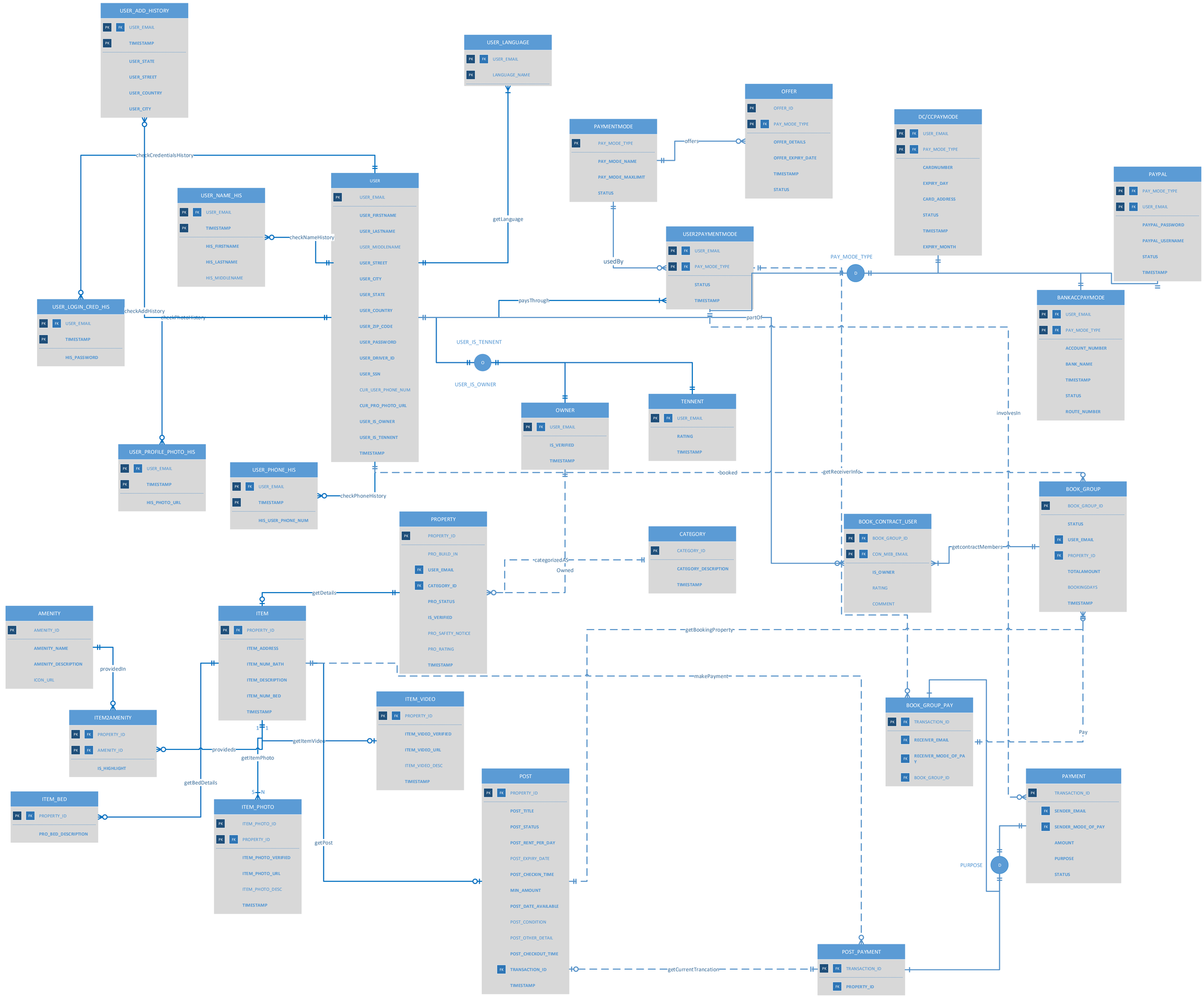
USER_ADD_HISTORY
PK FK USER_EMAIL
PK TIMESTAMP
USER_STATE
USER_STREET
USER_COUNTRY
USER_CITY
USER_LANGUAGE
PK FK USER_EMAIL
PK LANGUAGE_NAME
checkCredentialsHistory
USER_NAME_HIS
PK FK USER_EMAIL
PK TIMESTAMP
HIS_FIRSTNAME
HIS_LASTNAME
HIS_MIDDLENAME
checkNameHistory
USER
PK USER_EMAIL
USER_FIRSTNAME
USER_LASTNAME
USER_MIDDLENAME
USER_STREET
USER_CITY
USER_STATE
USER_COUNTRY
USER_ZIP_CODE
USER_PASSWORD
USER_DRIVER_ID
USER_SSN
CUR_USER_PHONE_NUM
CUR_PRO_PHOTO_URL
USER_IS_OWNER
USER_IS_TENNENT
TIMESTAMP
getLanguage
PAYMENTMODE
PK PAY_MODE_TYPE
PAY_MODE_NAME
PAY_MODE_MAXLIMIT
STATUS
offers
OFFER
PK OFFER_ID
PK FK PAY_MODE_TYPE
OFFER_DETAILS
OFFER_EXPIRY_DATE
TIMESTAMP
STATUS
DC/CCPAYMODE
PK FK USER_EMAIL
PK FK PAY_MODE_TYPE
CARDNUMBER
EXPIRY_DAY
CARD_ADDRESS
STATUS
TIMESTAMP
EXPIRY_MONTH
PAYPAL
PK FK PAY_MODE_TYPE
PK FK USER_EMAIL
PAYPAL_PASSWORD
PAYPAL_USERNAME
STATUS
TIMESTAMP
usedBy
USER2PAYMENTMODE
PK FK USER_EMAIL
PK FK PAY_MODE_TYPE
STATUS
TIMESTAMP
PAY_MODE_TYPE
D
BANKACCPAYMODE
PK FK USER_EMAIL
PK FK PAY_MODE_TYPE
ACCOUNT_NUMBER
BANK_NAME
TIMESTAMP
STATUS
ROUTE_NUMBER
paysThrough
partOf
USER_IS_TENNENT
O
USER_IS_OWNER
OWNER
PK FK USER_EMAIL
IS_VERIFIED
TIMESTAMP
TENNENT
PK FK USER_EMAIL
RATING
TIMESTAMP
involvesIn
checkAddHistory
checkPhotoHistory
USER_LOGIN_CRED_HIS
PK FK USER_EMAIL
PK TIMESTAMP
HIS_PASSWORD
USER_PROFILE_PHOTO_HIS
PK FK USER_EMAIL
PK TIMESTAMP
HIS_PHOTO_URL
USER_PHONE_HIS
PK FK USER_EMAIL
PK TIMESTAMP
HIS_USER_PHONE_NUM
