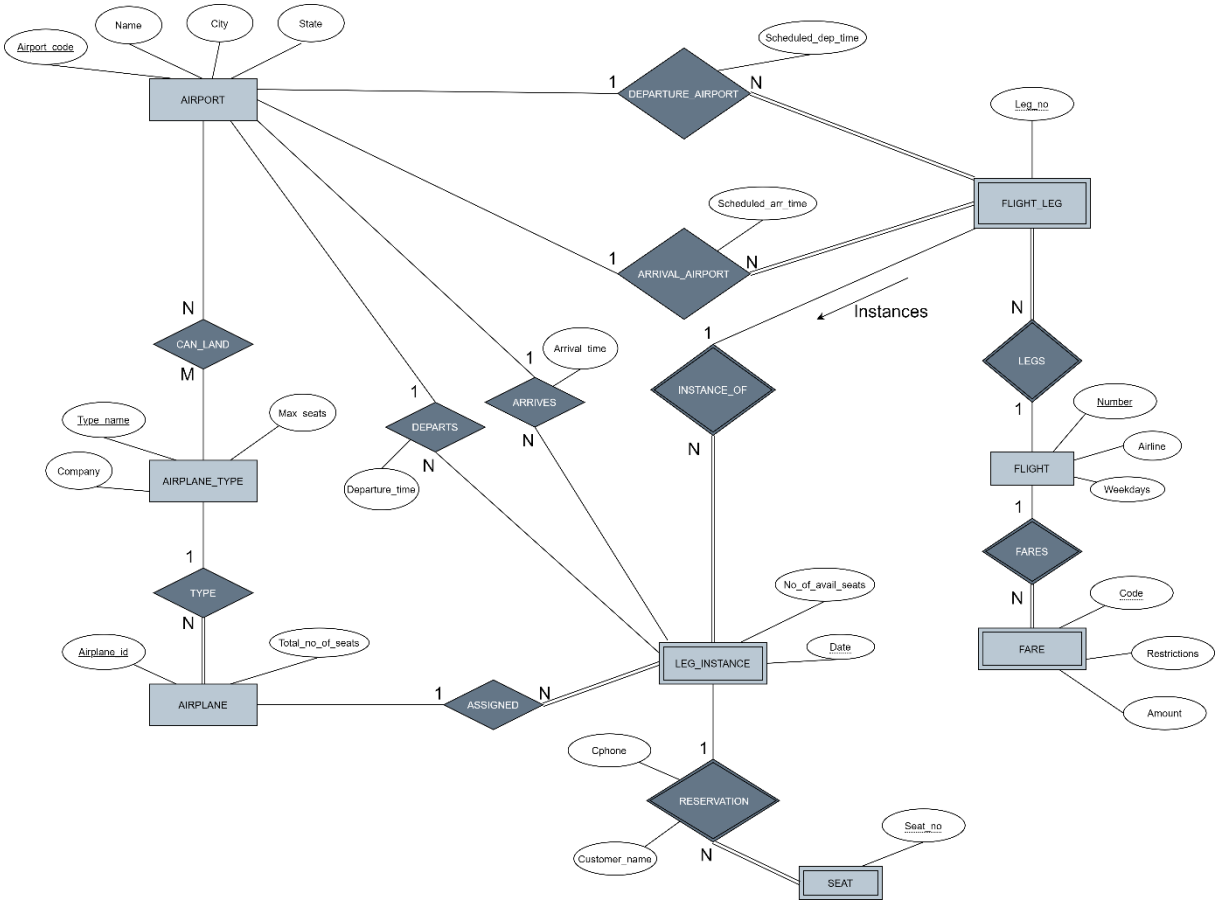
DATABASE MANAGEMENT
MIDTERM GROUP-1

EGE UNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ

MEHMET ANIL TAYSİ, 05170000022
TARKAN YAMAN, 05170000026
ENES ALPER BALTA, 05170000021
MUHAMMED NAIM YUSUFI, 05050007596

İçindekiler

1. Relational Schema'nın EER Diagramına Dönüştürülmesi



Şekil ***İlk Kısım için EER Diagramı***

1.1 Varlıkların açıklanması ve İlişki Özellikleri

AIRPORT : İcinde havalimanın sehir, eyalet, isim ve havalimanı kodu içeriyor. Ayrıca AIRPORT, bağlı olduğu tüm ilişkilerde strong entity olarak davranır. Zaten strong olarak tanımlanmıştır.

N AIRPORT'a M tane AIRPLANE_TYPE varlığı iniş yapabilir (CAN_LAND).
1 AIRPORT, N tane FLIGHT_LEG için İniş Havalimanı özelliği taşıyabilir.
1 AIRPORT, N tane FLIGHT_LEG için Kalkış Havalimanı özelliği taşıyabilir.
1 adet AIRPORT'tan, N adet LEG_INSTANCE'a ait uçak kalkabilir.
1 adet AIRPORT'a, N adet LEG_INSTANCE'a ait uçak iniş yapabilir.

AIRPLANE_TYPE : Uçak tiplerin adını, maksimum kaç koltuk tutuklarıyla ilgili bilgi tutuyor. Ayrıca ikinci kısımda entity olarak ekledikten sonra kalkacak Company attribute'u mevcuttur.

N tane AIRPLANE'in 1 tane AIRPLANE_TYPE varlığında tipi olmak zorundadır.

AIRPLANE: Uçağın ID'sini ve toplam koltuk sayısını tutuyor.

1 tane AIRPLANE, N tane LEG_INSTANCE'a atanmış olabilir. Ters taraftan bakacak olursak, N adet içinden her bir LEG_INSTANCE varlığına, bir adet uçağın atanması şarttır.

FLIGHT: Uçuş numarasını ve hangi günlerde uçtuğu ile ilgili bilgisini tutuyor. Ve 2. Bölümde eklenecek airline özelliği de burada tutuluyor.

Her 1 adet FLIGHT varlığı için, N adet FARE varlığı üzerinden ücretlerin elde edilmesi gerekiyor. Her 1 adet FLIGHT varlığı, N tane FLIGHT_LEG'e bölünebilir.

FARE: Uçuşla ilgili fiyat bilgisini, ödeme işlem kodunu ve kısıtlamalarını gösteriyor.

N adet FARE varlığının, 1 FLIGHT'ın ücret bilgilerini içermesi zorunludur, dolayısıyla eğer en az bir adet FLIGHT var ise, FARE varlığı da tanımlanmak zorundadır, bu da FARE'ı weak entity yapar.

FLIGHT_LEG: İlgili uçuşun bacağını sayısını tutuyor.

Eğer 1 FLIGHT tanımlanmışsa, FLIGHT_LEG tanımı mutlaka gereklidir. Bir uçuşun tek bacağı da olabilir (aktarmasız).

LEG_INSTANCE: Uçuş ile ilgili tarihini ve boş olan koltuk sayısını tutuyor.

EER diagramına baktığımızda zaten INSTANCE_OF ilişkisi ile bağlı olduklarından, LEG_ISNTANCE'a weak entity, FLIGHT_LEG'e de LEG_INSTANCE'a göre strong entity diyebiliriz (her ne kadar weak entity olarak tanımlansada). LEG_INSTANCE varlığının oluşturulabilmesi için öncelikle FLIGHT_LEG tanımlanmalıdır.

SEAT: Bu varlık, RESERVATION adlı weak relation üzerinden, LEG_INSTANCE ile bağlantı kurar. Üzerinde Seat_no (partial key), Customer_name, ve Cphone attribute'ları vardır. CUSTOMER entity'sini ikinci bölümde eklediğimizde, bu özellikleri de oraya aktaracağız.

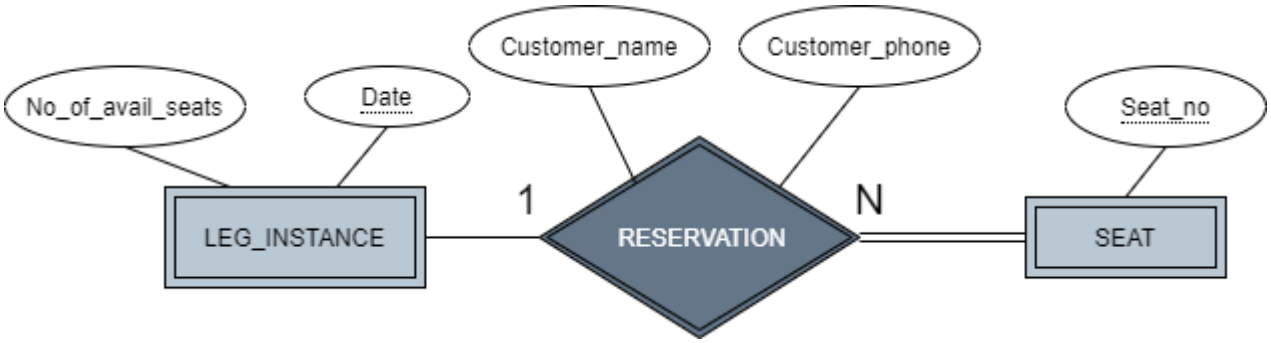
SEAT weak entity'dir, ancak ve ancak LEG_INSTANCE varlığı oluşturulsa meydana gelebilir, bu durum LEG_INSTANCE varlığını SEAT'e göre strong entity yapar. 1 LEG_INSTANCE için N tane koltuk rezervasyonu yapılabilir.

