CORVIS

A Custom Varsity Athlete Meal Delivery Service

Thomas Magnusson

Table of Contents

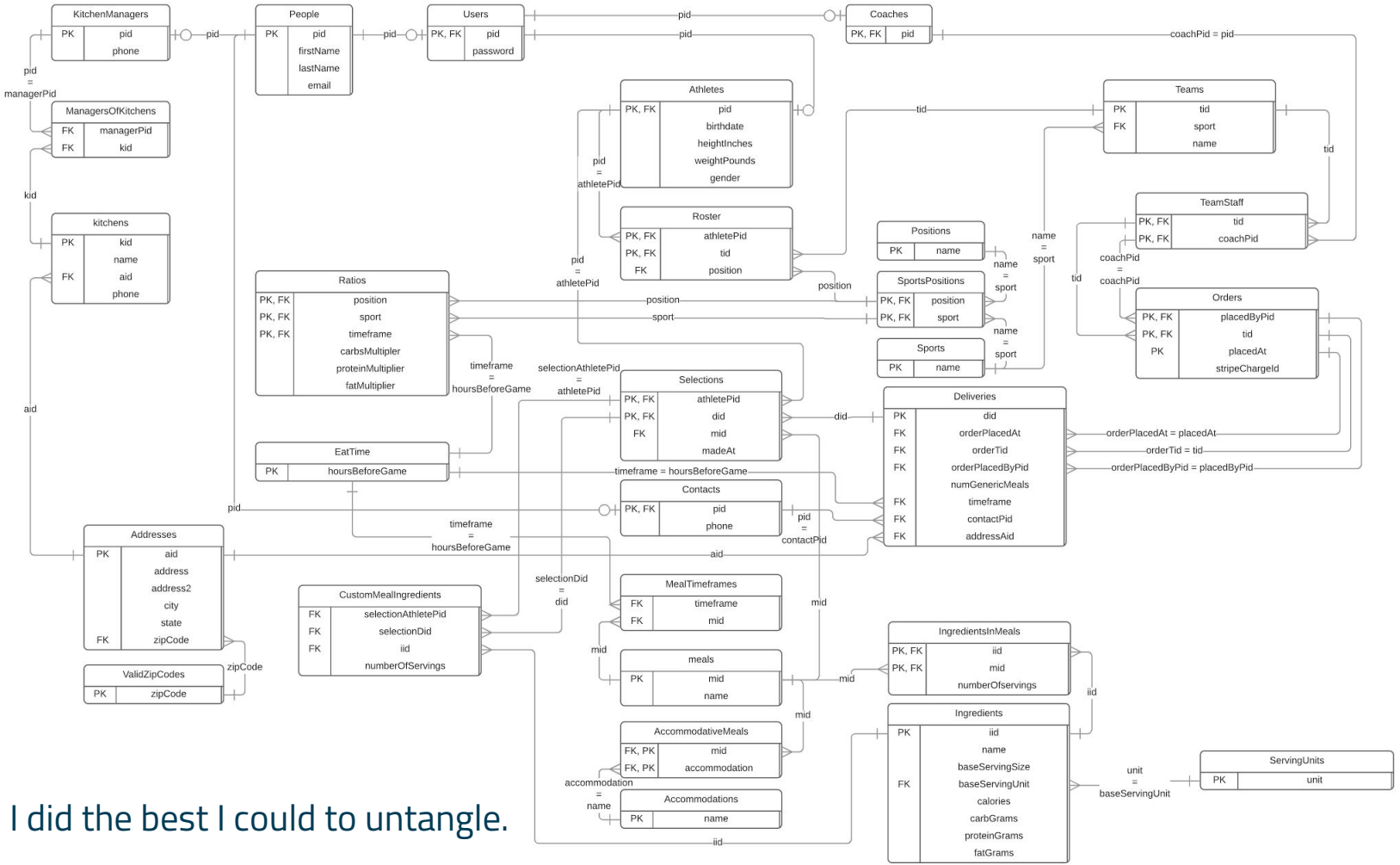
Executive Summary

Corvis is a **food delivery service that provides custom meals for athletes** based on biometrics and special formulas designed to calculate the **proper macronutrient ratios** for a given position in a sport. For example, a lineman on a football team would need lots of protein in his meal, to build muscle. An attack in lacrosse, however, might need more carbs because she would be running more often.

A coach visits the Corvis website, places an order, which can consist of many deliveries. Each delivery represents a place a team might visit, for example Fairfield University's pool. The meals will be delivered to that pool. Before the delivery date, the athletes must select the meal that they would like to receive. The meals that are offered to an athlete are based on the ratios of macronutrients needed for that athlete's position.

This is based on an actual company I've been working with. Concept used with permission

ER Diagram - See pdf file for full quality.



I did the best I could to untangle.

1. Tables

Lots of tables. Too many tables.

People

	pid	firstname	lastname	email
1	1	Tom	Magnusson	tommagnuss@gmail.com
2	8	Tom	Magnusson	tommagnuss@exmaple.com
3	9	Meat	Head	meat@example.com
4	10	Block	Head	block@example.com
5	11	Hot	Head	hot@example.com
6	12	Timmy	Foley	timmy@example.com
7	13	Brian	Damp	brian@example.com
8	14	Jack	Potenza	jack@example.com
9	15	Spencer	Foley	spencer@example.com
10	16	Matt	Clyne	matt@example.com
11	17	Daisy	Chu	daisy@example.com
12	18	Tanya	Elizabeth	tanya@example.com
13	19	Constantina	Marsden	dina@example.com
14	20	Taylor	Vahey	taylor@example.com
15	21	Andria	Lussier	andria@example.com
16	22	Jessica	Silver	jess@example.com
17	23	Emma	Litt	emma@example.com
18	24	Kim	Dionne	kim@example.com
19	25	Alexis	LaPlace	alexis@example.com
20	26	allison	Stall	allison@example.com
21	27	Marcus	Liu	marcus@example.com
22	28	Joey	Marie	joey@example.com
23	29	Megan	Barr	megan@example.com
24	30	Daniella	Dolce	daniella@example.com
25	31	Jeff	Ni	jeff@example.com
26	32	Frank	Hartman	frank@example.com
27	33	Charlotte	Harris	charlotte@example.com
28	34	Alan	Labouseur	alan@example.com
29	35	Steve	Cornish	steve@example.com

