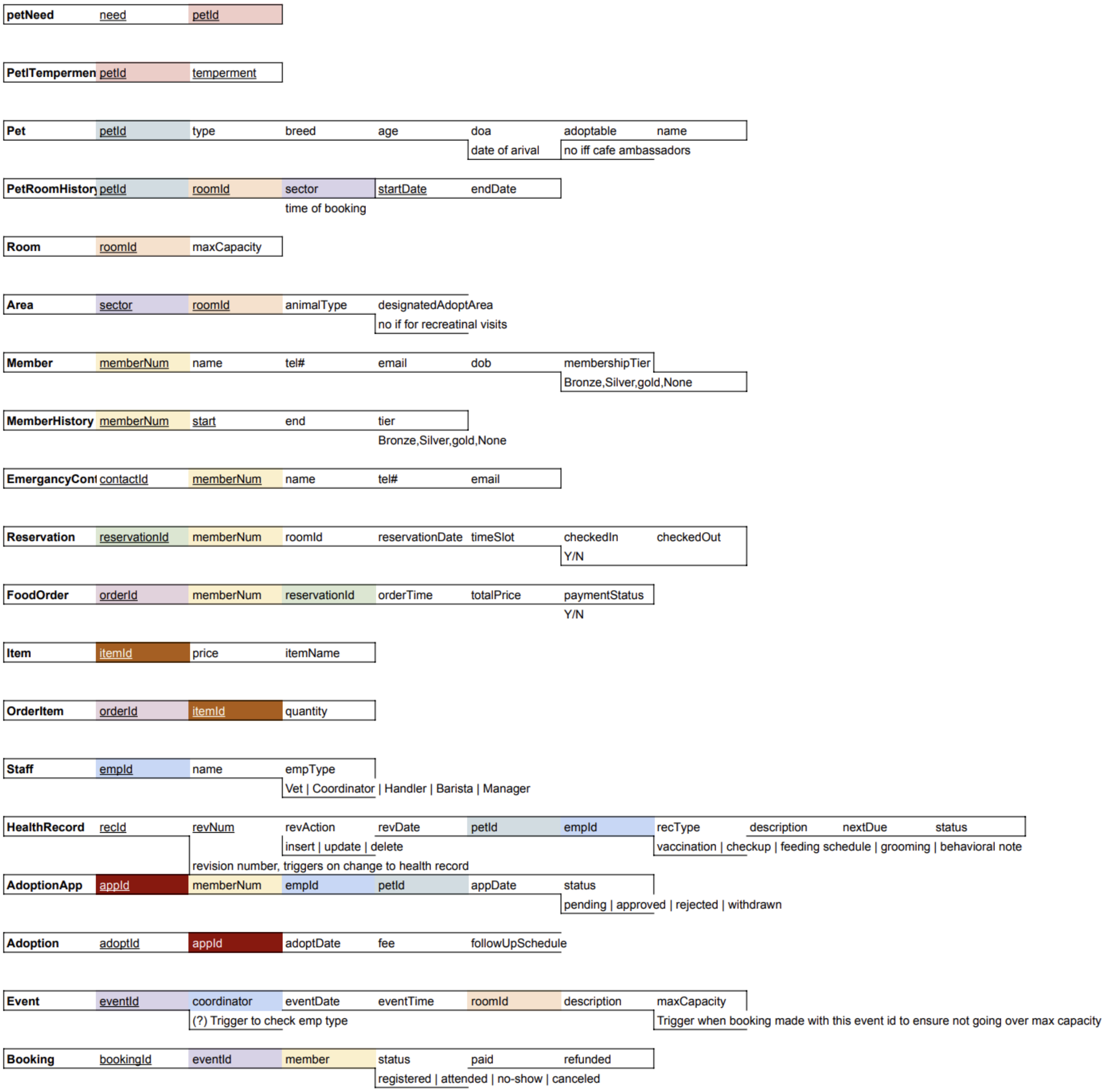
Conceptual Design



petNeed	need	petId								
PetITempermen	petId	temperment								
Pet	petId	type	breed	age	doa	adoptable	name			
					date of arival	no iff cafe ambassadors				
PetRoomHistor	petId	roomId	sector	startDate	endDate					
			time of booking							
Room	roomId	maxCapacity								
Area	sector	roomId	animalType	designatedAdoptArea						
				no if for recreatinal visits						
Member	memberNum	name	tel#	email	dob	membershipTier				
						Bronze,Silver,gold,None				
MemberHistory	memberNum	start	end	tier						
				Bronze,Silver,gold,None						
EmergancyCont	contactId	memberNum	name	tel#	email					
Reservation	reservationId	memberNum	roomId	reservationDate	timeSlot	checkedIn	checkedOut			
						Y/N				
FoodOrder	orderId	memberNum	reservationId	orderTime	totalPrice	paymentStatus				
						Y/N				
Item	itemId	price	itemName							
OrderItem	orderId	itemId	quantity							
Staff	empId	name	empType							
			Vet | Coordinator | Handler | Barista | Manager							
HealthRecord	recId	revNum	revAction	revDate	petId	empId	recType	description	nextDue	status
			insert | update | delete				vaccination | checkup | feeding schedule | grooming | behavioral note			
		revision number, triggers on change to health record								
AdoptionApp	appId	memberNum	empId	petId	appDate	status				
						pending | approved | rejected | withdrawn				
Adoption	adoptId	appId	adoptDate	fee	followUpSchedule					
Event	eventId	coordinator	eventDate	eventTime	roomId	description	maxCapacity			
		(?) Trigger to check emp type					Trigger when booking made with this event id to ensure not going over max capacity			
Booking	bookingId	eventId	member	status	paid	refunded				
				registered | attended | no-show | canceled						