Bu bölüm, hem diagramı tanıma açısından, hem de midterm dökünanında bulunan 'Abstractions of Classification' bölümünü gerçekleştirmek için yapılmıştır, sonradan eklenen kısımların tipleri ve özellikleri, ilgili bölümlerde zaten tanımlanmıştır.

2.EER Diagram'ının Genişletilmesi

2.1 CUSTOMER Varlığı eklenmesi ve gerçekleştirimi

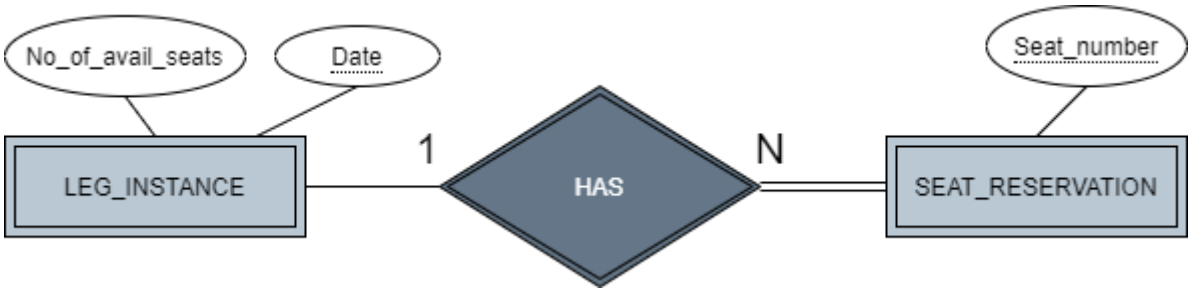
CUSTOMER varlığı ile ilgili işlemlere başlamadan önce tasarımımızı gerçekleştirmek için ER diagramımızı optimize etmek amacıyla birkaç adım gerçekleştirdik. ER diagramımızdaki LEG_INSTANCE varlığı ve SEAT varlığı arasında bulunan RESERVATION ilişkisini(Şekil 1) gerçekleştireceğimiz tasarımımıza göre yeniden düzenlemek istedik.



Şekil 1. *LEG_INSTANCE ve SEAT arasındaki RESERVATION ilişkisi*

- Şekil 1 üzerinde gösterilen ilişki için 'Bir LEG_INSTANCE içerisinde N kadar SEAT, RESERVATION edilebilir.' ve 'Bir SEAT sadece bir LEG_INSTANCE içerisinde RESERVATION edilmiş olmak zorundadır.' gereksinimleri söylenebilir.
- SEAT varlığı ve RESERVATION ilişkisi ile ayrılmış olan SEAT_RESERVATION tablosunu, tek bir varlık altında toplamak amacı ile SEAT_RESERVATION adında bir varlık oluşturduk.
- Buradaki amacımızı detaylandırırsak; bir uçuş ayağında gerçekleşen rezerve işlemi ile ilgili koltuk bilgilerini sadece SEAT varlığı üzerinden, müşteri ile ilgili bilgileri ise sadece RESERVATION ilişkisi üzerinden elde edebiliyorduk. Bu ayrımı kaldırarak müşteri bilgileri ile düzenli çalışabilecek bir yapı oluşturmak istedik. Bu işlemi de bize SEAT_RESERVATION olarak verilen tabloyu direkt olarak bir varlık olacak şekilde tanımlayarak gerçekleştirdik. Bu sayede tablo içerisinde verilen tüm özellikler tanımladığımız varlık içerisinden ulaşılabilir olacaktır. Ayrıca bir uçuş ayağında rezerve edilen koltukların bilgileriyle birlikte müşterilerin de bilgileri tutulacak, böylelikle daha sonra tasarımımıza eklenecek olan CUSTOMER varlığıyla birlikte optimize olarak çalışan bir yapı ortaya koyabileceğiz.
- Oluşturduğumuz SEAT_RESERVATION adlı varlığını LEG_INSTANCE varlığı ile ilişkilendirebilmek için 'Bir LEG_INSTANCE içerisinde N kadar SEAT_RESERVATION'a sahip olabilir.' Ve 'Bir SEAT_RESERVATION sadece bir LEG_INSTANCE içerisinde

yapılmak zorundadır.' Gereksinimleri dahilinde Şekil 2'deki HAS ilişkisini oluşturduk. HAS ilişkisinin notasyonlarını belirler iken, 'Bir LEG_INSTANCE içerisinde SEAT_RESERVATION yapılmış olabilir ya da hiç yapılmamış da olabilir.' Ve 'Eğer bir SEAT_RESERVATION yapılmış ise bu rezervasyon mutlaka bir LEG_INSTANCE üzerinde yapılmış olması gerekiyor.' Olasılıklarına dayanarak, LEG_INSTANCE – HAS ilişkisini partial ve HAS – SEAT_RESERVATION ilişkisini ise total olarak belirledik.

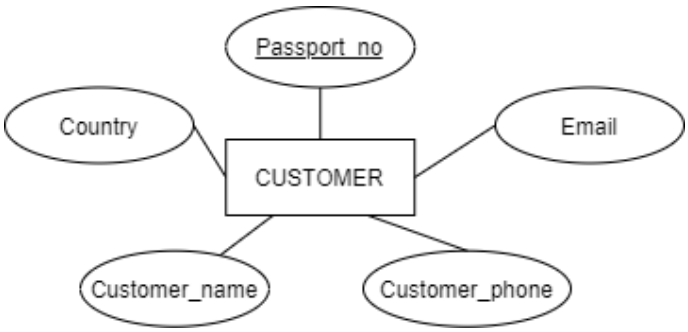


Şekil 2. *LEG_INSTANCE ve SEAT_RESERVATION arasındaki HAS ilişkisi*

- Bir uçuş ayağı bulunmadan bir koltuk rezervasyonu gerçekleştirilemez. Koltuk rezervasyonunun gerçekleştirilebilmesi için mutlaka bir uçuş ayağının bulunması gerekir. LEG_INSTANCE varlığının bulunmasına bağlı olan SEAT_RESERVATION, bu sebeple LEG_INSTANCE varlığına göre zayıf bir varlık (weak entitiy) durumundadır. Bu sebeple HAS ilişkisi de zayıf bir ilişkidir(weak relation).

Bu aşamadan itibaren CUSTOMER varlığını ekleme işlemine geçtik.

- CUSTOMER varlığı, müşteriyi temsil etmektedir.
- SEAT_RESERVATION içerisinde bulunan Customer_name ve Customer_phone özelliklerini ayırarak CUSTOMER varlığına aktarmamız ve e-mail, adress, country, passport number gibi özellikler ile de varlığımızı genişletmemiz isteniyor.
- Öncelikle CUSTOMER varlığını oluşturuyoruz. Daha sonrasında bizden istenen özellikleri CUSTOMER varlığına tanımlıyoruz. CUSTOMER varlığını oluştururken varlığımızı regular olarak oluşturuyoruz. Çünkü bir müşteri her şeyden bağımsız olarak sistemde var olabilir.