checkPhoneHistory
booked
getReceiverInfo
BOOK_GROUP
PK BOOK_GROUP_ID
STATUS
FK USER_EMAIL
FK PROPERTY_ID
TOTALAMOUNT
BOOKINGDAYS
TIMESTAMP
PROPERTY
PK PROPERTY_ID
PRO_BUILD_IN
FK USER_EMAIL
FK CATEGORY_ID
PRO_STATUS
IS_VERIFIED
PRO_SAFETY_NOTICE
PRO_RATING
TIMESTAMP
categorizedAS
Owned
CATEGORY
PK CATEGORY_ID
CATEGORY_DESCRIPTION
TIMESTAMP
BOOK_CONTRACT_USER
PK FK BOOK_GROUP_ID
PK FK CON_MEB_EMAIL
IS_OWNER
RATING
COMMENT
getcontractMembers
getDetails
getBookingProperty
makePayment
AMENITY
PK AMENITY_ID
AMENITY_NAME
AMENITY_DESCRIPTION
ICON_URL
providedIn
ITEM
PK FK PROPERTY_ID
ITEM_ADDRESS
ITEM_NUM_BATH
ITEM_DESCRIPTION
ITEM_NUM_BED
TIMESTAMP
1
1
ITEM2AMENITY
PK FK PROPERTY_ID
PK FK AMENITY_ID
IS_HIGHLIGHT
provideds
getItemVideo
getItemPhoto
ITEM_VIDEO
PK FK PROPERTY_ID
ITEM_VIDEO_VERIFIED
ITEM_VIDEO_URL
ITEM_VIDEO_DESC
TIMESTAMP
BOOK_GROUP_PAY
PK FK TRANSACTION_ID
FK RECEIVER_EMAIL
FK RECEIVER_MODE_OF_PAY
FK BOOK_GROUP_ID
Pay
PAYMENT
PK TRANSACTION_ID
FK SENDER_EMAIL
FK SENDER_MODE_OF_PAY
AMOUNT
PURPOSE
STATUS
PURPOSE
D
getBedDetails
ITEM_BED
PK FK PROPERTY_ID
PRO_BED_DESCRIPTION
5
N
ITEM_PHOTO
PK ITEM_PHOTO_ID
PK FK PROPERTY_ID
ITEM_PHOTO_VERIFIED
ITEM_PHOTO_URL
ITEM_PHOTO_DESC
TIMESTAMP
getPost
POST
PK FK PROPERTY_ID
POST_TITLE
POST_STATUS
POST_RENT_PER_DAY
POST_EXPIRY_DATE
POST_CHECKIN_TIME
MIN_AMOUNT
POST_DATE_AVAILABLE
POST_CONDITION
POST_OTHER_DETAIL
POST_CHECKOUT_TIME
FK TRANSACTION_ID
TIMESTAMP
getCurrentTrancation
POST_PAYMENT
PK FK TRANSACTION_ID
FK PROPERTY_ID

