

Olde Time Movie Rentals: JAB Solution

Akhil Devarapalli, Joseph Telaak, Ben Charest





01


THE PROBLEM

Overview of Old Time Movie Rentals



OLDE TIME MOVIE RENTALS

Olde Time Movie Rentals is a movie rental company located in Columbia, SC. The company opened two years ago, and is growing exponentially. Managing data has become overly time consuming, so they have seeked out JAB, a local database company, to engineer a solution.



SOLUTION SPECIFICATIONS:

- Each store has a single unique distributor, but said distributor can supply multiple stores
- Each store must have a *name*, *address*, and *phone number*
- Employee information must be stored within the data (*employee name*, *supervisor*, *store location*, *home address*, *phone number*, *SSN*, *hire date*)
- Customer information must be stored within the data (*name*, *address*, *phone number*)
- Movie rental data must be stored (*employee who made sale*, *customer*, *movie title*, *movie copy*, *payment*, *date*, *status*, *rental rate*, *due date*)
- Payment specifics must be stored (*payment type*, *type based specifics* - *CC number for credit cars*, *etc*)
- Inventory data must also be stored (*media condition*, *movie title*, *director*, *description*, *star-actor*, *movie rating*, *genre*)

The JAB Plan of Action

TABLES

First, we had to organize all of the required data into tables. During this process, we identified functional dependencies and normalized the database

SQL

We implemented our solution using mySQL to engineer our database. In our SQL code, we specified constraints to ensure our database integrity.

01

02

03

04

ER Diagram

After normalizing the database, we created an ERD to visualize the relationships. In the ERD, we specified the cardinalities of each relationship and specified if the entity was a strong or weak entity.

WEB APP / REGISTER

To verify that our database worked accurately, we engineered 10 queries that tested specifics of the database. We displayed the results as reports using a webapp. We also created a register application that could be used to create transactions.



02

TABLES

Our data organization and
design process

Functional Dependencies

Functional Dependencies:

DISTRIBUTOR Table

DistributorID → (DistributorName, DistributorCity, DistributorState, DistributorZIP, DistributorPhone)

STORE Table

StoreID → (DistributorID, StoreName, StoreStreet, StoreCity, StoreState, StoreZIP, StorePhoneNumber)

MOVIE Table

MovieID → (MovieTitle, StarName, MovieDescription, MovieDirector, Rating, Genre)

STOREMOVIE Table

MovieID → StoreID

MEDIA Table

MediaID → (MovieID, StoreID, MediaCondition)

MOVIERENTAL Table

(MediaID, MovieID, RentalDateTime) → (StoreID, CustomerID, EmployeeSIN, PaymentID, StatusID, Due, OverDueCharge, RentalRate)

EMPLOYEE Table

EmployeeSIN → (ManagerSIN, EmployeeName, EmployeeStreet, EmployeeCity, EmployeeState, EmployeePhoneNumber, EmployeeHireDate)

CUSTOMER Table

CustomerID → (CustomerName, CustomerStreet, CustomerCity, CustomerState, CustomerZip, CustomerPhone)

PAYMENT Table

PaymentID → (CustomerID, EmployeeSIN, StatusID, PaymentAmount, PaymentDateTime)

PAYMENTTYPE Table

PaymentID → Description

PAYMENTSTATUS Table

StatusID → Description

RENTALSTATUS Table:

StatusID → Description

DEBITCARD Table:

PaymentID → DebitCardNumber, DebitCardType, DebitCardExpiry

CREDITCARD Table:

PaymentID → CreditCardNumber, CreditCardType, CreditCardExpiry

CHECK Table:

PaymentID → CheckNumber, BankNumber, BankName

- FD - Where one entity can determine another entity
- Preliminary step to creating data model (ERD)
- Used in order to normalize the database into BCNF
- Helps to identify and prevent any redundancies

Tables

Tables in BCNF and 4NF:

MOVIERENTAL(**MediaID**, **MovieID**, StoreID, **RentalDateTime**, *CustomerID*, *EmployeeSIN*, *PaymentID*, *StatusID*, DueDate, OverDueCharge, RentalRate)

MEDIA(**MediaID**, *MovieID*, StoreID, MediaCondition)

STOREMOVIE(**MovieID**, **StoreID**)