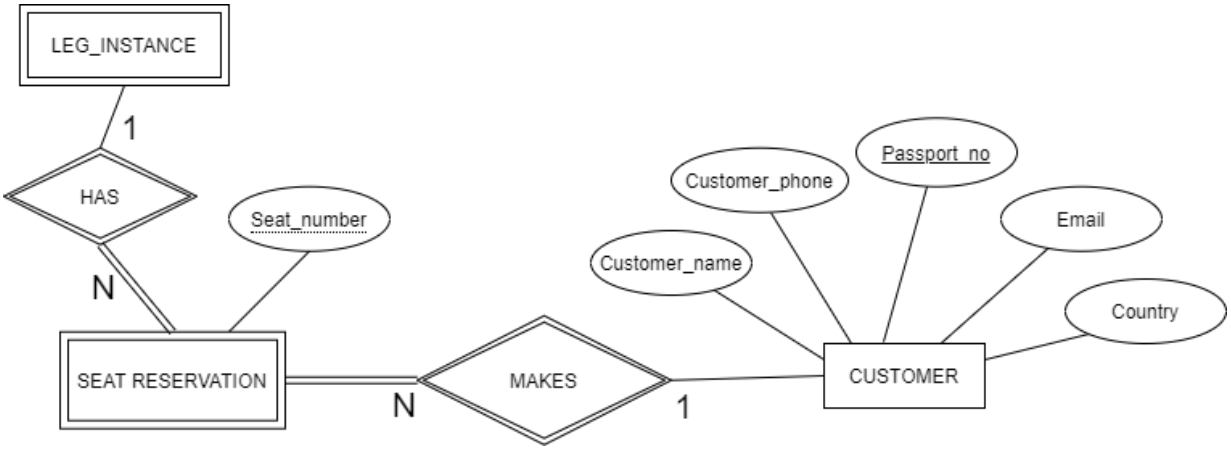


Şekil 3. *CUSTOMER varlığı ve özellikleri*

- Özellikleri tanımlar iken Passport Number özelliğini Primary Key olarak tanımlıyoruz. Çünkü her bir müşterinin pasaport numarası kendisine özgü bir değer(unique) olduğu için priamary key olarak belirliyoruz.
- CUSTOMER ve SEAT_RESERVATION varlıkları arasındaki gereksinimler ile ilgili
    - Bir müşteri birden fazla koltuk rezervasyonu yapabilir.
    - Bir koltuk rezervasyonu sadece ve sadece bir müşteri tarafından yapılmak zorundadır.
- Bu gereksinimler ışığında CUSTOMER ve SEAT_RESERVATION varlıkları arasında MAKES adlı bir ilişki tanımlıyoruz.
- İlk gereksinimde belirtilen bir müşterinin birden fazla koltuk rezervasyonu gerçekleştirmek için SEAT_RESERVATION varlığının notasyonunu N olarak belirleriz. Bir zorunluluk belirtilmediği için de CUSTOMER varlığının ilişkisini partial olarak belirleriz.
- İkinci gereksinimde belirtilen bir koltuk rezervasyonunun sadece bir müşteri tarafından yapılmak zorundadır ibaresi için CUSTOMER varlığının notasyonunu 1 olarak belirledik. Zorunluluk belirtildiği için ise SEAT_RESERVATION ilişkini total olarak belirtiriz.
- Bir CUSTOMER tarafından MAKES işlemi yapılmazsa, SEAT_RESERVATION oluşmayacağı için MAKE ilişkisini de Weak Relation olarak tanımlıyoruz. CUSTOMER, SEAT_RESERVATION için Strong Entity, SEAT_RESERVATION ise CUSTOMER için Weak Entity durumunda oluyor. Böylece istenen ilişki başarılı bir şekilde oluşturulmuş oluyoruz.

Şekil 4. *SEAT_RESERVATION ve CUSTOMER arasındaki MAKES ilişkisi*

LEG_INSTANCE, SEAT varlıkları arasındaki RESERVATION ilişkisini düzenledik. Bu düzenlemeyi LEG_INSTANCE ile SEAT_RESERVATION arasında gerçekleşen HAS ilişkisi oluşturarak gerçekleştirdik. Daha sonra verilen özellikler dahilinde CUSTOMER varlığını oluşturduk. Oluşturduğumuz bu varlık ile SEAT_RESERVATION arasında MAKES adında bir ilişki kurduk. Gerçekleştirdiğimiz tüm bu adımlar sonucunda CUSTOMER varlığını başarılı bir şekilde tasarımımıza eklemiş oluyoruz. Tüm düzenlemeler sonucu oluşan ilişkileri Şekil 5 üzerinde görebiliriz.



Şekil 5. *CUSTOMER varlığı eklendi.*

Yaptığımız gerçekleştirimi örnekleme adına tablolar oluşturacak olursak;

CUSTOMER, Primary Key = Passport_no

{Passport_no:int, Customer_name:varchar(45), Customer_phone:varchar(10), Email:varchar(20), Country:varchar(3)}

Passport_no	Customer_name	Customer_phone	Email	Coutry
4015005	Mark	+50514578	mark@yahoo.com	US
4013550	Loki	+35841525	loki@yahoo.com	FR
4012351	Frankie	+50321568	frankie@yahoo.com	US
4019874	Sue	+50718462	sue@yahoo.com	US
4016777	Bernard	+42154893	bernard@yahoo.com	UK

SEAT_RESERVATION, Primary Key = Flight_number + Leg_number + Date + Seat_number

{Flight_number:int, Leg_number:int, Date:date, Seat_number:int, Customer_name:varchar(10), Customer_phone:varchar(10)}

Flight_number	Leg_number	Date	Seat_number	Customer_name	Customer_phone
7	2	02/10/2020	20	Mark	+50514578
2	3	05/11/2020	35	Loki	+35841525
7	3	02/10/2020	06	Mark	+50514578
3	3	01/01/2021	35	Sue	+50718462

2.2 Company varlığının eklenmesi

Bizden gerçekleştirmemiz beklenen adım, COMPANY adlı varlığı tasarımımıza eklememizdir. Bu ekleme işlemini ise AIRPLANE varlığı ve AIRLINE varlığı üzerinden gerçekleştirmemiz isteniyor. Başlangıçta bize verilen Relational Schema üzerinde AIRLINE ile ilgili tek bilginin FLIGHT tablosu üzerinde olduğu görülüyor.

Şekil 6. *FLIGHT tablosu*

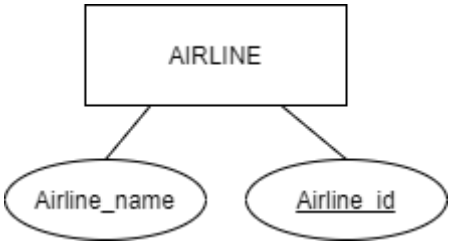
Bu sebeple oluşturacağımız AIRLINE varlığı ile FLIGHT varlığı arasında bir ilişki olacaktır. AIRLINE varlığı bir havayolu firmasını temsil ediyor ve FLIGHT ise bir uçuşu temsil ediyor. Bu ilişkiyi doğal dil olarak ifade edersek oluşacak ifadeler şunlardır; Bir havayolu birden fazla uçuş düzenleyebilir, bir uçuş sadece bir havayolu firması tarafından düzenlenmek zorundadır. Bu ilişkiler dahilinde FLIGHT ve AIRLINE varlıkları arasında ARRANGES ilişkini oluşturduk. FLIGHT varlığı, sadece bir AIRLINE tarafından oluşturulabildiği için AIRLINE varlığının notasyonunu N olarak belirledik. AIRLINE varlığı, birden fazla FLIGHT düzenleyebileceği için FLIGHT varlığının notasyonunu N olarak belirledik. Bir AIRLINE varlığının FLIGHT düzenlemesi zorunlu olmadığı için AIRLINE ve ARRANGES arasındaki ilişkiyi partial olarak tanımladık. FLIGHT varlığının mutlaka bir AIRLINE tarafından düzenlenmesi zorunluluğu bulunduğu için FLIGHT ve ARRANGES ilişkisini total olarak belirleriz. Sonuç olarak oluşan tasarımımız Şekil 7. Üzerinde görülmektedir.

Şekil 7. *AIRLINE ve FLIGHT arasındaki ARRANGES ilişkisi*

AIRLINE varlığına Airline_id ve Airline_name adında iki adet özellik tanımlıyoruz. Tanımlanan bu özelliklerden Airline_id değeri, içerisinde bir AIRLINE varlığını temsil edecek bir

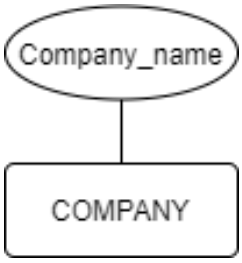
sayıyı barındıracaktır. Bu değer her bir AIRLINE varlığı için eşsiz(unique) bir değer olacağı için bu özelliğimizi Primary Key olarak tanımlıyoruz. Airline_name özelliği içerisinde de her

AIRLINE varlığının isimlerini barındıracaktır. Özelliklerin de belirlenmesi ile AIRLINE varlığı oluşturma işlemini tamamlarız.



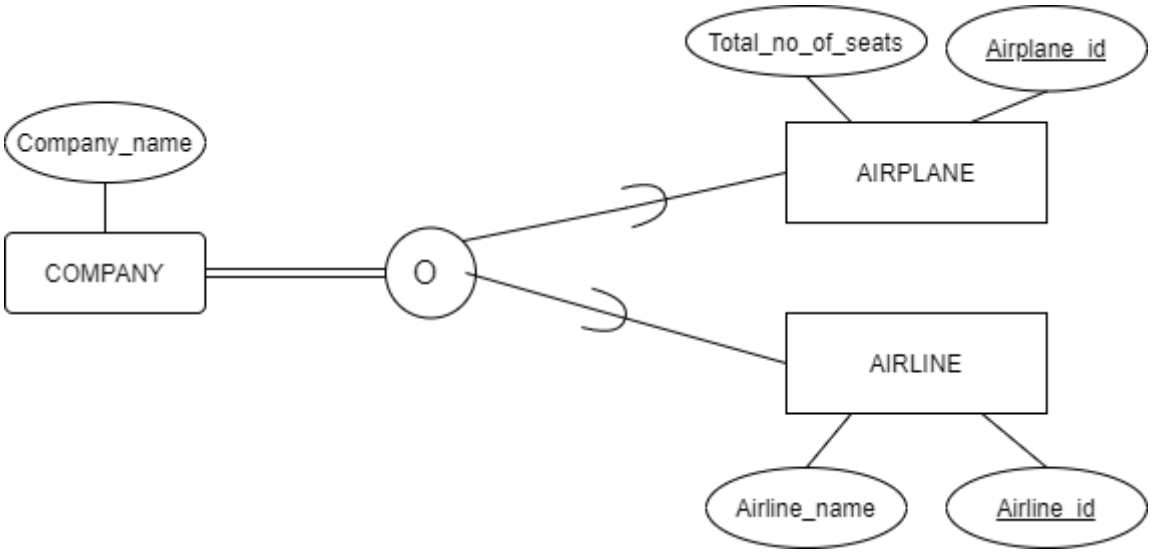
Şekil 8. *AIRLINE varlığı ve özellikleri*

COMPANY, bir şirketi temsil edecek olan varlık olacaktır. COMPANY varlığının işlevini doğal dil üzerinden ifade edecek olursak; bir isme sahip olan şirket hem havayolu firmasına hem de uçağa sahip olabilir, şeklinde ifadeyi gerçekleştirebiliriz. Burada ifade edilen havayolu firması AIRLINE varlığını, uçak ise AIRPLANE varlığını temsil etmektedir. Bu ifadeye göre COMPANY varlığı hem AIRLINE hem de AIRPLANE varlıklarının üstünde bulunan bir varlık olacaktır, çünkü bu varlıklara sahip olan bir varlıktır. Tüm bu değerlendirmeler sonucunda COMPANY, AIRLINE ve AIRPLANE varlıkları arasında 'Specialization' prensibini uygulayarak bizden istenen tasarımı gerçekleştirebileceğiz. Specialization prensibini nasıl gerçekleştiği ile ilgili detayları 2.2.1 numaralı başlık altında inceliyor olacağız. Fakat bu adım öncesinde COMPANY varlığının gerçekleştirimini yapacağız. Bu adımda COMPANY varlığının sahip olduğu özellikleri değerlendirirsek sadece şirketin ismini taşıyacak olan Company_name adında bir özelliğin tanımlanması tasarımımızda yeterli olacaktır. Bu sonuç ile gerçekleştirdiğimiz COMPANY tasarımımız Şekil 9 üzerinde görülmektedir.