Logical Design

```
CREATE TABLE Room(
   roomId INTEGER,
   maxCapacity INTEGER,
   PRIMARY KEY (roomId)
);

CREATE TABLE Pet (
   petId INTEGER NOT NULL,
   animalType VARCHAR2(15) NOT NULL,
   breed VARCHAR2(255),
   age INTEGER NOT NULL CHECK (age > -1),
   doa DATE NOT NULL,
   adoptable BOOLEAN NOT NULL,
   name VARCHAR2(255),
   PRIMARY KEY (petId)
);

CREATE TABLE PetNeed (
   need VARCHAR2(255),
   petId INTEGER,
   PRIMARY KEY (need, petId),
   FOREIGN KEY (petId) REFERENCES Pet(petId) ON DELETE CASCADE
);

CREATE TABLE PetTemperment (
   temperment VARCHAR2(255),
   petId INTEGER,
   PRIMARY KEY (temperment, petId),
   FOREIGN KEY (petId) REFERENCES Pet(petId) ON DELETE CASCADE
);

CREATE TABLE Area (
   sector INTEGER,
   roomId INTEGER,
   animalType VARCHAR2(10),
   designatedAdoptArea BOOLEAN,
   PRIMARY KEY (sector, roomId),
   FOREIGN KEY (roomId) REFERENCES Room(roomId) ON DELETE CASCADE
);

CREATE TABLE PetRoomHistory (
   petId INTEGER,
   roomId INTEGER,
   sector INTEGER,
   startDate DATE,
   endDate DATE,
   PRIMARY KEY (petId, startDate),
   FOREIGN KEY (petId) REFERENCES Pet(petId) ON DELETE CASCADE,
   FOREIGN KEY (roomId) REFERENCES Room(roomId) ON DELETE CASCADE,
   FOREIGN KEY (sector, roomId) REFERENCES Area(sector, roomId) ON DELETE CASCADE
);

CREATE TABLE Member (
   memberNum INTEGER NOT NULL,
   name VARCHAR2(255),
   tele_num VARCHAR2(20),
   email VARCHAR2(255),
   dob DATE,
   membershipTier VARCHAR2(20) NOT NULL CHECK (membershipTier IN ('BRONZE', 'SILVER', 'GOLD', 'NOT CURRENTLY MEMBER')),
   PRIMARY KEY (memberNum)
);

CREATE TABLE MemberHistory (
   memberNum INTEGER,
   startDate DATE,
   endDate DATE,
   membershipTier VARCHAR2(20) NOT NULL CHECK (membershipTier IN ('BRONZE', 'SILVER', 'GOLD', 'NOT CURRENTLY MEMBER')),
   FOREIGN KEY (memberNum) REFERENCES Member(memberNum) ON DELETE CASCADE
);

CREATE TABLE EmergencyContact (
   contactId INTEGER,
   memberNum INTEGER,
   name VARCHAR2(255),
   tele_num VARCHAR2(20),
   email VARCHAR2(255),
   PRIMARY KEY (contactId, memberNum),
   FOREIGN KEY (memberNum) REFERENCES Member(memberNum) ON DELETE CASCADE
);

CREATE TABLE Reservation (
   reservationId INTEGER NOT NULL,
   memberNum INTEGER NOT NULL,
   roomId INTEGER NOT NULL,
   reservationDate DATE,
   timeSlot INTERVAL DAY TO SECOND,
   checkedIn VARCHAR2(3) NOT NULL CHECK (checkedIn IN('YES', 'NO')),
   checkedOut VARCHAR2(3) NOT NULL CHECK (checkedOut IN('YES', 'NO')),
   PRIMARY KEY (reservationId),
   FOREIGN KEY (memberNum) REFERENCES Member(memberNum) ON DELETE CASCADE,
   FOREIGN KEY (roomId) REFERENCES Room(roomId) ON DELETE CASCADE
);

CREATE TABLE FoodOrder (
   orderId INTEGER,
   memberNum INTEGER,
   reservationId INTEGER,
   orderTime DATE,
   totalPrice NUMBER(6, 2),
   paymentStatus BOOLEAN,
   PRIMARY KEY (orderId),
   FOREIGN KEY (memberNum) REFERENCES Member(memberNum),
   FOREIGN KEY (reservationId) REFERENCES Reservation(reservationId) ON DELETE CASCADE
);

CREATE TABLE Item (
   itemId INTEGER,
   price NUMBER(6, 2), -- assuming we won't have prices higher than $9999.99
   itemName VARCHAR(255),
   PRIMARY KEY (itemId)
);

