

DATABASE SYSTEM DESIGN

ER DIAGRAM OF ONLINE RENTAL SERVICE(HW1)

In sHome online rental service, we have following entities with their relationship with other entities:

- (a) *User*: We'll have a primary key as *user_Id*, a sequential number because we cannot use driving license number as a primary (assuming not unique across the states). Moreover, we cannot use social security number as a primary because this is very crucial information and should be stored in the encrypted form. Email id column should have unique constraint which will disallow user to register again with same email id. We cannot use email as primary key because one user can have multiple emails ids for different business purposes.
- (b) For the scaled system of sHome, we can create a hierarchy version of the address field like Address -> ZIP Code-> City -> Country etc. which will be really helpful for creating a search engine. User address and property address are two different attributes where we are giving importance to property address rather than user address.
- (c) Because we have two types of users (*Renter* and *Owner*), we'll use user hierarchy named as U_TYPE and both the users are categorise as U_IS_OWNER and U_IS_RENTER. Both the entities (*Renter* and *Owner* will have *user_Id* as primary key and foreign key as well). User can have only address which can be same or different than property location.
- (d) *User-Account* relationship: A user must have an account with primary key as *user_Id* and same as a foreign key from User entity. An account can have multiple payment options with minimum cardinality of one.
- (e) An *Owner* can have zero or multiple properties(*items*). Since *User* to *Item* is weak relationship, we'll have a surrogate key(*property_Id*) as primary key for the *item* entity and *owner* id(*user_Id*) as foreign key referenced from *User* table.
- (f) Each *item* entry will have a unique id defined as *property_Id*. Moreover, *item* entity will have four foreign keys (1) *userId*(*Owner Id*), (2) *AddressId*, (3) apartment configuration id(*apt_config_Id*) and (4) Review Id(*review_id*) for apartment reviews.
- (g) For *apt_config*, we'll have a separate entity named as floor plans(*floor_plan*) which internally subsume many entities like *amenities*, media details(*media_details*). Each floor plan will be referenced to an *apt_config* entity. Each floor plan must have exactly one amenities and media details entity. Since *amenities* entity is using *floor_plan_Id* and primary key referenced back to floor plan entity, the relationships between these two are strong.
- (h) In floor plan, room category will be multi values attribute define as *category_small*, *category_large*, *category_medium* etc.

- (i) Each apartment can have zero or more post depending upon lease/available dates for each post, we'll have a *post_id* as primary key and *property_id* as foreign key referenced back to *item* with a weak relationship.
- (j) The relationship between payment and post can be M:N which has been handled by middle entity *contract and reservation*. Each post will have a direct entry in the *payment* table while posting a post on portal, *owner* has pay some amount for the post. payment entry will have *post_id* as a foreign key.
- (k) A *renter* can rent zero or any number of properties on different time span. While renting an apartment, renter has to sign a *contract* and has to be bind with some rules so a contract entity will be middle entity between a post and payment entities.
- (l) By making above mentioned relationship, we are making sure that only *renter* is signing the *contract* and paying the rent. One *renter* can sign zero or multiple *contracts*. Each contract will be identified its own primary key not by *user_id* so the relationship between *contract* and *renter* is weak. The contract will have *user_id* (renter's id), *post_id*, *payment_id* as foreign keys.
- (m) For each *contract*, there will be only one reservation/*payment* id However, each post can have multiple *contracts* based renter's choice of bookings and booking time span.
- (n) We can implement SQL triggers to update the availability of the post, activeness, number of days left etc. Moreover we can implement dynamic pricing if we are reaching towards sold out case. Addition, SQL triggers can help to activate to deactivate the post if it has passed the mentioned deadline.
- (o) Each *item* can have zero or one *review* from one *user* for different bookings/*reservations*. Moreover, each *item* can have at least one or multiple *apartments configuration*(*apt_configuration*) like penthouse, studios etc, therefor, the *apt_configuration* entity will have its own primary and *property_id* as foreign key relationship between item and *apartment configuration* is weak.
- (p) Once reservation date is over, renter can post reviews. Review entity uses *reservation Id*, *property_id*, *owner_id* as foreign keys. We need reservation Id to make sure that only valid renters review the property and owner id to make sure that owner is able to answer back the posted reviews, Hence the relationship between review and other referenced entities like payment, property is weak. One reservation can have minimum zero and at most one review.

Summary of the relationships:

- One Owner can have zero or multiple properties at different locations.
- One property can have more than one apartment configuration.
- One apartment can have exactly one type of amenities.
- One apartment can have exactly same media details like photos and video link because media details are photos and videos of building structure and room size which are static.
- One floor plan or apartment configuration can have zero or multiple posts from same owner.
- One post can have multiple contract with same user over different time span but different reservation and payment details. Item post activeness and apartment availability changes based on last leased dates on the post.
- One contract has exactly one payment/reservation.