Şekil 9. *COMPANY varlığının gerçekleştirimi*

COMPANY varlığını oluşturduktan sonra specialization adımına geri dönüyoruz. Specialization prensibini gerçekleştirirken Superclass,Subclass ve aralarında ilişkiyi daha önceden belirlemek gerekiyor. Burada Superclass olarak tanımlanacak varlık, COMPANY varlığıdır, çünkü diğer iki varlığa da sahip olacağından ikisini de kapsayacaktır. Subclass olarak da diğer iki varlığımızı belirleriz, çünkü AIRLINE ve AIRPLANE varlıkları COMPANY varlığı altında bulunmaktadır. İlişki olarak da 'Overlapping' ilişkisini belirledik, sebebi ise bir COMPANY hem AIRLINE hem de AIRPLANE sahibi olabileceği için ikisini de gerçekleştirebilecek olan overlapping yapısını kullanıyoruz. Bu değerlendirmeler ışığında tasarımımızı gerçekleştirirsek Şekil 10 üzerinde görülecek olan tasarımı elde ederiz.



Şekil 10. COMPANY varlığı üzerinde Specialization gerçekleştirimi

Yaptığımız gerçekleştirimi örnekleme adına tablolar oluşturacak olursak;

COMPANY

{Company_name:varchar(15)}

Company_name
Turkish Airlines
Emirates
Etihad

AIRLINE, Primary Key = Airline_id

Airline_id	Airline_name	Company_name
0	Turkish Airlines	Turkish Airlines
1	Emirates Airways	Emirates
2	Etihad Airways	Etihad

2.2.1 Generalization / Specialization prensibinin uygulanması

Verilerin büyümesi ve karmaşıklığının artması ile birlikte mevcut ER Modelin ile komplike uygulamaları daha iyi ele almak için bazı iyileştirmeler veya geliştirmeler yapılmıştır. Bu geliştirmeler ile birlikte mevcut ER Modeline yeni kavramlar eklenmiştir. Eklenen bu kavramlar ise;

- Generalization
- Specialization
- Aggregration

Bu konu altında Generalization ve Specialization prensiblerinden bahsedeceğiz.

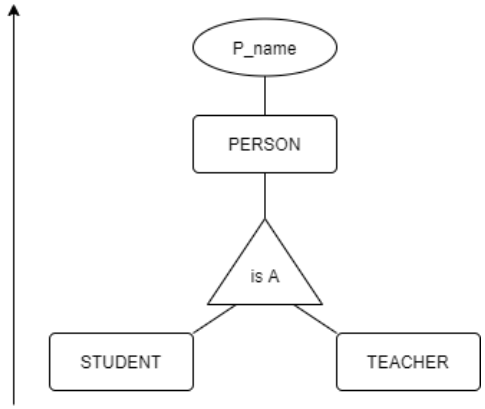
Generalization

Generalization, bir dizi varlıktan ortak özellikleri çıkarma ve ondan genelleştirilmiş bir varlık oluşturma sürecidir. Alt seviyedeki varlıklardan daha yüksek seviyeli bir varlık oluşturmak için ortak özelliklerin birleştirilmesiyle oluşan, aşağıdan yukarıya doğru bir yol izleyen yaklaşımdır. İki veya daha fazla varlığın ortak bazı özelliklere sahip olmaları halinde daha yüksek seviyeli bir varlığa genelleştirilebildiği bir prensiptir.

Daha çok Superclass ve Subclass sistemine benzeyen bu prensibin tek farkı aşağıdan yukarıya doğru bir yol izlemesidir. Dolayısıyla varlıklar daha genel bir varlık oluşturmak üzere bir araya getirilir, yani subclasslar bir superclass oluşturmak için birleştirilir.

Örneğin, STUDENT ve TEACHER varlıkları genelleştirilebilir ve her ikisini de kapsayan, PERSON adında bir varlık yaratılabilir. Alt tablo içerisinde bulunan ortak özellikler ana tabloya eklenir ve alt tablo içerisinden ortak özellikler kaldırılır. Böylece alt tablolarda sadece o tablolara özgü olan özel bilgiler kalır. Örnek verecek olursak Name özelliği her iki varlık için de ortaktır. Bu sebeple PERSON tablosuna P_name ismiyle koyulabilir ancak STUDENT tablosunun altında bulunan GRADE adlı not

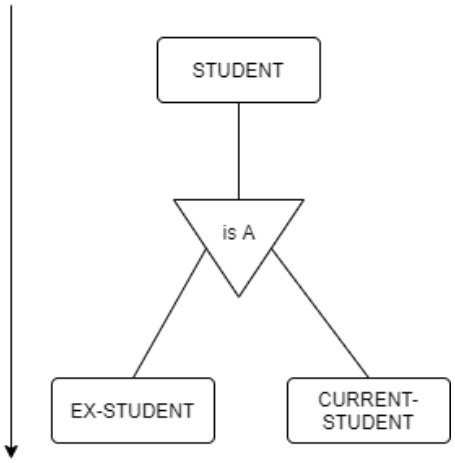
bilgisi, TEACHER isimli varlık altında bulunmadığı için PERSON tablosuna eklenemez, sadece STUDENT'a özgü olduğu için o tabloda kalır.



Şekil 11. *Genaralization Gerçekleştirimi*

Specialization

Specialization prensibinde bir varlık, özelliklerine göre alt varlıklara ayrılır. Daha yüksek seviyeli varlığın iki veya daha fazla alt seviyeli varlığa özelleştirildiği yukarıdan aşağıya doğru bir yol izleyen yaklaşımdır. Specialization, Generalization'ın tam tersidir denilebilir. Örneğin EX-STUDENT ve CURRENT-STUDENT varlıklarını tek bir STUDENT tablosu altında tutmak yerine EX-STUDENT tablosu altında eski öğrencileri, CURRENT-STUDENTE altında ise şuan ki mevcut öğrencileri tutarak kullanımı daha optimize hale getirebiliriz.



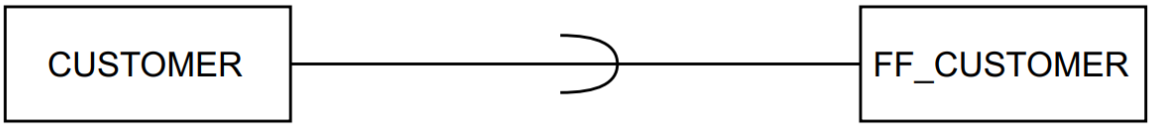
Şekil 12. *Specialization Gerçekleştirimi*

2.3 FFC Varlığının Tanımı

Frequent Flyer Customer, daha önce 2.1 bölümünde tanımladığımız 'Customer' varlığının düzenli olarak uçuş gerçekleştiren hali olarak görülebilir. Yani dolayısıyla FFC, CUSTOMER varlığından türemiş bir varlık olarak düşünülebilir çünkü teknik olarak CUSTOMER varlığının bütün özelliklerini taşıması gerekmektedir.

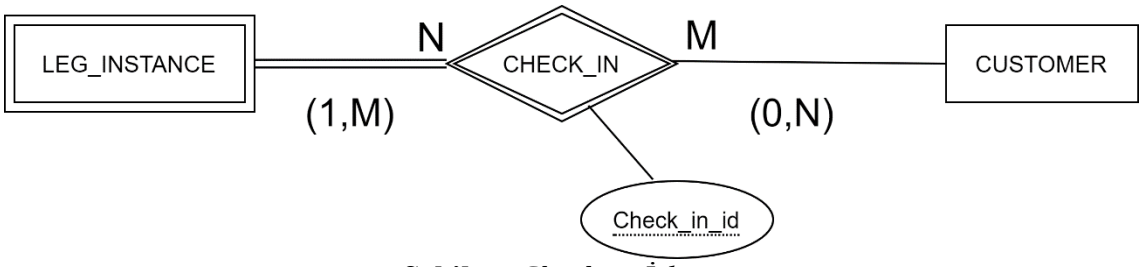
2.3.1 FF_CUSTOMER – CUSTOMER İlişkisi

İlk paragrafta da belirttiğimiz gibi FF_CUSTOMER, CUSTOMER varlığından türemiş bir varlık olarak düşünmek doğru olacaktır. Böylece CUSTOMER'ı FF_CUSTOMER için superclass, FF_CUSTOMER'ı de CUSTOMER için bir subclass olarak düşünebiliriz.