Functional Dependencies

(pid) → firstName, lastName, email

Description

Every person in this database has a first name, last name and email. The base entity type for lots of entity subtypes (see comments below).

```
-- All the people in the database, including but not limited to
--   Users, which are Athletes and Coaches
--   Contacts, who are the contact for a given Delivery
--   KitchenManagers, who manage the kitchens (obviously)
CREATE TABLE People (
  pid SERIAL PRIMARY KEY,
  firstName TEXT NOT NULL,
  lastName TEXT NOT NULL,
  email TEXT NOT NULL,
);
```

Users

pid	password
1	hashedPassword
9	password
10	asdf1234
11	hunter2
12	something
13	#hashed
14	pass
15	luvfootball
16	somethingsecure
17	somethingsecure
18	asdfjlk1234321
19	hashed##
20	something!!!
21	what is a password?
22	why are passwords a thing?
23	probably for security reasons
24	companionCube
25	lorem ipsum
26	I should figure out how to generate these
27	there is probably something out there
29	to do that, but here is some releif
30	from grading hopefully
31	anyway hows blockchain going?
32	its going okay for me
33	lots of boilerplate
34	alpaca

Functional Dependencies

(pid) → password

Description

A subtype of Person. All users have to log onto the website to use the services, so they must provide a password. Ideally the application the password into the database will hash it.

```sql
-- People who use the Corvis website or mobile app.
-- Limited to: Athletes and Coaches.
CREATE TABLE Users (
  pid INTEGER PRIMARY KEY REFERENCES People(pid),
  password TEXT NOT NULL
);
```

Coaches

Functional Dependencies

(pid) →

Description

A subtype of a user. Represents a coach. A coach is the coach of a team and has the power to place orders.

pid
1
34

```
-- Coaches are the primary users of Corvis.
-- They can order food for everyone.
CREATE TABLE Coaches (
  pid INTEGER PRIMARY KEY REFERENCES Users(pid)
);
```

Athletes

pid	birthda...	heightinches	weightpounds	gender
9	1998-12-01	70	200	male
10	1998-12-17	75	190	male
11	1998-07-07	68	150	male
12	1998-11-18	70	200	male
13	1996-03-11	71	170	male
14	1994-03-22	67	150	male
15	1998-08-04	69	156	male
16	1998-12-24	66	153	male
17	1998-03-16	62	135	female
18	1998-12-21	70	160	female
19	1997-05-24	75	210	female
20	1998-05-31	77	200	female
21	1996-03-13	75	240	female
22	1997-01-03	67	140	female
23	1996-04-16	66	155	female
24	1996-03-03	66	153	female
25	1998-12-17	68	147	female
26	1995-09-16	63	136	female
28	1996-02-14	63	140	male
29	1997-11-03	64	133	female
30	1997-04-01	67	146	female
31	1997-12-25	66	158	male
32	1998-10-31	69	170	male
33	1996-08-19	60	191	female
34	1998-07-13	62	122	male
35	1994-07-15	60	134	male

Functional Dependencies

(pid) → birthdate, heightInches, weightPounds, gender

Description

Athletes select meals per delivery. They belong to a team. Corvis takes in their biometrics to use for calculating the proper meals. Gender uses a **check constraint**.

```sql
-- Athletes select a meal out of a bunch of possible meals
-- generated based on their position on a sport.
CREATE TABLE Athletes (
  pid INTEGER PRIMARY KEY REFERENCES Users(pid),
  birthdate DATE NOT NULL,
  heightInches INTEGER NOT NULL,
  weightPounds INTEGER NOT NULL,

  -- Athletes participate in either male or female sports, appropriate dichotomy.
  gender TEXT NOT NULL CHECK (gender IN ('male', 'female'))
);
```

Sports

name
Football
Soccer
Volleyball
Water Polo
Lacrosse

Functional Dependencies

(name) →

Description

An alternative to a check constraint. A list of valid **sports** about which Corvis has macronutrient ratios.

```
-- Like, you know, Baseball and all that.
CREATE TABLE Sports (
  name TEXT PRIMARY KEY
);
```

Positions

name
Lineman
Quarterback
Running Back
Wide Receiver
Defender
Middy
Attack

Functional Dependencies

(name) →

Description

An alternative to a check constraint. A list of valid **positions** about which Corvis has macronutrient ratios.

```sql
-- Positions (for a sport)
CREATE TABLE Positions (
  name TEXT NOT NULL PRIMARY KEY
);
```

SportsPositions

Functional Dependencies

(sportsName, positionName) →

Description

Relates a position to the sport to which it belongs. It doesn't seem like a many to many relationship, but the same position might appear in many different sports.

sportname	positionsname
Football	Lineman
Football	Quarterback
Football	Running Back
Football	Wide Receiver
Lacrosse	Defender
Lacrosse	Attack
Lacrosse	Middy

```sql
-- Relates sports to a position.
-- Seems like it's not a many to many relation,
-- but the same position name might be
-- in different sports.
CREATE TABLE SportPositions (
  sportName TEXT NOT NULL REFERENCES Sports(name),
  positionsName TEXT NOT NULL REFERENCES Positions(name),

  PRIMARY KEY(sportName, positionsName)
);
```

Teams

Functional Dependencies

(tid) → sport, name

Description

Teams that coaches create. They have many players.

```sql
-- Like Da Bears
CREATE TABLE Teams (
  tid SERIAL PRIMARY KEY,
  sport TEXT REFERENCES Sports(name),
  name TEXT NOT NULL
);
```

tid	sport	name
1	Football	Wildcats
2	Football	Stags
3	Lacrosse	Knighthawks

TeamStaff

Functional Dependencies

(coachPid, tid) →

Description

Which coaches are on what team. Many to many relationship.

