CPS 510 Assignment 8

Arshpreet Singh (501030338)
James Tan (501042067)
Sania Syed (501017695)

user_acc(<u>Email</u>, FirstName, LastName, DateOfBirth, Password)
{ Email } → FirstName, LastName, DateOfBirth, Password

This relation is in BCNF because the left hand side of the non-trivial, irreducible FD is a super key.

payment_method(<u>Email, PaymentDetails,</u> MethodType)
Step 1: Determining the FDs
FDs:
{ Email, PaymentDetails } → MethodType
{PaymentDetails} → MethodType

Step 2: Breaking RHS (in this case none) and finding redundancies
EP → M: EP+ = {E,P,M} We get M here, therefore the FD: EP → M is redundant.
P → M: P+ = {p} Not redundant.

So our FDs after removing EP → M are : FD = {P → M}.

Step 3: Finding the keys
E is not in the LHS or RHS therefore, it is part of the key.
P is in the LHS but not in the RSH therefore, it is part of the key.
M is not in the LHS but in the RHS therefore, it is not part of the key.
This implies that our key is {E,P}

Step 4: Making the tables
So now we finally have the FD: {P → M}
We make a relation for P → M: R1(P,M).
Since the relation R1 does not have the key, we add another relation for the key.
Therefore, we get 2 relations:
R1(P,M) with FD: P→M
R2(E,P) with no FDs.

These relations are now in 3NF according to Bernstein's algorithm.
These relations are also in BCNF because the left hand side of all these non-trivial, irreducible FDs are super keys.

movie(<u>MovieID</u>, MovieName, Genre, Director, MovieRating, ReleaseYear, Resolution, Runtime, Synopsis, MovieCast)
{ MovieID } → MovieName, Genre, Director, MovieRating, ReleaseYear, Resolution, Runtime, Synopsis, MovieCast

This relation is in BCNF because the left hand side of the non-trivial, irreducible FD is a super key.

orders(<u>Email, MovieID, PaymentDetails</u>)

This is a many-to-many relationship, hence, it has no functional dependencies.

studio(<u>StudioName</u>, ProducedMovies)
{ StudioName } → ProducedMovies

This relation is in BCNF because the left hand side of the non-trivial, irreducible FD is a super key.

produces(<u>MovieID</u>, <u>StudioName</u>)

This is a many-to-many relationship, hence, it has no functional dependencies.

rental(<u>MovieID</u>, RentalPrice, RentalDate, RentalExpiry)
{ MovieID } → RentalPrice, RentalDate, RentalExpiry

This relation is in BCNF because the left hand side of the non-trivial, irreducible FD is a super key.

purchase(<u>MovieID</u>, BuyingPrice, PurchaseDate)
{ MovieID } → BuyingPrice, PurchaseDate

This relation is in BCNF because the left hand side of the non-trivial, irreducible FD is a super key.

movie_review(<u>RevID</u>, WrittenRev, NumericRating, Email, FirstName, LastName)
Step 1: Determining the FDs
FDs:
{RevID} → WrittenRev, NumericRating, Email, FirstName, LastName
{Email} → FirstName, LastName

Step 2: Breaking RHS into single attributes and finding redundancies
FD = {RevID → WrittenRev, RevID → NumericRating, RevID → Email, RevID → FirstName, RevID → LastName, Email → FirstName, Email → LastName}

RevID → WrittenRev: RevID+ = {RevID, NumericRating, Email, FirstName, LastName}
Not redundant
RevID → NumericRating: RevID+ = {RevID, WrittenRev, Email, FirstName, LastName}
Not redundant
RevID → Email: RevID+ = {RevID, WrittenRev, NumericRating, FirstName, LastName}
Not redundant
RevID → FirstName: RevID+ = {RevID, WrittenRev, NumericRating, Email, FirstName, LastName}
We get FirstName, therefore the FD RevID → FirstName is redundant.
RevID → LastName: RevID+ = {RevID, WrittenRev, NumericRating, Email, FirstName, LastName}
We get LastName, therefore the FD RevID → LastName is redundant.
Email → FirstName: Email+ = {Email, LastName}
Not redundant
Email → LastName: Email+ = {Email, FirstName}
Not redundant

We remove the 2 redundant FDs which leaves
FD = {RevID → WrittenRev, RevID → NumericRating, RevID → Email, Email → FirstName, Email → LastName}

Step 3: Finding the keys
RevID is in the LHS but not in the RHS therefore, it is part of the key.
WrittenRev, NumericRating, FirstName and LastName are not in the LHS but in the RHS therefore, they are not part of the key.
Therefore {RevID} should be part of any key of relations R.
We now test keys by combining the rest of the attributes.

RevID+ = {RevID, WrittenRev, NumericRating, Email}
This is not the key since we do not get all the attributes.
{RevID, Email}+ = {RevID, WrittenRev, NumericRating, Email, FirstName, LastName}
This is the key since we get all the attributes.

Step 4: Making the tables
We combine the FDs with same LHS':
RevID → WrittenRev, NumericRating, Email
Email → FirstName, LastName

We now form the following relations:
R1(RevID, WrittenRev, NumericRating, Email)
R2(Email, FirstName, LastName)
We do not need to add another relation for the key since R1 already has RevID,Email.

These relations are now in 3NF according to Bernstein's algorithm.
These relations are also in BCNF because the left hand side of all these non-trivial, irreducible FDs are super keys.

fanclub_membership(<u>MembershipID</u>, RentalDiscount, PurchaseDiscount, RentalSpecialOffers, PurchaseSpecialOffers, OffersRentalStartDate, OffersRentalEndDate, OffersPurchaseStartDate, OffersPurchaseEndDate)
{ MembershipID } → RentalDiscount, PurchaseDiscount, RentalSpecialOffers, PurchaseSpecialOffers, OffersRentalStartDate, OffersRentalEndDate, OffersPurchaseStartDate, OffersPurchaseEndDate

This relation is in BCNF because the left hand side of the non-trivial, irreducible FD is a super key.

has(<u>MembershipID, StudioName</u>)

This is a many-to-many relationship, hence, it has no functional dependencies.