Şekil 13. *Superclass-Subclass*

2.3.2 Check-In İşleminin Sağlanması

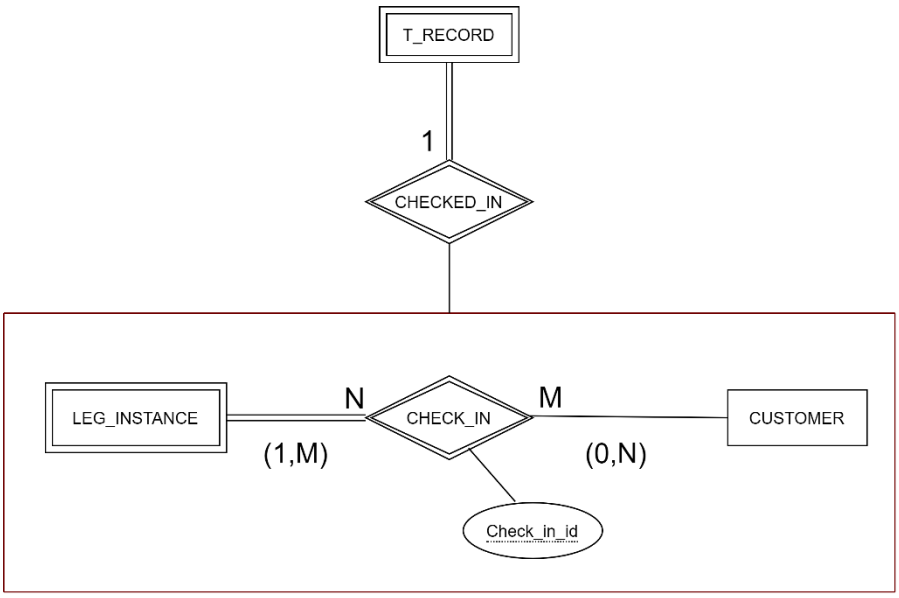


Şekil 14 *Check-in İşlemi*

2.3'te bizden gerçekleştirilmesi istenen nokta, her CUSTOMER'ın bağlı olduğu LEG_INSTANCE'a Check-in işleminin sağlanıp, buna göre Frequent Flyer (devamlı müşteri diyebiliriz) Customer'ların tanımlanmasıdır. Bunun için Şekilde iki varlığı birbirlerine CHECK_IN ilişkisi ile bağladığımızda, karşımıza bazı soru işaretleri gelmektedir. Veritabanımızın içerdiği bir başka varlık olan T_RECORD buraya nasıl bağlanabilir? İşte tam bu soruyla ilgili önce 'aggregation' yaklaşımını kullanarak bir yaklaşım gerçekleştirelim. 2.3.4 kısmında ise alternatif bir yaklaşımı inceliyor olacağız.

Öncelikle, Şekilde ilişkiye baktığımızda, 'M tane CUSTOMER, N tane LEG_INSTANCE için CHECK_IN işlemini gerçekleştirebilir' diyebiliriz. Şeklin bu haliyle unutulmaması gereken nokta, M:N (many-to many) bir ilişki tanımladığımız için CHECK_IN ilişkisinin, bağlı olduğu her iki varlığın da Primary Keylerini bünyesine alıyor olmasıdır. Ayrıca kendine ait 'Check_in_id' adında bir Partial Key'e sahiptir. Daha sonraki bölümlerde bu ilişki-varlık için tablo tanımlayacağız.

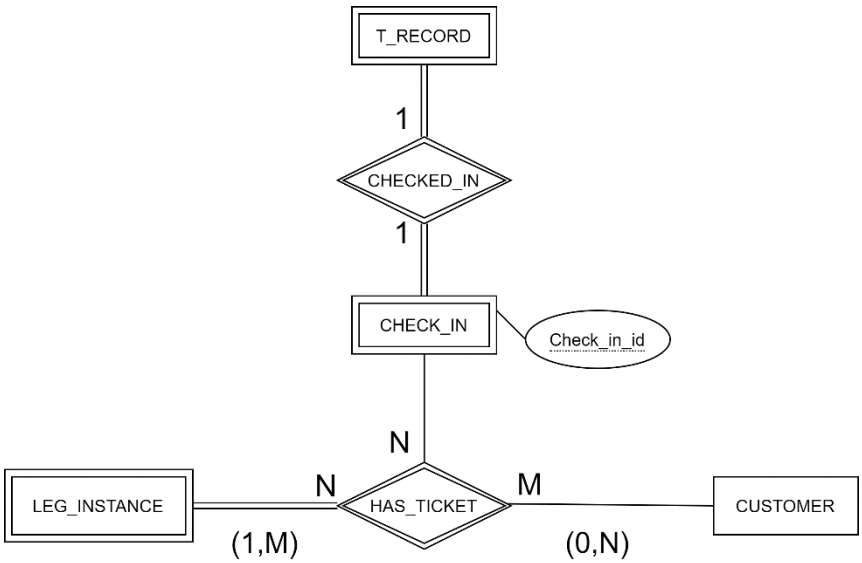
2.3.3 Aggregation Prensibinin Gerçekleştirilmesi



Şekil 15 *Aggregation Yöntemi (High Level entity)*

Şekli incelediğimizde, Aggregation yönteminin temel amacı olan 'belirli bir varlık-ilişki kümesini tek bir entity'e indirgeme' işlemini görüyoruz. Burada, LEG_INSTANCE ve CUSTOMER varlığı ile, CHECK_IN ilişkisini tek bir Higher Level Entity'de göstermiş oluyoruz. Bu durumda elimizde Bu Higher Level Entity ile, T_RECORD ile CHECK_IN üzerinden ilişki kuran bir binary relationship kalmış oluyor, yani genel yapıyı basite indirmiş olduk. Fakat bu durum genel EER Diagram kurallarına uymuyor (Referans: *Elmasri, Database Systems, sayfa: 133*)

Bu durumda yapımızda değişikliğe gitmeliyiz. Bu ilişkileri birbirine bağlamak yerine, elimizde mevcut bulunan CHECK_IN ilişkisini bir weak entity olarak tanımlamıyız. Sistemde CHECK_IN ilişkisi yerine geçecek bir ilişki daha tanımlamalıyız. Yeni oluşturacağımız CHECK_IN weak entity'sini T_RECORD'a bağlayacak ilişki zaten hali hazırda olan CHECKED_IN ilişkisidir. Böylece hem aggregation mantığını sağlamış olacağız hem de EER kurallarına uygun bir diagram elde edeceğiz. (Referans: *Elmasri, Database Systems, sayfa: 133*)



Şekil 16 Aggregation yönteminin doğru uygulanışı

Son diagram parçamızın, bize ulaşmak istediğimiz şeyi sağladığını düşünmekteyiz. Son durumda şekilde de görüleceği üzere 'Check_in_id' Partial Key'ini ait olduğu CHECK_IN weak entity'sine bağladık. Bu şekil üzerinden okuma yaptığımızda;

- 'Her bilet alımı yapılmış N LEG_INSTANCE için, bileti almış olan CUSTOMER, N tane CHECK_IN işlemi yapılabilir.'