MOVIE(**MovieID**, MovieTitle, StarName, MoveDescriptor, MovieDirector, Rating, Genre)

STORE(**StoreID**, *DistributorID*, StoreName, StoreStreet, StoreCity, StoreState, StoreZip, StorePhoneNumber)

DISTRIBUTOR(**DistributorID**, DistributorName, DistributorCity, DistributorState, DistributorZip, DistributorPhone)

RENTALSTATUS(**StatusID**, Description)

CUSTOMER(**CustomerID**, CustomerName, CustomerStreet, CustomerCity, CustomerState, CustomerZip, CustomerPhone)

EMPLOYEE(**EmployeeSIN**, *ManagerSIN*, EmployeeName, EmployeeStreet, EmployeeCity, EmployeeState, EmployeePhone, EmployeeHireDate)

PAYMENT(**PaymentID**, *CustomerID*, *EmployeeSIN*, *StatusID*, PaymentAmount, PaymentDateTime)

PAYMENTSTATUS(**StatusID**, Description)

PAYMENTTYPE(**PaymentID**, Description)

DEBITCARD(**PaymentID**, DebitCardNumber, DebitCardType, DebitCardExpiry)

CREDITCARD(**PaymentID**, CreditCardNumber, CreditCardType, CreditCardExpiry)

CHECK(**PaymentID**, CheckNumber, BankNumber, BankName)

CASH(**PaymentID**)

- Organized data into tables
- Created primary keys and foreign keys to identify relations
- Tables were already given, but we had to identify relations, cardinalities, and keys

Primary Keys

All Primary Keys:

MOVIERENTAL → MediaID, MovieID, RentalDateTime

MEDIA → MediaID

STOREMOVIE → MovieID, StoreID

MOVIE → MovieID

STORE → StoreID

DISTRIBUTOR → DistributorID

RENTALSTATUS → StatusID

CUSTOMER → CustomerID

EMPLOYEE → EmployeeSIN

PAYMENT → PaymentID

DEBITCARD → PaymentID

CREDITCARD → PaymentID

CHECK → PaymentID

CASH → PaymentID

- The primary key is a specific key that can be used to determine all other attributes in a table
- Mostly used surrogate keys
 - Generated ID numbers
 - Entirely unique
 - Easy to query
- Had to use a concatenated key for a few entities

Foreign Keys

- Foreign keys are primary keys that occur in other tables
- They are used to reference the table they are the primary key of

All Foreign Keys:

MOVIERENTAL → CustomerID, EmployeeSIN, PaymentID, StatusID

MEDIA → MovieID

STOREMOVIE → MovieID, StoreID

STORE → DistributorID

EMPLOYEE → ManagerSIN

PAYMENT → CustomerID, EmployeeSIN, StatusID

PAYMENTTYPE → PaymentID

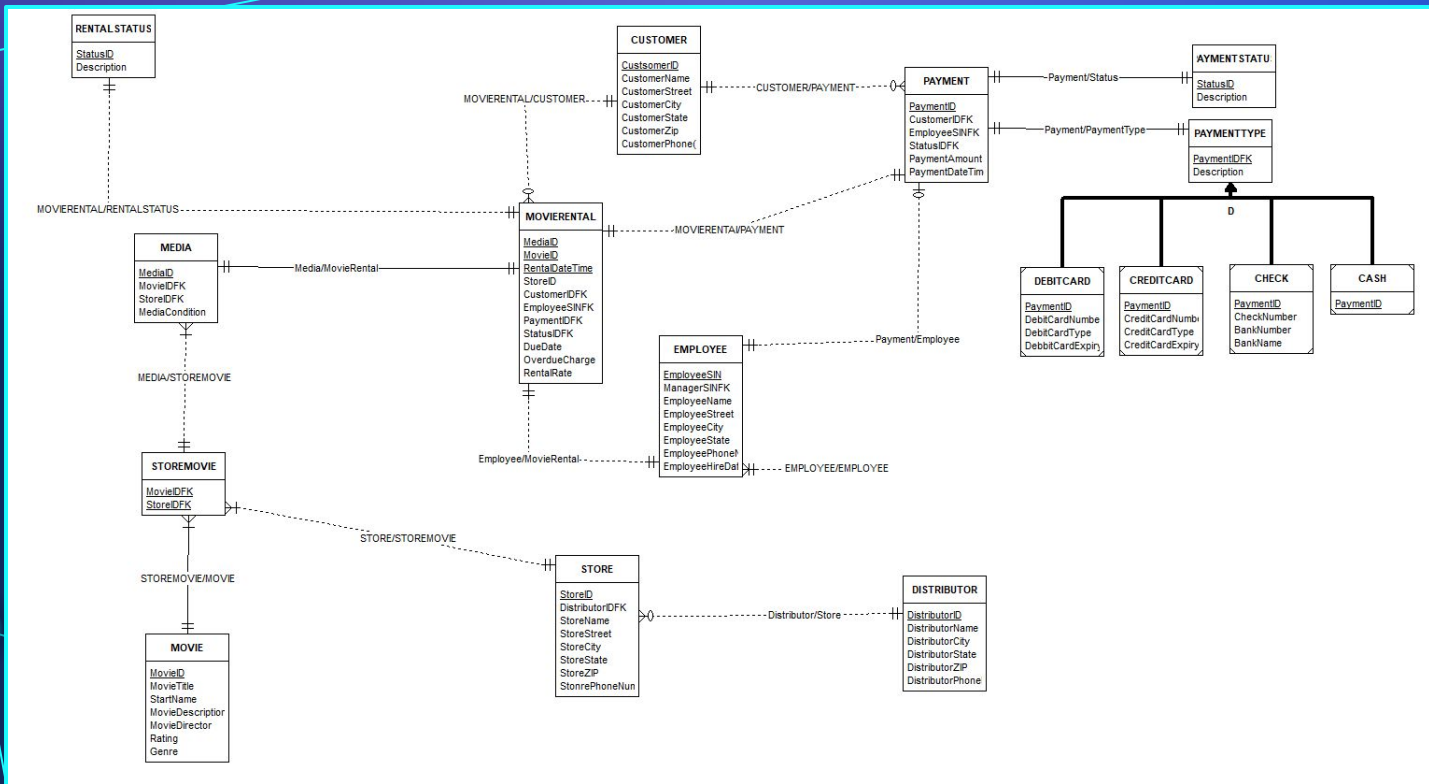


03

ER DIAGRAM

Our visual representation of the
proposed database solution

Complete ERD



The ERD Can Be Divided Into 3 Sections

- *Inventory*
- *Payment*
- *Movie rental*

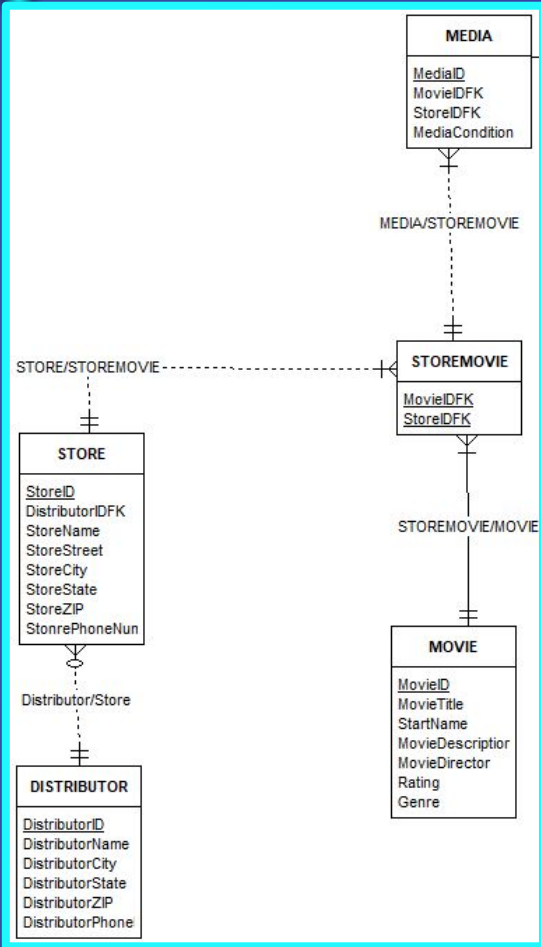




04

INVENTORY

Inventory management
with our database



Assumptions:

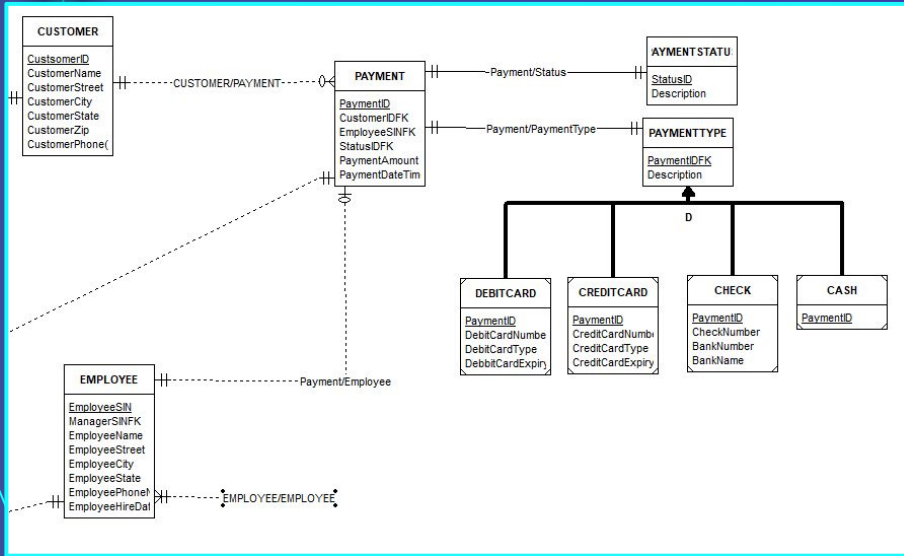
- A STORE must have one and only ONE distributor
- A DISTRIBUTOR can supply multiple stores
- A STORE must have STOREMOVIE
 - Ex. a store cannot exist if it does not carry any movies
- A MOVIE must have A STOREMOVIE
 - A movie won't be represented in our database unless it is carried in a store
- MEDIA must have STOREMOVIE
 - Media copy can't exist without a store movie



05

PAYMENT SYSTEM

Payment handling within
the database



Assumptions:

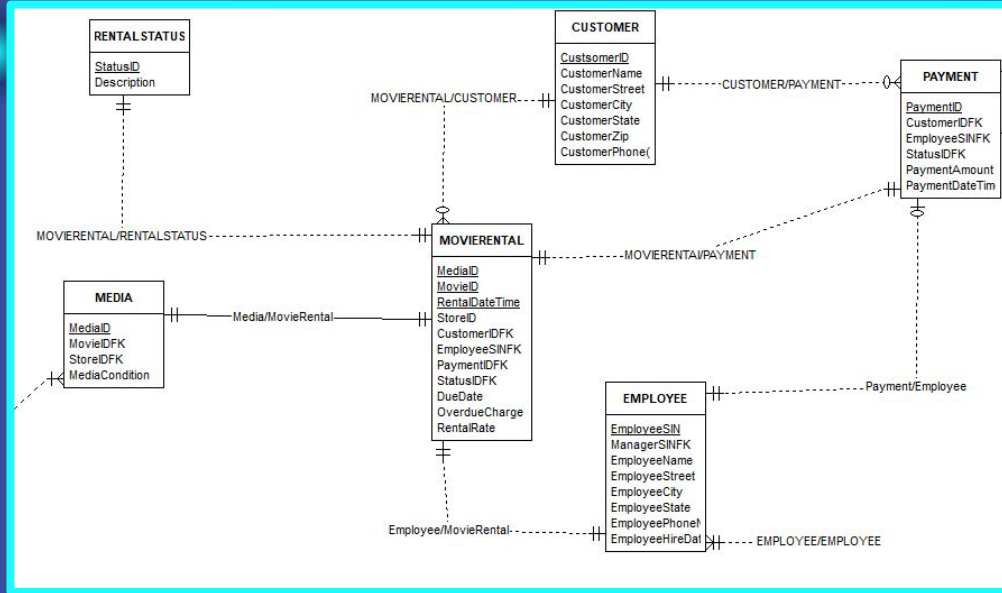
- PAYMENT must have both a CUSTOMER and EMPLOYEE
 - Ex. There cannot be a payment without there first being a customer
- PAYMENT must have PAYMENTSTATUS and PAYMENTTYPE
- The customer must complete the payment with only one payment type
 - Ex. the customer cannot pay half with cash and half with card