CREATE TABLE OrderItem (
   orderId INTEGER,
   itemId INTEGER,
   quantity INTEGER,
   PRIMARY KEY (orderId, itemId),
   FOREIGN KEY (orderId) REFERENCES FoodOrder(orderId) ON DELETE CASCADE,
   FOREIGN KEY (itemId) REFERENCES Item(itemId) ON DELETE CASCADE
);

CREATE TABLE Staff (
   empId INTEGER,
   name VARCHAR2(255),
   empType VARCHAR2(255) NOT NULL CHECK (empType IN ('VET', 'CRD', 'HDL', 'BAR', 'MGR')),
   PRIMARY KEY (empId)
);

CREATE TABLE HealthRecord (
   recId INTEGER,
   revNum INTEGER, -- Add triggers for auto making revs on table changes
   revAction VARCHAR2(10) NOT NULL CHECK (revAction IN ('insert', 'update', 'delete')),
   revDate DATE,
   petId INTEGER,
   empId INTEGER,
   recType VARCHAR2(10) NOT NULL CHECK (recType IN ('VET', 'CHK', 'SCH', 'GRM', 'BHN')),
   description VARCHAR2(255),
   nextDue DATE,
   status VARCHAR2(10),
   PRIMARY KEY (recId, revNum),
   FOREIGN KEY (petId) REFERENCES Pet(petId) ON DELETE CASCADE,
   FOREIGN KEY (empId) REFERENCES Staff(empId) ON DELETE SET NULL
);

CREATE TABLE AdoptionApp (
   appId INTEGER,
   memberNum INTEGER,
   empId INTEGER,
   petId INTEGER,
   appDate DATE,
   status VARCHAR2(10) NOT NULL CHECK (status IN ('PEN', 'APP', 'REJ', 'WIT')),
   PRIMARY KEY (appId),
   FOREIGN KEY (memberNum) REFERENCES Member(memberNum) ON DELETE CASCADE,
   FOREIGN KEY (empId) REFERENCES Staff(empId) ON DELETE SET NULL,
   FOREIGN KEY (petId) REFERENCES Pet(petId) ON DELETE CASCADE
);

CREATE TABLE Adoption (
   adoptId INTEGER,
   appId INTEGER, -- removed pet, member, and emp as the application has fields already
   adoptDate DATE,
   fee NUMBER(6, 2),
   followUpSchedule VARCHAR2(255),
   PRIMARY KEY (adoptId),
   FOREIGN KEY (appId) REFERENCES AdoptionApp(appId) ON DELETE CASCADE
);

CREATE TABLE Event (
   eventId INTEGER,
   coordinator INTEGER,
   eventDate DATE,
   eventTime INTERVAL DAY TO MINUTE,
   roomId INTEGER,
   description VARCHAR2(255),
   maxCapacity INTEGER,
   PRIMARY KEY (eventId),
   FOREIGN KEY (coordinator) REFERENCES Staff(empId) ON DELETE SET NULL,
   FOREIGN KEY (roomId) REFERENCES Room(roomId) ON DELETE CASCADE
);

CREATE TABLE Booking (
   bookingId INTEGER,
   eventId INTEGER,
   member INTEGER,
   status VARCHAR2(10) NOT NULL CHECK (status IN ('REG', 'ATT', 'NOS', 'CAN')),
   paid BOOLEAN,
   refunded BOOLEAN,

   PRIMARY KEY (bookingId),
   FOREIGN KEY (eventId) REFERENCES Event(eventId) ON DELETE CASCADE,
   FOREIGN KEY (member) REFERENCES Member(memberNum) ON DELETE CASCADE
);
```

Normalization analysis

Both PetNeed and PetTemperament are trivially in BCNF

All of the relations don't have set-valued attributes, so all are in 1NF

Pet FDs {petId→type, petId→breed, petId→age, petId→doa, petId→adoptable, petId→name}
It is in 2NF because all non-prime attributes are composed of the whole PK petId
It is in 3NF because petId is a superkey, and it composes everything

PetRoomHistorypetId FDs { {petId,roomId,startDate}→sector, {petId,roomId,startDate}→endDate }
It is in 2NF because sector and endDate are the non-prime attributes that depend on all PK petId,roomId,startId.
It is in 3NF because all dependencies have superkeys.