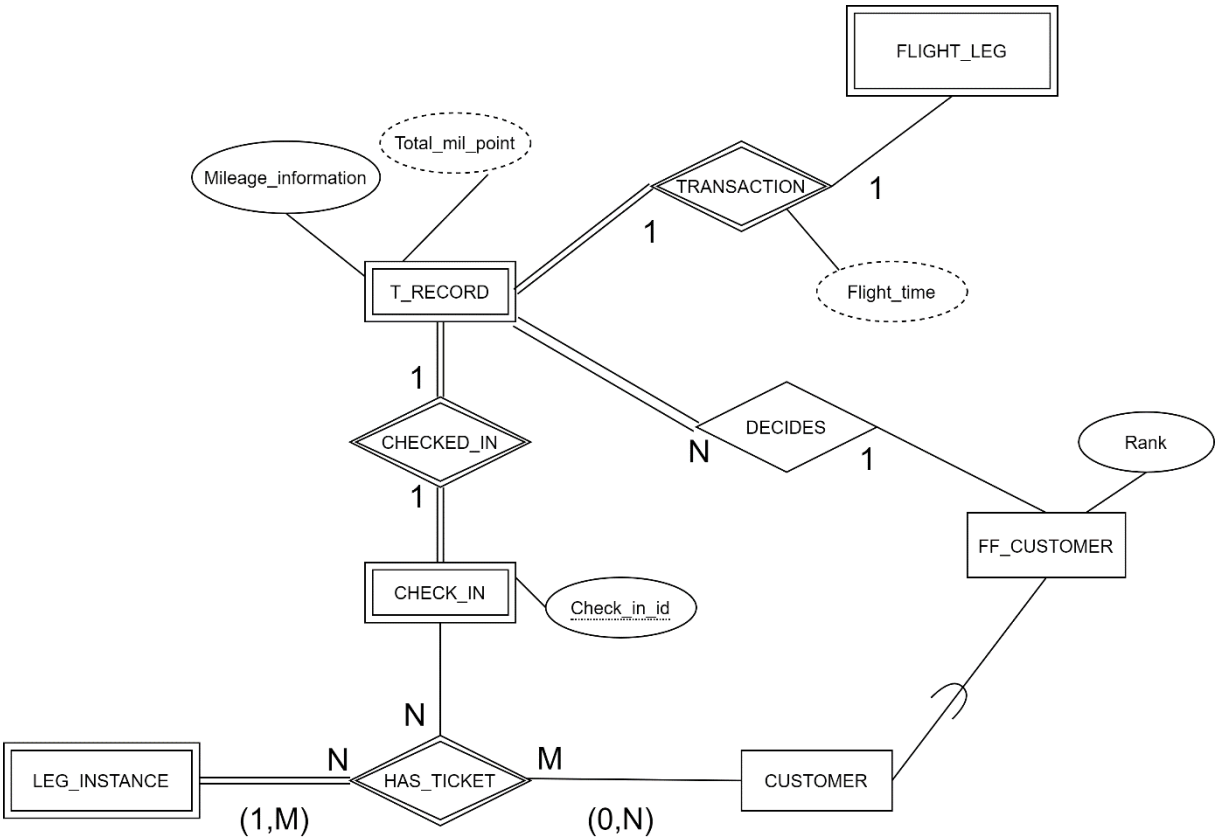
'yapılabilir' ibaresi, her bilet alınan uçuş için CHECK_IN gerçekleştirilemeyebileceğine referans eder.

- 'M CUSTOMER arasından 1'i, HAS_TICKET işlemini gerçekleştirmiş olduğu N LEG_INSTANCE için N tane CHECK_IN gerçekleştirebilir.'
- 'Eğer N içinden 1 CHECK_IN gerçekleştirildiyse, Bu işlemi gerçekleştirmiş olan M içerisinden 1 CUSTOMER, ve HAS_TICKET işlemini gerçekleştirmiş olduğu N içerisinden 1 adet LEG_INSTANCE'a bağlı olmak zorundadır.'

'zorundadır' ibaresi aslında CHECK_IN işlemi yapılabilmesi için hem LEG_INSTANCE'dan hem CUSTOMER'dan mantıklı birer öğenin HAS_TICKET işlemini gerçekleştirmesi gerektiğini ifade eder. Böylece, CHECK_IN weak entity'sinin, hem LEG_INSTANCE'a hem de CUSTOMER'a göre weak olduğunu ifade edebiliriz.

Şimdi, bütün bu yaptıklarımızı birleştirip FF_CUSTOMER'a nasıl bağlanacağını açıklamaya çalışıyoruz.

2.3.4 Aggregation ile Çözüm Yöntemi



Şekil. *Aggregation ile çözüm*

Önceki bölümlerin üstüne, T_RECORD'dan devam edecek olursak, T_RECORD tablomuz, iki adet kendine has özellik barındırır, bunlar 'Milage_information' ve 'Total_mil_point'dir. Milage information, gerekli olan mil bilgilerinin, FLIGHT_LEG tablosu üzerindeki iki tane uçuş süre bilgisi barındıran bilgilerin çıkarma işlemi sonucunu (Scheduled_Arrival_time – Scheduled_Departure_time = Flight_time) TRANSACTION yoluyla alıp sonradan belirnecek bir tip yolcu uçağının ortalama saat başı sürat değeriyle çarpılıp elde edilecektir. Şekil üstünde bulunan Flight_time verisini *eğer mümkünse* tablolar içinde göstermeyip algoritmik yollar ile halletmeyi düşünüyoruz (yani dönüştürme işlemlerini). Bu alınan milage information verisi, her aynı customer_passport verisi için toplanıp 'Derived Attribute' olan Total_mil_point isimli veriye yazılır. Eğer toplam belirlenen mil sınırını aşan bir customer olursa, DECIDES isimli ilişki üzerinden FF_CUSTOMER'a yazılır ve Rank attribute'u belirlenmiş olur.

Bu anlattıklarımızı örneklemek adına tablolar oluşturacak olursak;

CHECK_IN, Primary Key = Check_in_id + Customer_passport

{Check_in_id:int, Passport_no:int, Flight_number:int, Leg_number:int, Date:Date}

Check_in_id	Passport_no	Flight_number	Leg_number	Date
101	4015005	7	2	02/10/2020
102	4013550	2	3	05/11/2020
103	4015005	7	3	02/10/2020

T_RECORD

{Check_in_id:int, Passport_no:int, Milage_information:float, Total_mil_point:float}

Check_in_id	Customer_passport	Milage_information	Total_mil_point
101	4015005	1532,5	1532,5
102	4013550	780	780
103	4015005	380	1912,5

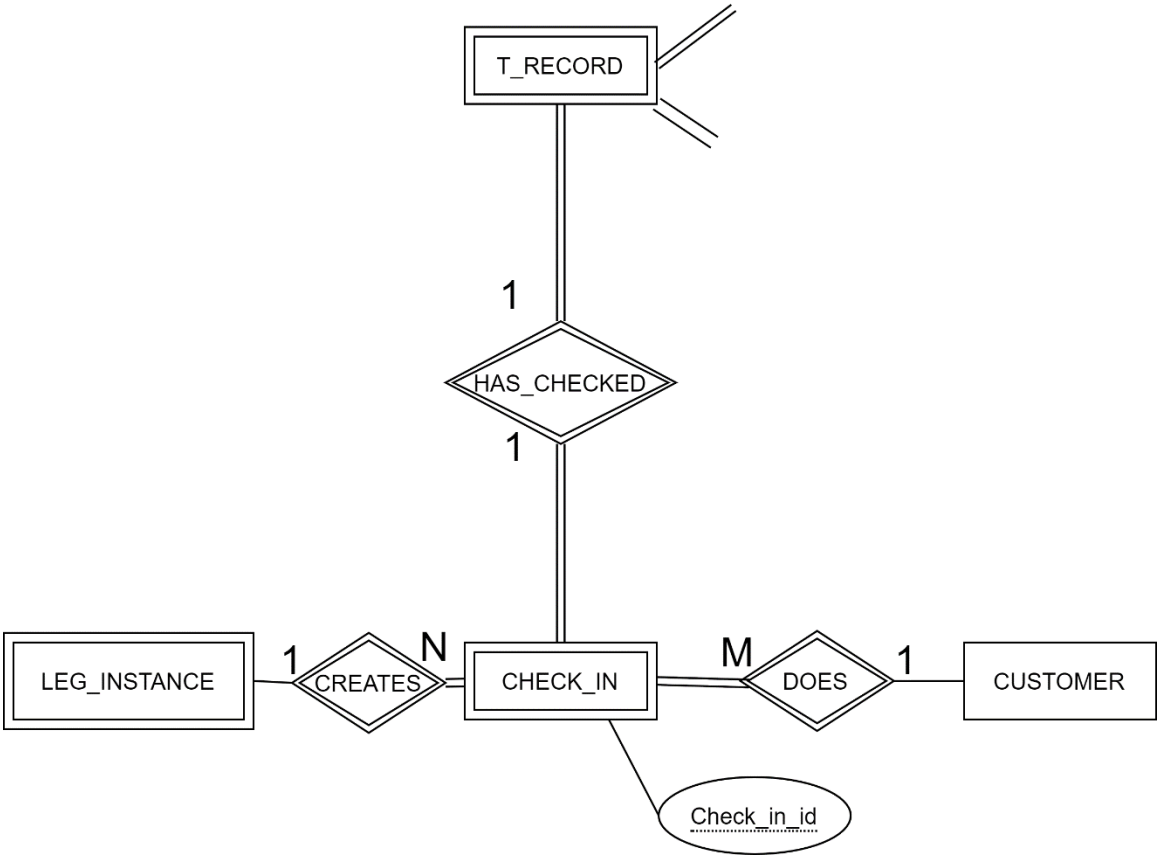
Örnek olarak FF_CUSTOMER'lar için Total_mil_point barajını 1700 mil olarak düşünürsek, FF_CUSTOMER Primary Key: Customer_passport

{Customer_passport:int, Customer_name:varchar(10), Phone:varchar(10), Email:varchar(15), Country:varchar(3), Rank:int}

Customer_passport	Customer_name	Phone	Email	Country	Rank
4015005	Mark	+50514578	mark@yahoo.com	US	1

4015005 pasaport numaralı customer, 2 kez Check_in yapmıştır. CHECK_IN tablomuzdan bu bilgiye ulaşıyoruz ve T_RECORD tablomuza bu durumu aktarıyoruz. T_RECORD tablosu, CHECK_IN tablosuna göre weak entity olduğundan, Primary Key'lerin T_RECORD tablosuna aktarıldığını görüyoruz. Buradan sonrası artık hesaplama işlerine kalıyor. FLIGHT_LEG tablosundan ilgili hesaplamaları (o tablonun eklenmesine gerek görmedik, amacımız FF_CUSTOMER'a geçişi göstermek) yaparak milleri hesaplıyoruz. Görüldüğü gibi her check_in_id için mil bilgilerini elde edebiliyoruz. Buna göre, eğer aynı passport numaralı bir Customer birden çok kez check_in yaparsa total_mil_point artarak gidiyor. Örnek olarak 1700 sınırını aştığından bir Customer'ı FF_CUSTOMER entity'sine aktardık. Rank'i de 1 oldu, belirli farklı mil puanları için Rank artabilir, böylece ücret konusunda indirim miktarı belirlenip bu indirim Rank attribute'una göre arttırılabilir.