06

MOVIE RENTAL

The overall movie rental system within
the database



Assumptions:

- MOVIERENTAL must have PAYMENT, CUSTOMER, EMPLOYEE, RENTALSTATUS, and MEDIA
- A customer can make more than one movie rental



07

SQL

Our implementation of our solution
using MySQL

Implementation with SQL

- Created the database and its entities using SQL
- This includes
 - Creating the tables
 - Primary keys and Foreign Keys
- Used SQL to engineer queries that show that the database is functional

Constraints

- In order to maintain referential integrity constraints, the SQL specifies the constraints of the table
- For most attributes, we did not allow NULL values since NULL values tend to create problems in databases
- Example constraint for PAYMENTTYPE:

```
alter table PAYMENTTYPE
  add constraint PAYMENTTYPE___fk
  foreign key (PaymentID) references PAYMENT(PaymentID)
```

```
USE CASE03_OTMR;
```

```
create table PAYMENTTYPE
```

```
(  
    PaymentID char(30) not null,  
    Description char(30) not null,  
    primary key (PaymentID)  
);
```

```
USE CASE03_OTMR;
```

```
alter table PAYMENTTYPE
```

```
add constraint PAYMENTTYPE___fk  
foreign key (PaymentID) references PAYMENT(PaymentID)
```

```
use CASE03_OTMR;
```

```
create table MOVIE
```

```
(  
    MovieID char(30) not null,  
    MovieTitle char(30) not null,  
    StartName char(30) not null,  
    MovieDescriptor char(30) not null,  
    MovieDirector char(30) not null,  
    Rating char(30) not null,  
    Genre char(30) not null,  
  
    primary key (MovieID)  
);
```

```
USE CASE03_OTMR;
```

```
create table DISTRIBUTOR
```

```
(  
    DistributorID char(30) not null,  
    DistributorName char(30) not null,  
    DistributorCity char(30) not null,  
    DistributorState char(30) not null,  
    DistributorZIP char(30) not null,  
    DistributorPhoneNumber char(30) not null,  
  
    primary key (DistributorID)  
);
```

```
USE CASE03_OTMR;
```

```
create table STOREMOVIE
```

```
(  
    MovieID char(30) not null,  
    StoreID char(30) not null,  
  
    primary key (MovieID, StoreID),  
    foreign key (MovieID) references MOVIE(MovieID),  
    foreign key (StoreID) references STORE(StoreID)  
);
```



08

WEB APP AND REPORTS

The web application used to display
our SQL query reports



09

REGISTER

Application usable by employees to
create and manage transactions



10

CHALLENGES

Problems we encountered through our
process

Challenges Encountered

- Understanding the business rules
- Learning a new library for python to build our register
- Importing data into our database in a efficient manner
- Maintaining referential integrity constraint
- Learning Flask and React frameworks



11

REFLECTION

Overall thoughts on the case study as a whole

Reflection

- Primary design steps help make the whole design process much simpler and more efficient.
 - ERD
- The use of GitHub helped all of us have the same version of the project which streamlined workflow
- Use of Excel sheet helped keep track of everyone's responsibilities and progression of project
 - Communication essential to success as a team
- The case study was very interesting due to its real world applications
- Case study also helped solidify and implement the theory learned in class
- The case study was completed in around 3 weeks

GitHub

- <https://github.com/The1TrueJoe/Database-Case-Studies>
 - <https://github.com/The1TrueJoe/Database-Case-Studies/releases/tag/release1>

References

- Griffith University. (n.d.). *Griffith University Normalization Tool*. Griffith University Normalization Tools.
 - http://www.ict.griffith.edu.au/normalization_tools/normalization/ind.php
- Kroenke, D., Auer, D., Yoder, R., & Vandenberg, S. (2018). *Database Processing: Fundamentals, Design, and Implementation* (15th ed.). Pearson.
 - *Implementation* (15th ed.). Pearson.
- Oracle. (n.d.). *MySQL: MySQL Documentation*. MySQL. <https://dev.mysql.com/doc/>