The Agile Family

A DATABASE EXPERIMENT

IST 659 DATA ADMINISTRATION CONCEPTS AND DATABASE MANAGEMENT

SAMPLE PROJECT OUTPUT

This is an example of a LARGE project. Yours may not be as involved (between 5 and 40 pages, depending on your problem domain). Use this to ensure you have all the sections covered and as representative of excellent work. Some portions have been redacted for the purposes of anonymizing the deliverable.

CONTENTS

THE AGILE FAMILY

A DATABASE EXPERIMENT

SUMMARY

PROJECT DESCRIPTION

BUSINESS CASE

While there are tools like Trello which can be used to create a family "sprint", the concept of agile methodology is largely contextualized by the software development world. Taking agile out of its traditional context is the premise of this project. For all intents and purposes, this is a POC (proof of concept) for managing the data and information generated by an experiment of this nature.

THE MARKETING PITCH

Life is busy. It is dynamic, complex, and downright messy at times. As a professional woman in tech, a mother of two young children, a life-partner, a parental caregiver, and a full-time grad student, managing time, tasks, relationships, projects, and information with greater efficiency and less tears is always a high priority.

Meet the Agile Family. An information management solution for the family.

THE DATA MODEL

ENTITIES

BACKLOG

At its most fundamental level, a backlog is a big list of all the work that needs to be accomplished for a project to be considered complete.

In Agile Family terms:

A Backlog is defined as a collection of items to be accomplished.

A backlog is not timeboxed.

USER

An Agile Family end-user is defined as an actor who logs into the system to create or manipulate data. A user supplies some basic contact information and defines an alias for herself such as "Wonder Woman" or "Queen of the Universe", or "Mom". The reason for an alias and a

username relates directly to the user experience. It simply adds an element of customization, flexibility, and fun.

The user role is also user-defined and functions as a way determine responsibility. Roles may change in that they are not permanently assigned by the system. The user has the ability to change a role definition as it relates to a project or sprint.

In Agile Family terms:

A user is an actor who manipulates data.

A user is an actor who creates data.

REPORTER AND ~~ASSIGNEE~~

The ~~Reporter~~ and User entities are used to assign multiple users to multiple items without creating dependencies within a relation.

The REPORTER entity has been removed.

In Agile Family terms:

~~The reporter is a user who has created an item.~~

The assignee is a user to whom the item is assigned.

SPRINT

A sprint is the timeframe in which items pulled from the backlog are assigned to users for completion.

In the figure, the shaded days represent a sprint.

In Agile Family terms:

A sprint is defined as the timeframe in which items should be completed.

A sprint is defined as work to be done in the present.

ITEM

When agile methodology is implemented in a software development context, an item may be a user story, task, bug, or some other user defined term. In this context, the system will provide suggestions for defining an Item Type attribute, but will also allow the user to supply the text for this field.

All Item attribute fields that are metadata are user supplied. The additional Item attributes are relational data points that point to other entities.

In Agile Family terms:

An item is defined as a trackable data point.

An item encapsulates its own metadata.

STATUS

The status of an item is a user defined term used to indicate progress (or even a lack thereof).

In Agile Family terms:

A status describes the state of an item.

PROJECT

For Agile Family users, the project entity is the starting point. It is user-defined, and will be specific or broad depending on the user's preference. Maybe it's a family event like Thanksgiving or process that spans months and weeks like moving. Either way, a user has the ability to supply the project attribute name so that it is meaningful for their purpose.

The project type attribute is another user supplied value. In this field, a user may enter terms like "Outdoor", "Holiday", "Dad", "Basement", etc.

In Agile Family terms:

A project is the highest hierarchal entity.

EPIC

In software development, we create epics when a component or application meets the following criteria:

- It has a broad scope but is not an independent product.
- It can be broken down into smaller units of work like user stories and/or tasks.
- It will take more than one sprint to complete.

Likewise, in the Agile Family system, a user may have a project called "Moving". Due to the scope and low-level details that need managed, it makes sense to create an epic called "Packing" and another called "Logistics". In the framework of the "Packing" epic, a user may create backlog items based on rooms and/or *stuff* within a room or belonging to a specific family member. On the other hand, the "Logistics" epic is what the user would associate with item tasks like "truck rental", "update utilities", and "change address".

In Agile Family terms:

Conceptually, an epic is a higher hierarchical entity than an item.

COMMENT

The comment feature is Agile Family functionality that enables users to interact about a specific backlog item. Much like comments on a social media platform, this is how and where users provide feedback, communicate impediments, or even store notes relevant to item completion.

In Agile Family terms:

A comment is extraneous, user-supplied information about an item.