coachpid	tid
1	1
34	3

```
-- The coaches for a given team.
CREATE TABLE TeamStaff (
  coachPid INTEGER REFERENCES Coaches(pid),
  tid INTEGER REFERENCES Teams(tid),

  PRIMARY KEY(coachPid, tid)
);
```

Roster

athletepid	tid	position
9	1	Lineman
10	1	Lineman
11	1	Lineman
12	1	Quarterback
13	1	Running Back
14	1	Running Back
15	1	Wide Receiver
17	3	Defender
18	3	Defender
19	3	Attack
20	3	Attack
21	3	Attack
22	3	Attack
23	3	Middy
24	3	Middy
25	3	Middy
26	3	Middy

Functional Dependencies

(athletePid, tid) → position

Description

Relates athletes to teams, which determine what position that athlete is. Corvis does not allow multi-position athletes on the same team.

```sql
-- A specific athlete on a team,
-- and what that athlete's position is.
CREATE TABLE Roster (
  athletePid INTEGER NOT NULL REFERENCES Athletes(pid),
  tid INTEGER NOT NULL REFERENCES Teams(tid),
  position TEXT NOT NULL REFERENCES Positions(name),

  PRIMARY KEY(athletePid, tid)
);
```

ValidZipCodes

zipcode
15007
15008
15009
16210
19019

```
-- Corvis only has a finite number
-- of available zip codes for delivery.
CREATE TABLE ValidZipCodes (
  zipCode TEXT NOT NULL PRIMARY KEY
);
```

Functional Dependencies

(zipCode) →

Description

Constant list of zip codes to which Corvis can deliver.

Addresses

aid	address	address2	city	state	zipcode
2	7 Bond Lane	<null>	Bakerstown	PA	15007
3	42 Galaxy St.	<null>	Adrian	PA	16210
4	1 Baking Avenue	<null>	Philadelphia	PA	19019

Functional Dependencies

(aid) → address, address2, city, state, zipCode

Description

The addresses that are within the valid zip codes to which corvis can deliver.

```
-- Addresses for kitchens and
-- deliveries.
CREATE TABLE Addresses (
  aid SERIAL NOT NULL PRIMARY KEY,
  address TEXT NOT NULL,
  address2 TEXT, -- nullable because not all addresses have second line
  city TEXT NOT NULL,
  state TEXT NOT NULL,
  zipCode TEXT NOT NULL REFERENCES ValidZipCodes(zipCode)
);
```

ServingUnits

unit
grams
slices
ounces
fluid ounces
cups
eggs
tablespoons
breast

Functional Dependencies

(unit) →

Description

The units of measure for ingredients in the ingredients table. Lets the kitchen know how much to make of what.

```
-- Like oz, dollops, slices, etc.
CREATE TABLE ServingUnits (
  unit TEXT NOT NULL PRIMARY KEY
);
```

Ingredients

```sql
-- For a meal. Base serving size because an Ingredient like Ham
-- might require a multiple amount of the base serving size, which
-- would probably be 1 oz or something like that. Other Ingredients,
-- like Gatorade, would have base serving size as 1 and
-- the serving unit as 8oz. Bottle.
CREATE TABLE Ingredients (
  iid SERIAL PRIMARY KEY,
  name TEXT NOT NULL,

  baseServingSize DECIMAL NOT NULL,
  baseServingUnit TEXT NOT NULL REFERENCES ServingUnits(unit),

  calories INTEGER NOT NULL,

  carbGrams DECIMAL NOT NULL,
  proteinGrams DECIMAL NOT NULL,
  fatGrams DECIMAL NOT NULL
);
```

.	name	baseservingsize	baseservingunit	calories	carbgrams	proteingrams	fatgrams
1	Ham	100	grams	145	1.5	21	5.5
2	Turkey	100	grams	100	4.2	17	1.6
3	Red Gatorade	20	fluid ounces	125	35	0	0
4	Whole Wheat Bread	2	slices	140	13	3	1
5	Lettuce	2.5	ounces	5	1	1	1
6	Mayonnaise	1	ounces	175	0	0	19
7	Lays Classic Chips	1	ounces	185	5	19	19
10	Spaghetti	0.5	cups	110	22	4	0.5
11	Meat Sauce	1	cups	275	58	12	2
12	Water	8	ounces	0	0	0	0
13	Morrison Meatballs	4	ounces	1220	124	50	61
14	Egg Yolk	1	eggs	60	1	7	2
15	Olive Oil	1	tablespoons	340	20	15	22
16	Walnuts	1	cups	1870	89	118	102
17	Roasted Chicken Breast	1	breast	1470	188	185	11
18	Broccoli	100	grams	34	7	3	0

Functional Dependencies

(iid) → name, baseServingSize, baseServingUnit, calories, carbGrams, proteinGrams, fatGrams

Description

Ingredients for the custom meals, along with the nutritional information pertinent to the Corvis algorithm.

Meals

mid	name
1	Classic Turkey Sandwich
3	Caesar Salad
4	Chicken Dinner
2	Classic Spaghetti
5	Meatball Spaghetti

Functional Dependencies

(mid) → name

Description

The promotional meals that Corvis offers. Meals have ingredients, whose serving sizes can be tweaked to meet the ratio for a given athlete's selection.

```
-- Promotional meals, like "Delicious Ham Sandwich"
-- See IngredientsInMeals to find out what are in these meals.
CREATE TABLE Meals (
  mid SERIAL PRIMARY KEY,
  name TEXT NOT NULL
);
```

IngredientsInMeals

iid	mid	numberofservings
3	1	1
4	1	1
7	1	1
12	2	2
12	3	2
14	3	2
15	3	3
16	3	0.25
12	4	2
18	4	2
10	5	4
2	1	<null>
5	1	<null>
17	4	<null>
13	5	<null>
6	1	<null>
10	2	<null>
11	2	<null>