ASSUMTION/DESIGN CHOICES

- I make specialized relationship by circle and relationship complete by 1:1 and single by one arrow on one side as there is no other option available.
- Each TIMESTAMP attribute involves CREATED_AT, UPDATED_AT option.
- USER can upload at most one video of property. (as it will acquire lot of memory on the server side)
- Make PROPERTY_ID surrogate key because not other attribute can be act as key address is good can be consider but sometime street name change so it become invalid
- MAKE BOOK_GROUP_ID as surrogate key because BOOK_GROUP not completely depend on Tennent and can be used later to get payment history that's why decided not to make weak entity.
- I have introduced redundancy in payment table because payment details can be huge so to track how much payment has been done by user or to which user how much money has been received by user involves lot of joins so to increase the speed of searching I introduce redundancy. This increase size but we can search the table without involving other tables.
- We can introduce SLOT table to track the date available for the property to rent or we can use AVAILABLEDATE attribute which can be in JSON or STRING which can be updated as soon user booked for rent. I go with attribute one as it will not cause me to create new table and from using joins.
  JSON can be of form **{{date, status,book_group_id}}** or we can create extra table (SLOT)for it which have 1:M relation with the BOOK_GROUP.
- ASSUME that there can be at max one post for the property weather it is valid or invalid there can be at max only one post. When we pay for it again to repost it can check weather it record of property already exist or not. If exist then just update the TransactionID.
- Assume the amenity table containing list of all amenities property may have. This is to just increase the consistency of term at application level and if we want to add some icon with that amenity we can do that easily.
- Allow any user to book property of owner for anyone as he will be paying for the property so it doesn't matter who will book property for whome.

OPTIONAL/ADDITIONAL VALUE/ADDITIONAL ENTITY

- Assuming we have to store history of user such as password, profile photo history, name, phone_number, address as it can be used for business purpose later on. All History storing table are linked with 1:M relationship.
- Added details to save information of user account in tables such as DC/CCPAYMODE, BANKACCPAYMODE, PAYPAL.
- Added offer table assuming some paymentmode offer users to entice to used that.
- used USER_LANGUAGE to track detail of user language as it can be helpful detail for user.
- In USER table USER_MIDDLENAME is optional as not many users have middle name but can be used filter purpose.
- USER_PASSWORD can be used for login
- CUR_USER_PHONE_NUM is optional as it is not mention in the business rule it can act as a secondary key to get OTP to reset the account.

- PRO_BUILD_IN, PRO_SAFETY_NOTICE, PRO_RATING as they might be not present for property but essential to help to entice Tennent.
- STATUS, VERIFIED, RATING(DERIVED for PROPERTY, OWNER, TENNANT) as they make improve user experience, act as validate for database engineer and require less time to get results for rating of property and OWNER, TENNANT can be used immediately without any tedious calculation. Also it allow another validation to show relevant result to user as well as owner and also give them additional functionality like to deactivate the property from renting during renovation etc
- POST_CHECKIN_TIME, POST_CHECKOUT_TIME (ALL these can be helpful for users to get Idea of timing) , MIN_AMOUNT(OWNER) to pay min amount to publish ad, PAY_MODE_MAX_LIMIT is to track what user can pay at max using that paymentMode.
- In ITEM_TABLE is added bed_description of bed as to give more information to user.
- TOTALAMOUNT in BOOK_GROUP table is just to track total amount for booking, It is redundant as it can be calculated by checking POST_DATE_AVAILABLE but I will require computation so to save computation overhead introduced little redundancy.
- RATING and COMMENT in BOOK_CONTRACT_USER is not mandatory as they are not mandatory by business rule.

MULTIVALUED ATTRIBUTE

1) In user table user language is multivalued attribute as user can speak many language which can be used to entice people who can't spoken other language to visit the palace where owner is familiar with their language so that they can a pleasant conversation.

RELATIONSHIPS

- **USER and (USER_NAME_HIS, USER_LOGIN_CRED_HIS, USER_PROFILE_PHOTO_HIS, USER_PHONE_HIS, USER_ADD_HISTORY)** - all have 1:1 – 0:M relationship. As they need to maintain past user details. If user change his name, address, phone all details will be saved so it may that user do not change so then history will be empty for that user. That's why 0:M but each history belong to each user so 1:1 with user. And they are **STRONG RELATIONSHIP** as the part of primary key from is formed from primary key of other table and user history is valid if we have user by context.
- **USER and USER_LANGUAGE** has 1-M relationship as user can used many languages but he know at least one.
- **PAYMENTMODE and OFFER** have 0: M as payment mode can have multiple payment mode and can have none also but offer must attach to one payment mode (as company offers on particular payment mode)
- **PAYMENTMODE and USER2PAYMENTMODE** has 0:M as PAYMENTMODE can be used by many user or not used at all
- **USER and USER2PAYMENTMODE** has 1:M reason as user must have at least one payment option.