Room FDs {roomId→maxCapacity}
It is in 2NF because non-prime attributes just max capacity is composed of the only CK roomId
It is in 3NF because roomId is a superkey, and maxCapacity depends on it

Area FDs { {sector, roomId} → animalType, {sector, roomId} → designatedAdoptArea}
It is in 2NF because animalType and designatedAdoptArea are composed by all CK sector & roomId
It is in 3NF because all of the dependencies have a superkey on the LHS

Member FDs { memberNum→name, memberNum→tel#, memberNum→email, memberNum→dob, memberNum→membershipTier }
It is in 2NF because all non-prime attributes depend on memberNum, the single CK
It is in 3NF because all of the dependencies have a superkey on the LHS

MemberHistory FDs { {memberNum, start} → end, {memberNum, start} → tier }
It is in 2NF because all non-prime attributes depend on memberNum and start, which is all of the CK
It is in 3NF because all of the dependencies have a superkey on the LHS

EmergancyContact FDs { {memberNum, contactId} → name, {memberNum, contactId} → tel#, {memberNum, contactId} → email
It is in 2NF because all non-prime attributes depend on memberNum and contactId
It is in 3NF because all of the dependencies have a superkey on the LHS

Reservation FDs { reservationId→memberNum, reservationId→roomId, reservationId→reservationDate, reservationId→timeSlot, reservationId→checkedIn, reservationId→checkedOut}
It is In 2NF because all non-prime attributes depend on only CK reservationId
It is in 3NF because all of the dependencies have a superkey on the LHS

FoodOrder FDs {orderId→memberNum, orderId→reservationId, orderId→orderTime, orderId→totalPrice, orderId→paymentStatus}
It is in 2NF because all non-prime attributes depend on only CK orderId
It is in 3NF because all of the dependencies have a superkey on the LHS

Item FDs {itemId→price, itemId→name}
It is in 2NF because all non-prime attributes depend on only CK itemId
It is in 3NF because all of the dependencies have a superkey on the LHS

OrderItem FDs { {orderId, itemId} → quantity }

Is in 2NF because quantity is dependent off all CK
It is in 3NF because all of the dependencies have a superkey on the LHS

Staff FDs {empId→name, empId→empType}
It is in 2NF because all non-prime attributes depend on only CK empId
It is in 3NF because all of the dependencies have a superkey on the LHS

HealthRecord FDs { {recId, revNum} → revAction, {recId, revNum} → revDate, {recId, revNum} → petId, {recId, revNum} → empId, {recId, revNum} → recType, {recId, revNum} → description, {recId, revNum} → nextDue, {recId, revNum} → status }
It is in 2NF because all non-prime attributes depend on every CK recId, revNum
It is in 3NF because all of the dependencies have a superkey on the LHS

AdoptionApp FDs {appId→memberNum, appId→empId, appId→petId, appId→appDate, appid→status}
It is in 2NF because all non-prime attributes depend on only CK appId
It is in 3NF because all of the dependencies have a superkey on the LHS

Adoption FDs {adoptId→appId, adoptId→addoptDate, adoptId→fee, adoptId→followUpSchedule}
It is in 2NF because all non-prime attributes depend on only CK adoptId
It is in 3NF because all of the dependencies have a superkey on the LHS

Event FDs {eventId→coordinator, eventId→eventDate, eventId→eventTime, eventId→roomId, eventId→description, eventId→maxCapacity}
It is in 2NF because all non-prime attributes depend on only CK eventId
It is in 3NF because all of the dependencies have a superkey on the LHS

Booking FDs {bookingId→eventId, bookingId→member, bookingId→status, bookingId→paid, bookingId→refunded}
It is in 2NF because all non-prime attributes depend on only CK bookingId
It is in 3NF because all of the dependencies have a superkey on the LHS

Query Description

Our custom query that we implemented is to display adoption information that has been made within the past X month where x is the users input. This query is pretty helpful in a real world scenario because it allows the staff an easy access to how many adoptions have taken place and include this information for reporting purposes for example.