ATTRIBUTES

An ID attribute is a system generated key.

An ID attribute is unique.

An entity description is not a composite attribute.

RELATIONSHIPS

A **project** may have more than one **backlog**.

A **backlog** must have a **project**.

A **project** may have more than one **sprint**.

A **sprint** must have a **project**.

A **backlog** may have many **items**.

An **item** must have a **backlog**.

A **sprint** may have many **items**.

An **item** may have many **sprints**.

An **item** may have many **statuses**.

A **status** must have an **item**, and is dependent on an item.

An **item** may have many **comments**.

A **comment** must have an **item**, and is dependent on an item.

A **comment** must have a **user**.

A **user** may have many **comments**.

An **item** may have many **labels**.

A **label** may have many **items**.

~~A **user** is a **reporter**.~~

A **user** is an **assignee**.

An **item** is reported by a **user**.

An **item** is assigned to an **assignee**.

STAKEHOLDERS

At this point, the stakeholders are limited to the professor and student. If the student were to attempt to bring this concept to life, the stakeholders would expand to business partners, investors, and potential end-users.

EXPECTATIONS AND OUTCOMES

The outcome of this project should answer the following questions about the student's data:

1. As a user, how can I create and update items, and then assign those items to a backlog?
    a. There is a stored procedure to create items. Please see page 31 for more details.

```
exec spCreateItem
@Item_Type = 'DB Project Task',
@Item_Name = 'Create Data',
@Item_Description = 'DB Table Population',
@Item_Priority = '5',
@Status = 'In Progress',
@Label = 'IST659',
@Backlog_ID = '2',
@Sprint_ID = '4',
@Epic_ID = '1',
@Label_ID = 1,
@Related_Item_ID = '',
@Assignee_ID = '8';
```

    b. At present, we will update items through a simple update statement using the ITEM_ID as the qualifier in the where clause.
    c. Items may be assigned to a backlog at the time of item creation if the backlog exists. Otherwise, the item will be linked to a backlog through an update statement.
2. As a user, how can I see the items in a backlog, and then move those items to a sprint?
    a. Using the following select statement, a user may view items by Backlog_ID:

```
SELECT *
FROM ITEM
WHERE Backlog_ID = 2;
```

100 %

Results | Messages

	Item_ID	Item_Type	Item_Name	Item_Description	Item_Priority	Status	Label	Backlog_ID	Sprint_ID	Epic_ID	Label_ID	Realated_Item_ID	Assignee_ID
1	12	DB Project Task	Create Data	DB Table Population	5	In Progress	IST659	2	4	1	1	0	8
2	13	DB Project Task	Create User Views	Bring together disparate data points for user	4	In Progress	IST 659	2	4	1	1	0	8
3	14	Info Policy Project Task	Merge Partner Contributions	I need to combine sections written my partner a...	5	In Progress	IST 618	2	4	1	2	0	8

    b. Items may be assigned to a sprint at the time of item creation if the sprint exists. Otherwise, the item will be linked to a sprit through an update statement.

3. As a user, how do I manage my user information?
    a. User information would be managed primarily through a UI form. On the backend, a stored procedure has been created to both create and update user information. Please see page 24 for more information

```
EXEC spCreateUpdateUser
@User_Alias = 'Jane Eyre',
@User_Role = 'Governess',
@User_Mobile_Phone = '555-555-2222',
@User_Email_Address = 'jane@thornfieldhall.net'
;
GO
```

4. As the system, how do I provide access to contextualized information derived from data points across entities?
    a. At present, the Agile Family system contains three views to contextualize data points into information that is useful to a user working with in the system:
        i. View: Backlog & Project Tables (Page 35)
        ii. View: User and Assignee Tables (Page 35)
        iii. View: Unique User Views (Page 36)
5. As the system, how do I maintain historical data?
    a. As a concept, the archiving and maintenance of historical data is still in development. This question relates to the concept of "closing out a sprint," which re-occurs on a consistent basis in agile software development. While item-related data should remain accessible to the user, we need to answer questions related to storage, diskspace, and automated archiving. We need to define "historical." Additionally, there are discussion points around our data's life cycle and how we can offer an analytics application that would provide insight into historical data.

REFLECTION

In the spirit of agile methodology, this section will reflect the end of sprint ceremonies known respectively as the sprint review and sprint retrospective. In my experience, the sprint review is focused on the product while retrospective is focused on the team and how the team has managed to build (or failed to build) parts of the product within the sprint.