Functional Dependencies

(iid, mid) → numberOfServings

Description

The number of servings of an ingredient in a meal. Null if the number of servings can be tweaked to fit a ratio. See CustomMealIngredients for more.

```sql
-- The ingredients that go into a meal.
CREATE TABLE IngredientsInMeals (
  iid INTEGER NOT NULL REFERENCES Ingredients(iid),
  mid INTEGER NOT NULL REFERENCES Meals(mid),

  -- Nullable, null if this ingredient is a variable one,
  -- meaning the serving size can be tweaked to fit a position's
  -- ratio profile.
  numberOfServings DECIMAL,

  PRIMARY KEY(iid, mid)
);
```

Accommodations

Functional Dependencies

(name) →

Description

Constant list of meals that conform to an accommodation.

name
Nut Free
Gluten Free
Vegan
Vegetarian

```sql
-- Gluten Free, Nut Free, etc.
CREATE TABLE Accommodations (
  name TEXT NOT NULL PRIMARY KEY
);
```

AccommodativeMeals

mid	accommodation
1	Nut Free
2	Nut Free
4	Nut Free
5	Nut Free

Functional Dependencies

(mid, accommodation) →

Description

Marks which meals adhere to what accommodation.

```sql
-- A meal that satisfies an Accommodation, like gluten free.
CREATE TABLE AccommodativeMeals (
  mid INTEGER NOT NULL REFERENCES Meals(mid),
  accommodation TEXT NOT NULL REFERENCES Accommodations(name),

  PRIMARY KEY(mid, accommodation)
);
```

Orders

tid	placedbypid	placedat	stripechargeid
1	1	2017-11-06 16:29:24.569000	fakeStripeCharge1
1	1	2017-11-06 16:40:04.324000	fakeStripeCharge2
3	34	2017-12-02 16:33:18.976000	fakeStripeCharge4

```sql
-- Coaches place Orders, each of which can have many deliveries
CREATE TABLE Orders (
  tid INTEGER NOT NULL REFERENCES Teams(tid),
  placedByPid INTEGER NOT NULL REFERENCES Coaches(pid),
  placedAt TIMESTAMP NOT NULL,

  -- The id for Stripe, an online payment service.
  -- Paying with a card online generates and id.
  stripeChargeId TEXT NOT NULL UNIQUE,

  PRIMARY KEY(tid, placedByPid, placedAt)
);
```

Functional Dependencies

(tid, placedByPid, placedAt) → stripeChargeId

Description

Orders are placed by coaches. They consist of many deliveries. Stripe is an online payment service that handles the transactions with credit cards.

EatTime

hoursbeforegame
-2
-1
0
1
2
3
4
5
6

Functional Dependencies

(hoursBeforeGame) →

Description

A list of constants that represent how many hours before a game a delivery is being brought to the venue. The kind of meal the athlete should consume changes based on this factor.

```sql
-- The timeframe the athletes will be eating the meals before (or after)
-- the event in which they are participating.
CREATE TABLE EatTime (
  hoursBeforeGame INTEGER NOT NULL PRIMARY KEY
);
```

Contacts

Functional Dependencies

(pid) → phone

Description

A contact is a person who should be contacted if there are problems with a delivery.

pid	phone
1	205-555-5555
34	204-1515-2000

```
-- A person who is a contact for a delivery in case the kitchens
-- need to contact a representative on the team besides the coach.
-- A contact can actually be a coach in this database design :).
CREATE TABLE Contacts (
  pid INTEGER NOT NULL PRIMARY KEY REFERENCES People(pid),
  phone TEXT NOT NULL
);
```

Deliveries

Functional Dependencies

(did) → orderTid, orderPlacedByPid, orderPlacedAt, numGenericMeals, hoursbeforeGame, contactPid, addressAid, eventTime

Description

Represents a specific time and place to deliver the meals for a team. Generic meals are non-custom meals that are for coaching and staff. Many deliveries are part of one order.

```sql
-- A delivery represents an event to be delivered to.
-- Part of a single Order.
CREATE TABLE Deliveries (
  did SERIAL NOT NULL PRIMARY KEY,

  orderTid INTEGER NOT NULL,
  orderPlacedByPid INTEGER NOT NULL,
  orderPLacedAt TIMESTAMP NOT NULL,
  FOREIGN KEY (orderTid, orderPlacedByPid, orderPlacedAt)
  REFERENCES Orders(tid, placedByPid, placedAt),

  numGenericMeals INTEGER NOT NULL,
  hoursBeforeGame INTEGER NOT NULL REFERENCES EatTime (hoursBeforeGame),
  contactPid INTEGER NOT NULL REFERENCES Contacts(pid),
  addressAid INTEGER NOT NULL REFERENCES Addresses(aid),
  eventTime TIMESTAMP NOT NULL
);
```

did	eventtime	ordertid	orderplacedbypid	orderplacedat	numgenericmeals	hoursbeforegame	contactpid	addressaid
1	2017-12-17 16:29:24.569000	1	1	2017-11-06 16:29:24.569000	1	2	1	2
2	2017-12-19 16:29:24.569000	1	1	2017-11-06 16:29:24.569000	1	3	1	2
3	2017-12-20 16:29:24.569000	1	1	2017-11-06 16:40:04.324000	2	2	1	3
4	2017-12-21 16:29:24.569000	3	34	2017-12-02 16:33:18.976000	3	-2	34	3

Selections

athletepid	did	mid	madeat
10	1	2	2017-12-03 13:51:22.715000
15	1	1	2017-12-03 08:53:33.599000
9	1	1	2017-12-03 14:53:53.809000
10	2	3	2017-12-03 16:55:43.052000
15	2	1	2017-12-03 07:55:45.899000
20	4	2	2017-12-03 05:57:21.781000
26	4	4	2017-12-03 12:59:58.734000

```sql
-- An Athlete selects a meal for a given delivery. This is the meal
-- they will receive for that delivery.
CREATE TABLE Selections (
  athletePid INTEGER NOT NULL REFERENCES Athletes(pid),
  did INTEGER NOT NULL REFERENCES Deliveries(did),

  -- nullable, if the athlete has not made a selection yet.
  mid INTEGER DEFAULT NULL REFERENCES Meals(mid),
  madeAt TIMESTAMP NOT NULL,

  PRIMARY KEY(athletePid, did)
);
```

Functional Dependencies

(athletePid, did) → mid, madeAt

Description

Once an order is placed, a selection is created for all of the athletes that are attending the event to Corvis will be delivering. The meal has a **default** of null because the selected meal for the event is unknown. Once athletes choose their meal, the mid will be updated according to the chosen meal.

