Fee	25.00	2015-05-03 10:00:00.000	NULL

7.4 Movie_Director Table Data

```
SELECT * FROM Movie_Director
```

100 %

Results | Messages

	movie_director_ID	first_name	last_name
1	401	David	Cronenberg
2	402	Michael	Mann
3	403	Clint	Eastwood
4	404	Woody	Allen
5	405	Sofia	Coppola
6	406	Sam	Raimi
7	407	George	Lucas
8	408	Rich	Christiano

7.5 Movie_Genre Table Data

```
SELECT * FROM Movie_Genre
```

100 %

Results | Messages

	genre_ID	genre
1	501	Crime
2	502	Western
3	503	Drama
4	504	Action
5	505	Thriller
6	506	Sci-Fi
7	507	Comedy
8	508	Miscellaneous

7.6 Movie Table Data

```
1  SELECT * FROM Movie
```

100 %

Results | Messages

	movie_ID	movie_director_ID	genre_ID	title	year_released
1	601	401	501	A History of Violence	2005
2	602	402	501	Heat	1995
3	603	403	502	Unforgiven	1992
4	604	404	501	Match Point	2005
5	605	405	503	Lost in Translation	2003
6	606	405	503	Marie Antoinette	2006
7	607	406	504	Spider-Man	2002
8	608	407	507	American Graffiti	1973
9	609	407	506	Star Wars Episode VI: Return of the Jedi	2002
10	610	408	503	Second Glance	1992
11	611	408	503	A Matter of Faith	2014

7.7 Queue_Item Table Data

```
1  SELECT * FROM Queue_Item
```

100 %

Results | Messages

	customer_ID	movie_ID	no_in_queue
1	201	602	1
2	201	604	2
3	201	606	3
4	201	608	4
5	202	602	1
6	203	601	7
7	203	602	2
8	203	603	8
9	203	604	3
10	203	605	9
11	203	606	4
12	203	607	5
13	203	608	10
14	203	610	1
15	203	611	6
16	205	601	1
17	205	603	2
18	205	605	3
19	208	601	2
20	208	602	4
21	208	603	6
22	208	604	8
23	208	607	7
24	208	608	5
25	208	609	3

7.8 Copy Table Data

```
1  SELECT * FROM Copy
```

100 %

Results | Messages

	copy_ID	movie_ID	copy_status
1	701	601	Available
2	702	601	Available
3	703	601	Rented
4	704	601	Rented
5	705	601	Rented
6	706	602	Rented
7	707	602	Available
8	708	603	Rented
9	709	603	Available
10	710	603	Available
11	711	604	Available
12	712	604	Available
13	713	605	Rented
14	714	605	Rented
15	715	605	Rented
16	716	606	Available
17	717	606	Rented
18	718	606	Rented
19	719	606	Available
20	720	606	Rented
21	721	607	Available
22	722	607	Available
23	723	608	Rented
24	724	608	Rented
25	725	608	Rented
26	726	609	Rented
27	727	609	Rented
28	728	610	Rented
29	729	610	Available
30	730	610	Available
31	731	610	Available
32	732	611	Available
33	733	611	Available

7.9 Rental Table Data

```
1  SELECT * FROM Rental
```

100 %

Results | Messages

	rental_ID	customer_ID	copy_ID	start_date	return_date
1	801	201	703	2015-02-05 10:10:00.000	NULL
2	802	202	704	2014-10-15 18:24:00.000	NULL
3	803	203	705	2015-04-03 01:26:00.000	NULL
4	804	203	706	2015-04-08 03:33:00.000	NULL
5	805	204	708	2014-08-22 19:20:00.000	NULL
6	806	201	713	2015-01-05 23:00:00.000	NULL
7	807	202	714	2014-12-25 13:13:00.000	NULL
8	808	205	715	2014-07-19 07:19:00.000	NULL
9	809	205	717	2014-08-15 11:21:00.000	NULL
10	810	208	718	2014-09-15 10:19:00.000	NULL
11	811	209	720	2015-02-27 06:06:00.000	NULL
12	812	204	723	2014-11-21 02:59:00.000	NULL
13	813	201	724	2015-04-03 05:44:00.000	NULL
14	814	202	725	2015-01-01 11:06:00.000	NULL
15	815	208	726	2015-06-03 21:22:00.000	NULL
16	816	209	727	2015-05-09 15:17:00.000	NULL
17	817	209	728	2015-03-17 14:48:00.000	NULL
18	818	201	730	2015-01-07 08:49:00.000	2015-01-17 10:22:00.000
19	819	202	732	2014-10-14 12:56:00.000	2014-10-19 17:17:00.000
20	820	202	733	2014-11-21 07:08:00.000	2014-11-28 03:22:00.000
21	821	204	702	2014-07-28 16:00:00.000	2014-08-18 12:28:00.000
22	822	205	701	2014-03-03 09:15:00.000	2014-05-03 09:34:00.000
23	823	208	707	2014-08-11 10:20:00.000	2014-08-28 18:27:00.000
24	824	209	731	2015-04-10 19:27:00.000	2015-04-19 19:53:00.000