2.3.5 Alternatif Çözüm Yöntemi

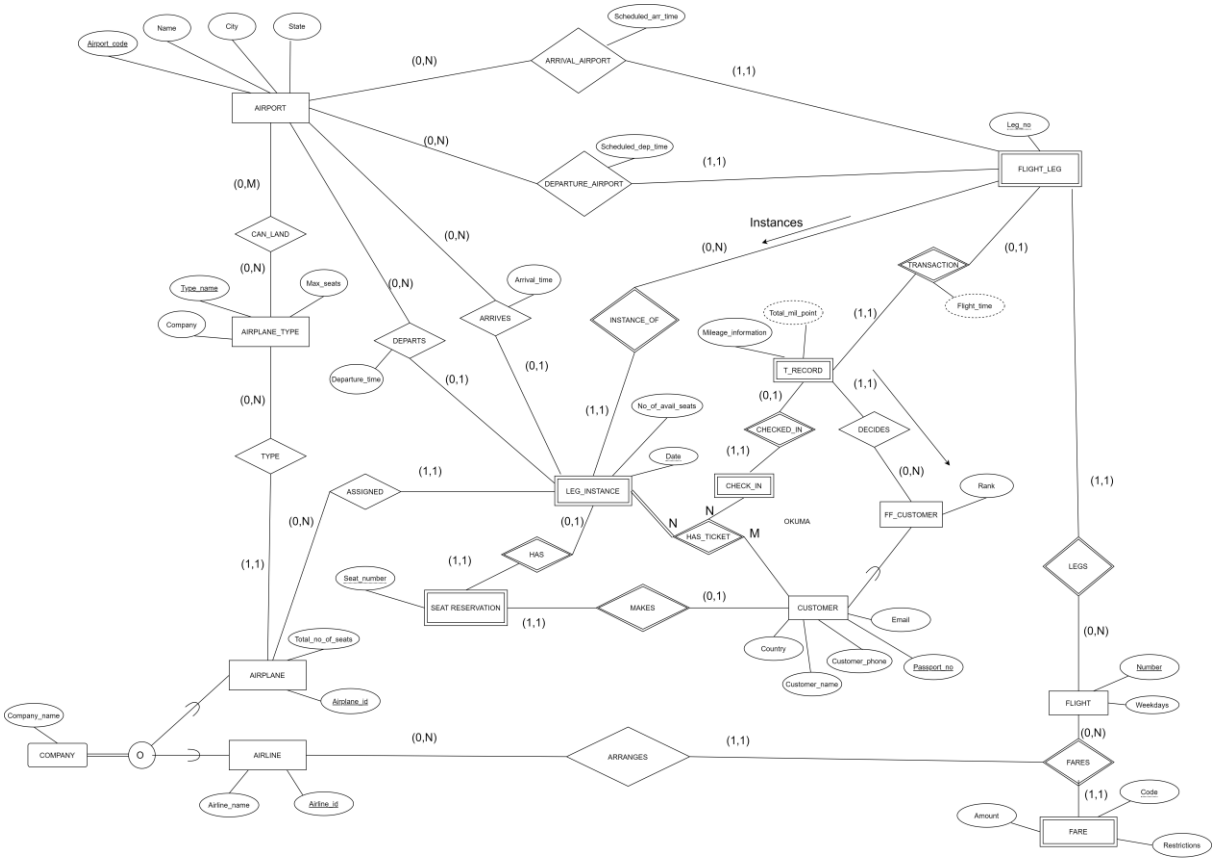


Şekil. *Ternary Bağlantıları ile çözüm yöntemi*

Bu çözüm yönteminde ise, Ternary bir ilişki çeşidini, weak relationship'ler ile birbirine bağlayıp çözüme ulaşabileceğimizi düşündük. (Referans: *Elmasri, Database Systems, Sayfa 89*) Buna göre, CHECK_IN weak relation'umuz bir weak entity oldu, bu entity'leri birbirine bağlayan 3 tane farklı weak relation tanımlamış olduk. Belki bu model üstünde biraz optimizasyon gerçekleştirilse, çözüm yolunda kullanılabilecek daha az karmaşık bir yöntem olması sebebiyle raporumuza eklemeye karar verdik. Buradaki tek problem, bu yöntemin normalinin aksine, CHECK_IN, yani sonradan weak entity haline getirilen varlık, T_RECORD'a göre strong bir hale geliyor çünkü mantıken CHECK_IN varlığı oluşmaz ise bir T_RECORD varlığından söz edemiyoruz. Ayrıca T_RECORD için CHECK_IN'de belirlediğimiz primary keyleri (önceki bölümün aynısı) aynı şekilde kullanabilme imkamınız da doğuyor. Dolayısıyla, programın implementasyon aşamasında bu yöntemin optimize edilerek denenmesi, hem performans kazancı açısından bize yeni fırsatlar sunabilir, hem de önceki tasarımdan daha kompakt bir hal alabilir. İşbu tasarım ek olarak verilmiş olup, şu an uyguladığımız EER tasarımında kullandığımız yöntem, bir önceki bölümdeki aggregation yöntemidir. Koşulların durumuna göre, kullanımı amaçlanabileceğinden rapora eklenmiştir.

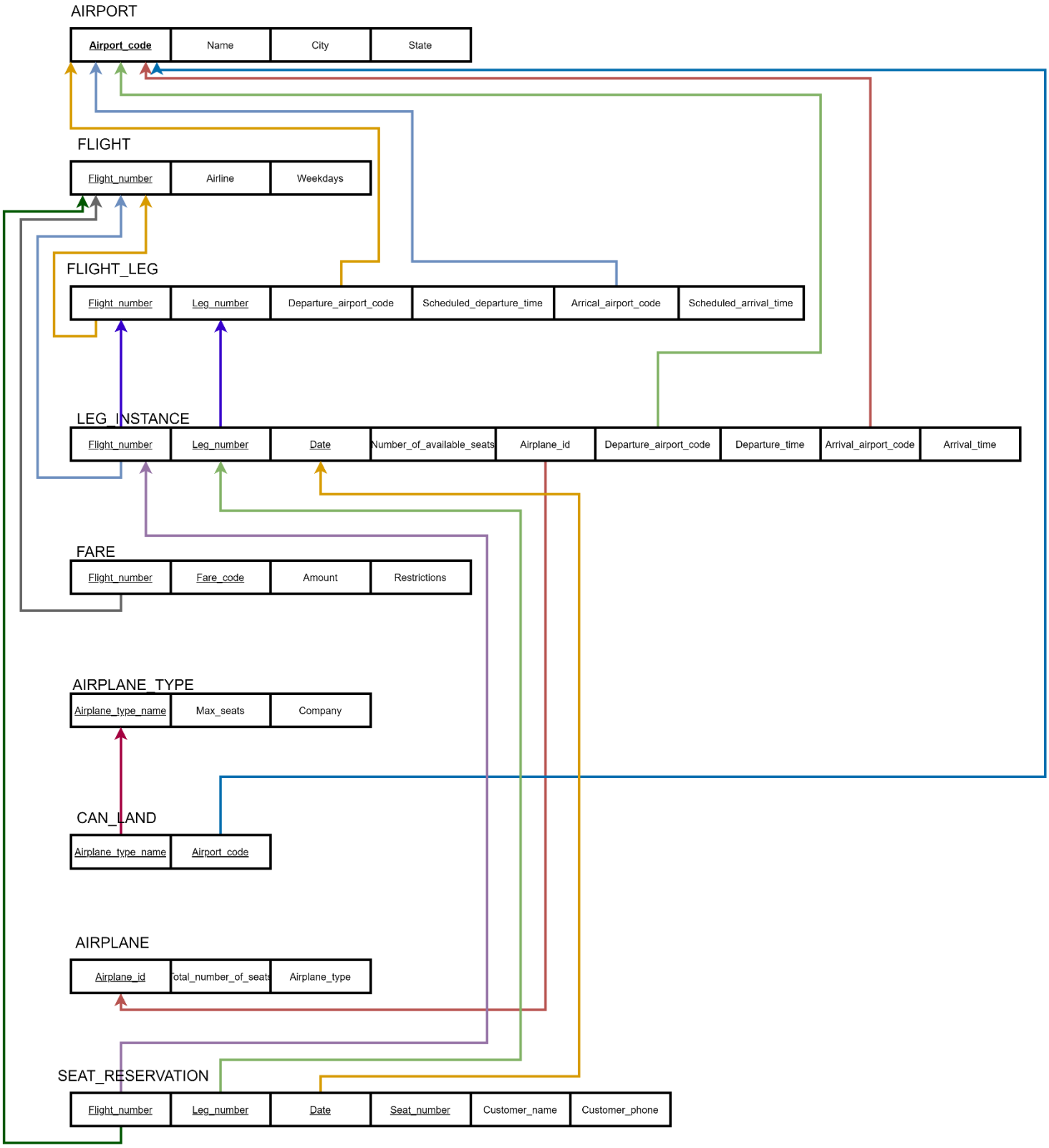
2.4 Kardinalite Değerleri ve (MIN,MAX) Notasyonu Gösterimi