CustomMealIngredients

selectionathletepid	selectiondid	iid	numberofservings
10	1	10	2.3
10	1	11	3.8
15	1	2	2
15	1	5	2.4
9	1	5	2
9	1	2	1.5
15	2	2	2
15	2	5	2.4
15	2	6	3
15	1	6	3
20	4	10	3.2
20	4	11	2.1
26	4	17	2.7

```
-- Ingredients can be converted to custom ingredients,
-- in order to fit the ratio of the specific athlete's position.
-- Ham, for example, can be sliced to any weight, therefore it can be
-- used to further balance a meal's ratio, rather than a bottle of gatorade,
-- which cannot be "split" (it has a set number of calories and macronutrients).
CREATE TABLE CustomMealIngredients (
  selectionAthletePid INTEGER NOT NULL,
  selectionDid INTEGER NOT NULL,
  FOREIGN KEY (selectionAthletePid, selectionDid)
  REFERENCES Selections(athletePid, did),

  iid INTEGER NOT NULL REFERENCES Ingredients(iid),
  numberOfServings DECIMAL NOT NULL DEFAULT 1,

  PRIMARY KEY (selectionAthletePid, selectionDid, iid)
);
```

Functional Dependencies

(selectionAthletePid,selectionDid, iid) → numberOfServings

Description

For the ingredients whose servings in a meal are **null** are slated to be filled in by the Corvis algorithm in this table. For example, more Turkey could be added to a Turkey Sandwich to increase the amount of protein for a lineman.

MealTimeframes

hoursbeforegame	mid
1	1
2	1
3	1
4	1
1	3
2	3
3	3
-2	4
-1	4
-2	2
-1	2
5	5
4	5

Functional Dependencies

(hoursBeforeGame, mid) →

Description

Which meals are appropriate for a given timeframes (eat time).

```
-- Which meals are meant for which time frames.
-- Lots of pasta 6 hours before a game is better than 2 hours after.
CREATE TABLE MealTimeframes (
  hoursBeforeGame INTEGER NOT NULL REFERENCES EatTime (hoursBeforeGame),
  mid INTEGER NOT NULL REFERENCES Meals(mid),

  PRIMARY KEY(hoursBeforeGame, mid)
);
```

KitchenManagers

pid	phone
1	203-555-2077
34	555-555-4242
35	602-555-9090

Functional Dependencies

(pid) → phone

Description

Kitchen managers are kept within the database as a referential catalog to be displayed on the website. Phone is the phone number of the manager, which the directory might not know (therefore it's nullable).

```
-- Who manages the kitchens?
CREATE TABLE KitchenManagers (
  pid INTEGER NOT NULL PRIMARY KEY REFERENCES People(pid),

  -- Nullable, a directory where we might not know the phone number of a manager
  phone TEXT
);
```

Kitchens

Functional Dependencies

(kid) → name, aid, phone

Description

Kitchens are where the food is prepared! They must have an address and a phone number on record.

kid	name	aid	phone
1	Primary Kitchen	4	255-555-1414
2	Auxiliary Kitchen	4	818-555-1690

```sql
-- Where the food is processed. Needed to provide contact information
-- to anyone needing further assistance.
CREATE TABLE Kitchens (
  kid SERIAL NOT NULL PRIMARY KEY,
  name TEXT NOT NULL,
  aid INTEGER NOT NULL REFERENCES Addresses(aid),
  phone TEXT NOT NULL
);
```

ManagersOfKitchens

Functional Dependencies

(managerPid, kid) →

Description

Associative entity describing which managers manage which kitchens.

managerpid	kid
1	1
1	2
34	2
35	1

```
-- Managers and the kitchens they manage.
CREATE TABLE ManagersOfKitchens (
  managerPid INTEGER NOT NULL REFERENCES KitchenManagers(pid),
  kid INTEGER NOT NULL REFERENCES Kitchens(kid),

  PRIMARY KEY(managerPid, kid)
);
```

Ratios

```sql
-- The ratios of macronutrients necessary for a given
-- position within a sport for a meal.
-- Also changes based on the eat time (when the athletes
-- will consume the meal in relation to the event they are
-- competing in).
CREATE TABLE Ratios (
  position TEXT NOT NULL,
  sport TEXT NOT NULL,
  FOREIGN KEY (position, sport)
  REFERENCES SportPositions(positionsName, sportName),

  hoursBeforeGame INTEGER NOT NULL REFERENCES EatTime (hoursBeforeGame),

  carbsMultiplier DECIMAL NOT NULL,
  proteinMultiplier DECIMAL NOT NULL,
  fatMultiplier DECIMAL NOT NULL,

  PRIMARY KEY(position, sport, hoursBeforeGame)
);
```

position	sport	hoursbeforegame	carbsmultiplier	proteinmultiplier	fatmultiplier
Lineman	Football	1	2	2	1
Lineman	Football	2	1.5	2	1
Lineman	Football	3	1	3	1
Middy	Lacrosse	-1	2.5	1	1
Attack	Lacrosse	5	2.3	3.4	1

Functional Dependencies

(position, sport, hoursBeforeGame) → carbsMultiplier, proteinMultiplier, fatMultiplier

Description

The ratios of carbs to protein to fat for a given position in a sport a certain time away from the game. This is the most important data in the database because the ratios are trade secrets.

I used made up data though, so don't get any ideas about stealing it!

2.
Views

Because we have to make sense of this normalization stuff.