- **OWNER and PROPERTY** has 0:M as owner can have many and at a time can have zero property.
- **PROPERTY and Items** are in zero and one relationship as each item provide details of property but for each property it is not mandatory have item maybe it is new property. It is strong relationship as ITEM existence do not matter without PROPERTY.
- **ITEM and AMENITY** table are linked with many to many relationship so I implement using **ITEM2AMENITY** bridge with 1:M relationship between **ITEM and ITEM2AMENITY** and **AMENITY and ITEM2AMENITY** table to show which amenity are available. I do not make a multivalued attribute because I want to make user experience better as there are option available their description and icons if we save it then it can be consistent across all property. ALL OTHER options like pet allowed, wifi available are included in amenities.
- **ITEM, ITEM_BED, ITEM_PHOTO, ITEM_VIDEO** are all extra details for the property. **ITEM and ITEM_PHOTO** photo has 1 to many relationship as shown by cardinality mention in business rules.
- **ITEM and ITEM_VIDEO** has zero and at most 1 relationship as item can have zero and at most 1 video.
- **CATERGORY and PROPERTY** have 1:M as many property belong to same category.
- **ITEM AND POST** have one to one relationship as ITEM can have zero post but post must have item. It is strong relationship as POST cannot exist without ITEM.
- **ITEM and POST_PAYMENT** have one (one to many). ONE to many because there can be many transactions for a payment to create post or to activate it again. It has lower limit zero because ITEM can be exist if there is no payment.
  And one to one is to update the current transaction on the property and update the expiry date depending on the payment. Duplicate the TRANSACTION_ID in post to separate the post payment history from the current one.
- **USER and BOOK_GROUP** has 1:M relationship as User can book many times and have min cardinality zero because there can be tenant who never booked.
- **POST and BOOK_GROUP** has 1:M relationship as each post can belong multiple BOOK_GROUP. It has minimum cardinality 0 because post can not belong to any BOOK_GROUP. ALSO as soon as user booked the POST_AVAILABLE_DATE in post updated accordingly.
- **BOOK_GROUP and USER** have M:M as each BOOK_GROUP involves two users and a single user can belong to multiple BOOK_GROUP to achieve this we bring the bridge table called **BOOK_CONTRACT_USER**
- **BOOK_GROUP and BOOK_CONTRACT_USER** have 1 to Many relationship as Contract contain two people owner and renter also this renter can be different from the one who is paying for rent. Thus allowing option to book for some one else also prevent owner to be the renter of his property.
- **USER and BOOK_CONTRACT_USER** have 1:M as both owner and renter belong to USER table. It also store rating and any comments given by member to another.
- **BOOK_GROUP_PAY and BOOK_GROUP**_PAY has 1:1 with minimum cardinality zero as there can be bookgroup with no transaction Id just as pending options for users to later book.
- **USER2PAYMENTMODE and PAYMENT** has 1:M relationship as SENDER_EMAIL, SENDER_MODE_OF_PAY and RECEIVER_EMAIL, RECEIVER_MODE_OF_PAY all belong to USER2PAYMENTMODE and many times users can involve.in payment

- **USER2PAYMENTMODE and BOOK_GROUP_PAY** has 1:M relationship as one user can be part of many receiving payment.

SPECIALIZATION HIERARCHY

- USER2PAYMENTMODE is further divided into three parts based on PAY_MODE_TYPE attribute. It will be disjoint and total completeness (mention as double line)
- USER further divided as OWNER, TENNENT (RENTER) they have overlapping and total completeness as user can be both in owner and renter table. USER_IS_OWNER, USER_IS_TENNENT are used as subtype discriminator. It has total completeness constraint.
- PAYMENT entity further divided into two parts as POST_PAYMENT(payment by owner for post(property) and BOOK_GROUP_PAY for payment for rent (between Tennant and Owner). . It has total completeness constraint as each payment belong to two types.

LIMITATION

1) USER CAN'T CHANGE THE EMAIL ADDRESS. IT IS UNIQUE(as it used as primary key)
2) USER can have right now only three payment modes and can save details of one account in these three-payment mode not multiple (BANK, CARD, PAYPAL).
3) THERE is at most one OWNER of the property.
4) Restrict someone to post at application level whose difference in start day for sublease and max_available_days_booking option is less than seven days.
5) Required application support to enter correct details like country, state, zip code etc.
6) One slot can be book by one person. And after successful transaction need to update status of book_group and update available date in POST table. for that depend on application to enter correct details it can be done by adding table slot but there we need application support to include slots of one property only. So by analyzing tradeoff I go with the shown approach.