İki gösterimi de içeren diagramımız aşağıya eklenmiştir. Her genel diagramın .drawio formatında linki raporumuzun en sonunda ‘github’ linki olarak paylaşılmıştır. (.drawio dosyaları app.diagrams.net sitesi üzerinden doğrudan açılabilir)



Min-Max Notasyonu ile diagram gerçekleştirimi

2.5 Referential Integrity İlişkilerinin Gösterimi



Şekil. ***Relational Schema***

FLIGHT_LEG.Departure_airport_code → AIRPORT.Airport_code

FLIGHT_LEG.Arrival_airport_code → AIRPORT.Airport_code

FLIGHT_LEG.Flight_number → FLIGHT.Flight_number

LEG_INSTANCE.Leg_number → FLIGHT_LEG.Leg_number

LEG_INSTANCE.Flight_number → FLIGHT.Flight_number

LEG_INSTANCE.Airplane_id → AIRPLANE.Airplane_id

LEG_INSTANCE.Departure_airport_code → AIRPORT.Airport_code

LEG_INSTANCE.Arrival_airport_code → AIRPORT.Airport_code

FARE.Flight_number → FLIGHT.Flight_number

CAN_LAND.Airplane_type_name → AIRPLANE_TYPE.Airplane_type_name

CAN_LAND.Airport_code → AIRPORT.Airport_code

SEAT_RESERVATION.Flight_number → FLIGHT.Flight_number

SEAT_RESERVATION.Leg_number → FLIGHT_LEG.Leg_number

SEAT_RESERVATION.Date → LEG_INSTANCE.Date

İkinci soruyla birlikte eklenen ilişkiler:

CHECK_IN.Customer_passport → CUSTOMER.Customer_passport

CHECK_IN.Flight_number → FLIGHT.Flight_number

CHECK_IN.Leg_number → FLIGHT_LEG.Leg_number

CHECK_IN.Date → LEG_INSTANCE.Date

T_RECORD.Customer_passport → CUSTOMER.Customer_passport

T_RECORD.Check_in_id → CHECK_IN.Check_in_id

2.6 Eventual Design ile Conceptual Design arasındaki Farklılıklar

-List details that will affect the eventual design but that have no bearing on the conceptual design

İlgili bölüme cevap olarak yazılmıştır.

Örneğin bizim EER dizaynımızda, AIRPLANE varlığının tipi ile ilgili bir belirleme yapılmamıştır. Eventual Design'da farklı AIRPLANE tipleri, yani farklı işlerde görevli AIRPLANE varlıkları(kargo uçağı gibi) oluşturulabilir fakat bu durum bizim için konsept dışı kalıyor, yani bizim veritabanımızın Kavramsal dizaynını etkilemiyor.

Başka bir örnek verecek olursak, SEAT RESERVATION varlığı için, rezervasyonların birbirinden hiçbir farkı yok. Yani, örneğin herhangi bir gerçek dünya uçak koltuk rezervasyon sistemine girdiğimizde, koltuk tiplerinin çok çeşitli seçeneklere bölünebildiğini görüyoruz (ekonomi, VIP sınıfları gibi) bizim konseptimizde bu gibi noktalara önem verilmemekte.

Bazı varlıklar, örneğin 2. Bölümde eklediğimiz FF_CUSTOMER varlığı gibi, kavramsal dizayn anlamında çeşitlilik olarak görülmektedir. Bu durumda, farklı gerçek dünya tasarımları ile kavramlar genişletilebilir, bunlar için ise ek varlıklar ve ilişkiler oluşturmak gerekir.

2.7 Semantic Constraints

AIRPORT

Atribute	Data_Type
Airport_code	int
Name	varchar(45)
City	varchar(45)
State	bit

FLIGHT

Atribute	Data_Type
Flight_number	int
Airline	varchar(45)
Weekdays	varchar(45)

FLIGHT_LEG

Atribute	Data_Type
Flight_number	int
Leg_number	int
Departure_airport_code	int
Scheduled_departure_time	time
Arrival_airport_code	int
Schehuled_arrive_code	int

LEG_INSTANCE

Atribute	Data_Type
Flight_number	int
Leg_number	int
Date	date
Number_of_available_seats	int
Airplane_id	int
Departure_airport_code	int
Departure_time	time
Arrival_airport_code	int
Arrival_time	time

FARE

Atribute	Data_Type
Flight_number	int
Fare_code	int
Amount	int
Restrictions	text

AIRPLANE_TYPE

Atribute	Data_Type
Airplane_type_name	varchar()
Max_seats	int
Company	varchar(45)

CAN_LAND

Atribute	Data_Type
Airplane_type_name	varchar()
Airport_code	int

AIRPLANE

Atribute	Data_Type
Airplane_id	int
Total_number_ofseats	int
Airplane_type	varchar(45)

SEAT_RESERVATION

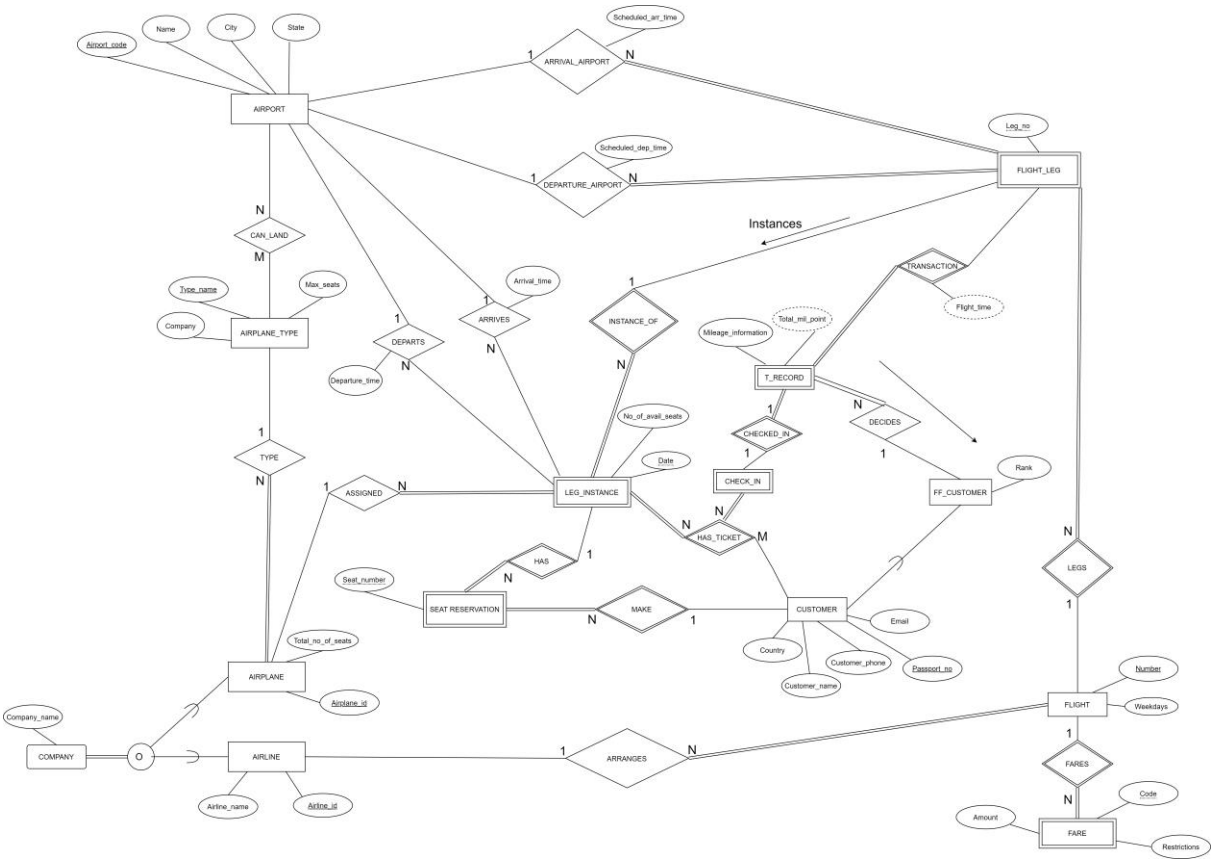
Atribute	Data_Type
Flight_number	int
Leg_number	int
Date	date
Seat_number	int

CUSTOMER

Atribute	Data_Type
Pasport_no	int
Customer_name	varchar(45)
Country	varchar(45)
Email	varchar(45)
Customer_phone	int

2. Bölümdeki ekstra tablolar için, ilgili bölümlerde semantic constraints tanımları yapılmıştır.

3.Diagramın Son Hali



Diagramın Gerekli Özellikler Eklendikten Sonraki Hali

Link: https://github.com/MehmetAnil/Group1_Diagrams

Linkte raporda olan bütün genel diagramlar vardır, ekstra olarak Alternatif çözüm bölümünde kullanılan sistemi de genel diagrama monte ettik.