AllAthletes

Description

Saving some typing for the application developers.

```sql
-- All Athletes allows programmers to avoid inner joins.
CREATE VIEW AllAthletes
AS
SELECT p.pid, p.firstName, p.lastName, p.email, a.birthdate, a.gender, a.heightInches, a.weightPounds
FROM People p
INNER JOIN Athletes a
  ON p.pid = a.pid;
```

pid	firstname	lastname	email	birthdate	gender	heightinches	weightpounds
9	Meat	Head	meat@example.com	1998-12-01	male	70	200
10	Block	Head	block@example.com	1998-12-17	male	75	190
11	Hot	Head	hot@example.com	1998-07-07	male	68	150
12	Timmy	Foley	timmy@example.com	1998-11-18	male	70	200
13	Brian	Damp	brian@example.com	1996-03-11	male	71	170
14	Jack	Potenza	jack@example.com	1994-03-22	male	67	150
15	Spencer	Foley	spencer@example.com	1998-08-04	male	69	156
16	Matt	Clyne	matt@example.com	1998-12-24	male	66	153
17	Daisy	Chu	daisy@example.com	1998-03-16	female	62	135
18	Tanya	Elizabeth	tanya@example.com	1998-12-21	female	70	160
19	Constantina	Marsden	dina@example.com	1997-05-24	female	75	210
20	Taylor	Vahey	taylor@example.com	1998-05-31	female	77	200
21	Andria	Lussier	andria@example.com	1996-03-13	female	75	240
22	Jessica	Silver	jess@example.com	1997-01-03	female	67	140
23	Emma	Litt	emma@example.com	1996-04-16	female	66	155
24	Kim	Dionne	kim@example.com	1996-03-03	female	66	153
25	Alexis	LaPlace	alexis@example.com	1998-12-17	female	68	147
26	allison	Stall	allison@example.com	1995-09-16	female	63	136
28	Joey	Marie	joey@example.com	1996-02-14	male	63	140
29	Megan	Barr	megan@example.com	1997-11-03	female	64	133
30	Daniella	Dolce	daniella@example.com	1997-04-01	female	67	146
31	Jeff	Ni	jeff@example.com	1997-12-25	male	66	158
32	Frank	Hartman	frank@example.com	1998-10-31	male	69	170
33	Charlotte	Harris	charlotte@example.com	1996-08-19	female	60	191
34	Alan	Labouseur	alan@example.com	1998-07-13	male	62	122
35	Steve	Cornish	steve@example.com	1994-07-15	male	60	134

AllCoaches

Description

More saving the developers some typing...

```sql
-- All Coaches allow programmers to avoid inner joins.
CREATE VIEW AllCoaches
AS
SELECT p.*
FROM People p
INNER JOIN Coaches c
  ON p.pid = c.pid;
```

1	Tom	Magnusson	tommagnuss@gmail.com
34	Alan	Labouseur	alan@example.com

AllIngredientsOfAllMeals

```sql
CREATE VIEW AllIngredientsOfAllMeals
AS
SELECT
  m.mid,
  m.name AS mealName,
  i.iid,
  i.name AS ingredientName,
  i.baseServingSize,
  i.baseServingUnit,
  i.calories,
  i.proteinGrams,
  i.carbGrams,
  i.fatGrams
FROM Ingredients i
INNER JOIN IngredientsInMeals iim USING (iid)
INNER JOIN Meals m USING (mid)
ORDER BY mid ASC;
```

Description

All the ingredients that are in all of the meals. This is not how much of each ingredient are in each meal. This view serves as a naming convenience for other views.

mid	mealname	iid	ingredientname	baseservingsize	baseservingunit	calories	proteingrams	carbgrams	fatgrams
1	Classic Turkey Sandwich	5	Lettuce	2.5	ounces	5	1	1	1
1	Classic Turkey Sandwich	2	Turkey	100	grams	100	17	4.2	1.6
1	Classic Turkey Sandwich	7	Lays Classic Chips	1	ounces	185	19	5	19
1	Classic Turkey Sandwich	4	Whole Wheat Bread	2	slices	140	3	13	1
1	Classic Turkey Sandwich	3	Red Gatorade	20	fluid ounces	125	0	35	0
1	Classic Turkey Sandwich	6	Mayonnaise	1	ounces	175	0	0	19
2	Classic Spaghetti	11	Meat Sauce	1	cups	275	12	58	2
2	Classic Spaghetti	12	Water	8	ounces	0	0	0	0
2	Classic Spaghetti	10	Spaghetti	0.5	cups	110	4	22	0.5
3	Caesar Salad	12	Water	8	ounces	0	0	0	0
3	Caesar Salad	14	Egg Yolk	1	eggs	60	7	1	2
3	Caesar Salad	16	Walnuts	1	cups	1870	118	89	102
3	Caesar Salad	15	Olive Oil	1	tablespoons	340	15	20	22
4	Chicken Dinner	12	Water	8	ounces	0	0	0	0
4	Chicken Dinner	17	Roasted Chicken Breast	1	breast	1470	185	188	11
4	Chicken Dinner	18	Broccoli	100	grams	34	3	7	0
5	Meatball Spaghetti	10	Spaghetti	0.5	cups	110	4	22	0.5

AccurateAllIngredientsOfAllMeals

```sql
CREATE VIEW AccurateAllIngredientsOfAllMeals
AS
SELECT
  a.mid,
  a.mealName,
  a.iid,
  a.ingredientName,
  COALESCE(iim.numberOfServings * a.baseServingSize, a.baseServingSize) AS numberOfServings,
  a.baseServingUnit,
  COALESCE(iim.numberOfServings * a.calories, a.calories) AS calories,
  COALESCE(iim.numberOfServings * a.proteinGrams, a.proteinGrams) AS proteinGrams,
  COALESCE(iim.numberOfServings * a.carbGrams, a.carbGrams) AS carbGrams,
  COALESCE(iim.numberOfServings * a.fatGrams, a.fatGrams) AS fatGrams
FROM AllIngredientsOfAllMeals a
INNER JOIN IngredientsInMeals iim
ON iim.iid = a.iid AND iim.mid = a.mid
ORDER BY mid;
```

mid	mealname	iid	ingredientname	numberofservings	baseservingunit	calories	proteingrams	carbgrams	fatgrams
1	Classic Turkey Sandwich	3	Red Gatorade	20	fluid ounces	125	0	35	0
1	Classic Turkey Sandwich	2	Turkey	100	grams	100	17	4.2	1.6
1	Classic Turkey Sandwich	4	Whole Wheat Bread	2	slices	140	3	13	1
1	Classic Turkey Sandwich	5	Lettuce	2.5	ounces	5	1	1	1
1	Classic Turkey Sandwich	6	Mayonnaise	1	ounces	175	0	0	19
1	Classic Turkey Sandwich	7	Lays Classic Chips	1	ounces	185	19	5	19
2	Classic Spaghetti	12	Water	16	ounces	0	0	0	0
2	Classic Spaghetti	10	Spaghetti	0.5	cups	110	4	22	0.5
2	Classic Spaghetti	11	Meat Sauce	1	cups	275	12	58	2
3	Caesar Salad	14	Egg Yolk	2	eggs	120	14	2	4
3	Caesar Salad	15	Olive Oil	3	tablespoons	1020	45	60	66
3	Caesar Salad	12	Water	16	ounces	0	0	0	0
3	Caesar Salad	16	Walnuts	0.25	cups	467.5	29.5	22.25	25.5
4	Chicken Dinner	18	Broccoli	200	grams	68	6	14	0
4	Chicken Dinner	12	Water	16	ounces	0	0	0	0
4	Chicken Dinner	17	Roasted Chicken Breast	1	breast	1470	185	188	11
5	Meatball Spaghetti	10	Spaghetti	2	cups	440	16	88	2
5	Meatball Spaghetti	13	Morrison Meatballs	4	ounces	1220	50	124	61

Description

This view contains all the accurate counts of the nutritional value for all the meals. If the ingredient is null for a meal, this view assumes there the number of servings is the base serving size.

3.
Reports

We want (normal people) to actually understand what's in the database.

BaselineMealNutrition

Description

This report is a view that reports the baseline ingredient statistics for all the meals.

This is useful for estimating how close to a given athlete ratio.

```sql
-- Report for the nutrition statistics
CREATE VIEW BaselineMealNutrition
AS
SELECT
  mid,
  mealName,
  COUNT(iid) AS numberOfIngredients,
  SUM(calories) AS totalCalories,
  SUM(proteinGrams) AS totalProteinGrams,
  SUM(carbGrams) AS totalCarbGrams,
  SUM(fatGrams) AS totalFatGrams,
  SUM(proteinGrams + carbGrams + fatGrams) AS totalMacros
FROM AccurateAllIngredientsOfAllMeals
GROUP BY (mid, mealName);
```

mid	mealname	numberofingredients	totalcalories	totalproteingrams	totalcarbgrams	totalfatgrams	totalmacros
1	Classic Turkey Sandwich	6	730	40	58.2	41.6	139.8
2	Classic Spaghetti	3	385	16	80	2.5	98.5
3	Caesar Salad	4	1607.5	88.5	84.25	95.5	268.25
4	Chicken Dinner	3	1538	191	202	11	404
5	Meatball Spaghetti	2	1660	66	212	63	341

UndeliveredSelections

```sql
-- Pending selections...
CREATE VIEW UndeliveredSelections
AS
SELECT
  p.firstName || ' ' || p.lastName AS athleteName,
  position,
  m.name,
  hoursBeforeGame,
  pe.firstName || ' ' || pe.lastName AS contactName,
  c.phone AS contactPhoneNumber,
  address,
  coalesce(address2, '') AS address2,
  city,
  state,
  zipcode,
  (d.eventTime - NOW()) AS countDown
FROM Selections s
INNER JOIN Deliveries d
  USING (did)
INNER JOIN People p
  ON p.pid = s.athletePid
INNER JOIN Roster r
  USING (athletePid)
INNER JOIN Addresses a
  ON a.aid = d.addressAid
INNER JOIN Meals m
  USING (mid)
INNER JOIN People pe
  ON d.contactPid = pe.pid
INNER JOIN Contacts c
  ON pe.pid = c.pid
WHERE d.eventTime > NOW();
```

athletename	position	name	hoursbeforegame	contactname	contactphonenumber	address	address2	city	state	zipcode	countdown
Meat Head	Lineman	Classic Turkey Sandwich	2	Tom Magnusson	205-555-5555	7 Bond Lane		Bakerstown	PA	15007	0 years 0 mons 12 days 18 hours 55 mins 17.791298 secs
Block Head	Lineman	Caesar Salad	3	Tom Magnusson	205-555-5555	7 Bond Lane		Bakerstown	PA	15007	0 years 0 mons 14 days 18 hours 55 mins 17.791298 secs
Block Head	Lineman	Classic Spaghetti	2	Tom Magnusson	205-555-5555	7 Bond Lane		Bakerstown	PA	15007	0 years 0 mons 12 days 18 hours 55 mins 17.791298 secs
Spencer Foley	Wide Receiver	Classic Turkey Sandwich	3	Tom Magnusson	205-555-5555	7 Bond Lane		Bakerstown	PA	15007	0 years 0 mons 14 days 18 hours 55 mins 17.791298 secs
Spencer Foley	Wide Receiver	Classic Turkey Sandwich	2	Tom Magnusson	205-555-5555	7 Bond Lane		Bakerstown	PA	15007	0 years 0 mons 12 days 18 hours 55 mins 17.791298 secs
Taylor Vahey	Attack	Classic Spaghetti	-2	Alan Labouseur	204-1515-2000	42 Galaxy St.		Adrian	PA	16210	0 years 0 mons 16 days 18 hours 55 mins 17.791298 secs
allison Stall	Middy	Chicken Dinner	-2	Alan Labouseur	204-1515-2000	42 Galaxy St.		Adrian	PA	16210	0 years 0 mons 16 days 18 hours 55 mins 17.791298 secs

Description

Extremely useful report for the kitchens. They get to see which players have selections that are pending delivery, and the countdown to when that player needs what meal.

Includes useful contact phone number and the address of the event.

4.

Stored Procedures

Coding in that god awful plpgsql.

averageCaloricConsumption

```
-- Reports an athlete's average caloric consumption for all selections.
CREATE OR REPLACE FUNCTION averageCaloricConsumption(pidOfAthlete INTEGER)
  RETURNS INTEGER AS $$
DECLARE
  avg INTEGER;
BEGIN
  SELECT AVG(totalCalories)
  INTO avg
  FROM Selections
  INNER JOIN BaselineMealNutrition USING (mid)
  GROUP BY Selections.athletePid
  HAVING Selections.athletePid = pidOfAthlete;
  return avg;
END;
$$ LANGUAGE 'plpgsql';

SELECT averageCaloricConsumption(15);
```

averagecaloricconsumption
730

Description

Useful analytics on how many calories a given athlete consumes over all the selected meals.

Could be used to to better understand which athletes are eating larger meals.

macronutrientPercentageOfMeal

Description

The percentage for each macronutrient of the total grams of macronutrients.

Useful for comparing meals to the ratios table and selecting candidate meals for the given ratios.

```
-- Find the macronutrient percentage of a meal
CREATE OR REPLACE FUNCTION macronutrientPercentageOfMeal(mealId INTEGER)
  RETURNS TABLE(
    proteinPercentage DECIMAL,
    fatPercentage DECIMAL,
    carbPercentage DECIMAL
  ) AS $$
BEGIN
  RETURN QUERY SELECT
    totalProteinGrams / totalMacros * 100 AS proteinPercentage,
    totalFatGrams / totalMacros * 100 AS fatPercentage,
    totalCarbGrams / totalMacros * 100 AS carbPercentage
  FROM BaselineMealNutrition
  WHERE BaselineMealNutrition.mid = mealId;
END;
$$ LANGUAGE 'plpgsql';

SELECT * FROM macronutrientPercentageOfMeal(2);
```

proteinpercentage	fatpercentage	carbpercentage
16.243654822335025381	2.538071065989847716	81.218274111675126904

5. Triggers

Making sure all the data's where it should be.

check_roster_position

```
-- when you insert an athlete into a roster
-- the position for that athlete must be a valid
-- position for that team's sport.
CREATE TRIGGER check_roster_position
BEFORE INSERT ON Roster FOR EACH ROW
EXECUTE PROCEDURE checkRosterPosition();

CREATE OR REPLACE FUNCTION checkRosterPosition()
  RETURNS TRIGGER AS $$
BEGIN
  IF NEW.position IN
         (SELECT positionsName
           FROM SportPositions
           WHERE sportName IN
              (SELECT sport
               FROM Teams
               WHERE tid = NEW.tid)) THEN
    RETURN NEW;
  END IF;
  RAISE EXCEPTION 'Athlete must have a position in correct sport.';

END;
$$ LANGUAGE 'plpgsql';
```

17	26	3	Middy
18	26	3	Lineman

```
[P0001] ERROR: Athlete must have a position in correct sport.
Where: PL/pgSQL function checkrosterposition() line 12 at RAISE
```

Description

Makes sure that the athlete's position is a valid one within the team's sport.

6.

User Roles/Security

Keeping those bad guys away from our data.

User Roles/Security

```sql
-- Roles
CREATE ROLE Admin;
CREATE ROLE Coach;
CREATE ROLE Athlete;
CREATE ROLE KitchenManager;
```

Admin → In charge of the database.

Coach → Deals with the Orders/Deliveries side of the database.

Athlete → Also deals with the Orders/Deliveries side of the database

KitchenManager → Deals with the food, ingredients side of the database.

```sql
-- Admins have a lot of power.
GRANT ALL ON ALL TABLES IN SCHEMA public TO Admin;

-- Revoke all the powers...
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM Coach;
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM Athlete;
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM KitchenManager;

-- Coaches deal only with these tables
GRANT SELECT, INSERT, UPDATE ON Teams, Orders, Deliveries, Selections,
People, Users, Athletes, Coaches, Roster, Contacts
TO Coach;

-- shouldn't be able to change any of the meals.
GRANT SELECT ON Meals, IngredientsInMeals, Ingredients,
CustomMealIngredients, Accommodations, AccommodativeMeals
TO Coach;

-- Update their selections.
GRANT SELECT, UPDATE ON Selections TO Athlete;
GRANT SELECT ON Roster, Athletes, Meals, IngredientsInMeals, Ingredients,
CustomMealIngredients, Accommodations, AccommodativeMeals
TO Athlete;

-- Kitchen managers have all the power to change meal-related things.
GRANT ALL ON EatTime, MealTimeframes, Meals, Accommodations,
AccommodativeMeals, Ingredients, CustomMealIngredients, Meals,
IngredientsInMeals, Kitchens,KitchenManagers, ManagersOfKitchens
TO KitchenManager;

-- But they can only see the stuff that Athletes and Coaches can change.
GRANT SELECT ON Deliveries, Contacts, Selections, Athletes, People
TO KitchenManager;
```

7. Problems/Future Enhancements

Alas, even DB designers are flawed. But we shall keep improving.

Problems/Future Enhancements

Monetary Tracking

The database does not keep track of what meals cost; it only has a reference to the Stripe charge ID, which has the total stored somewhere with Stripe. Stripe handles all the transaction detail, which frankly I'm thankful for.

Inventory Tracking

There is no **quantities** column for the Ingredients table because tracking inventory is out of the scope of the database. It's complicated enough as it is.

Duplicate Phone Columns

It seemed silly to have a single table of "Phonable" people. Also, some of the phone number columns are nullable, which wouldn't really work for the phoneable column

CustomMealIngredients

My best attempt at normalizing the database to only include variable ingredients per Selection. Requires setting nulls in the IngredientsInMeals table which makes me uncomfortable, but if null stands for "unknown" then I should be confident that I made a good design decision