REVIEW

From a high-level perspective, approaching this project as a POC (proof of concept) and/or experiment has been helpful. At a minimum, it gives me some solace in the following. At present, we have a data model that is a few steps away from exhibiting minimum viability. I think it needs to iterate through a couple more development cycles, but it's a good start. For the purposes of IST659, I think what we have here as a body of work is probably satisfactory. However, as a student and IT professional, I wish I had a better sense of the following:

- The cohesion, or lack thereof, of my data model – My time on this projected has been a bit disjointed. While I am confident I've checked off the action items laid out in the

project description document, I don't have a strong sense that I've brought my data points, data model, and UI ideas together in one, cohesive design.
- Moving fluidly between granular detail to high-level perspective – this is something I tend to struggle with when a concept or idea is still developing in my mind. While this project may have reached the end of its lifespan for IST659, I can see many opportunities to improve it.
- The relationship between the UI and the data – I think more wireframes of a UI would help here.

RETROSPECTIVE

PLUS

This section describes positives, things I like about the project.

- Learning to write stored procedures – I probably did this in undergrad, but don't remember.
- The deliverable is fairly comprehensive as documentation of the process and final product.
- I think there is a good chance the data model is close to minimum viability as prototype.

MINUS

This section describes negatives, things I wish would have worked out differently.

- I would have liked to write more advanced stored procedures, learned to write triggers, and gained a better understanding or database programming. I had ideas that I didn't know how to bring to fruition through SQL.
- I wish I would have had more time to put into wireframing a UI.
- I would have liked to reconstruct the conceptual and logical models for the purpose of iterating one cycle closer to a prototype.

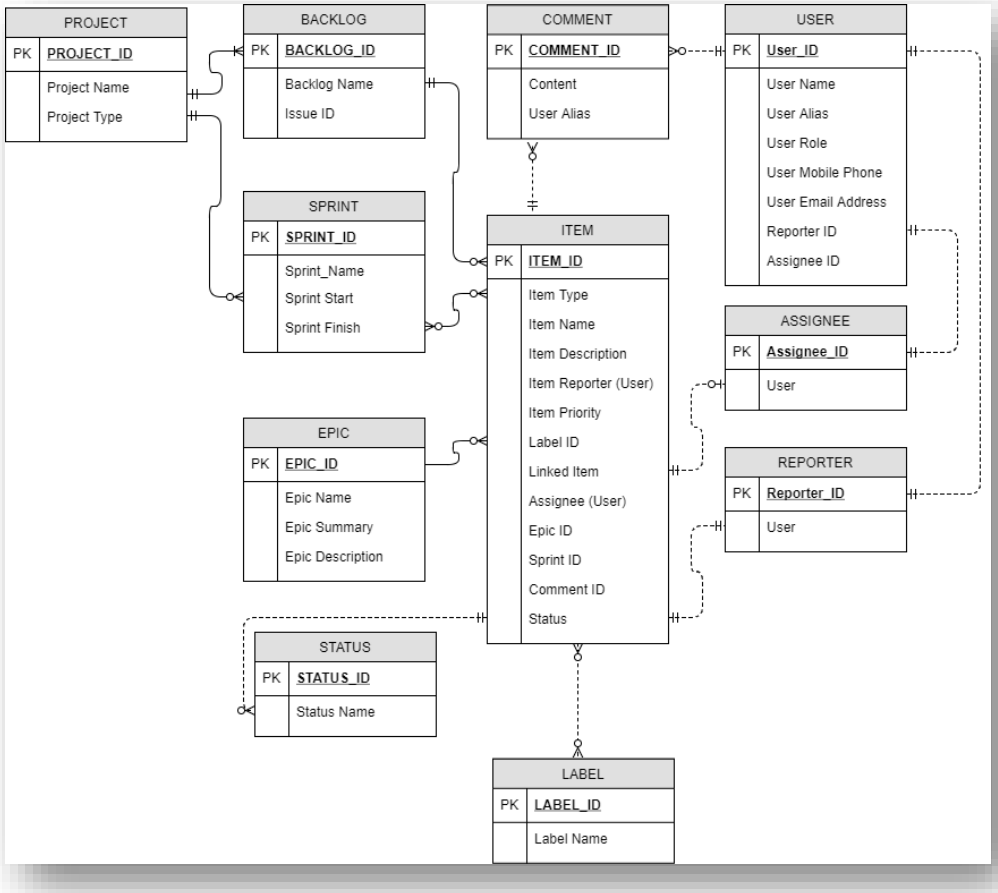
INTERESTING

This section is for observations, thoughts, and ideas that have yet to be vetted.

- If I had this to do again, I would have created and maintained a change log. I feel pretty disconnected from where the data is now compared to what I thought it would be based on conceptual and logical modeling. While this is all captured in the code, I don't have a concise summary of the changes.

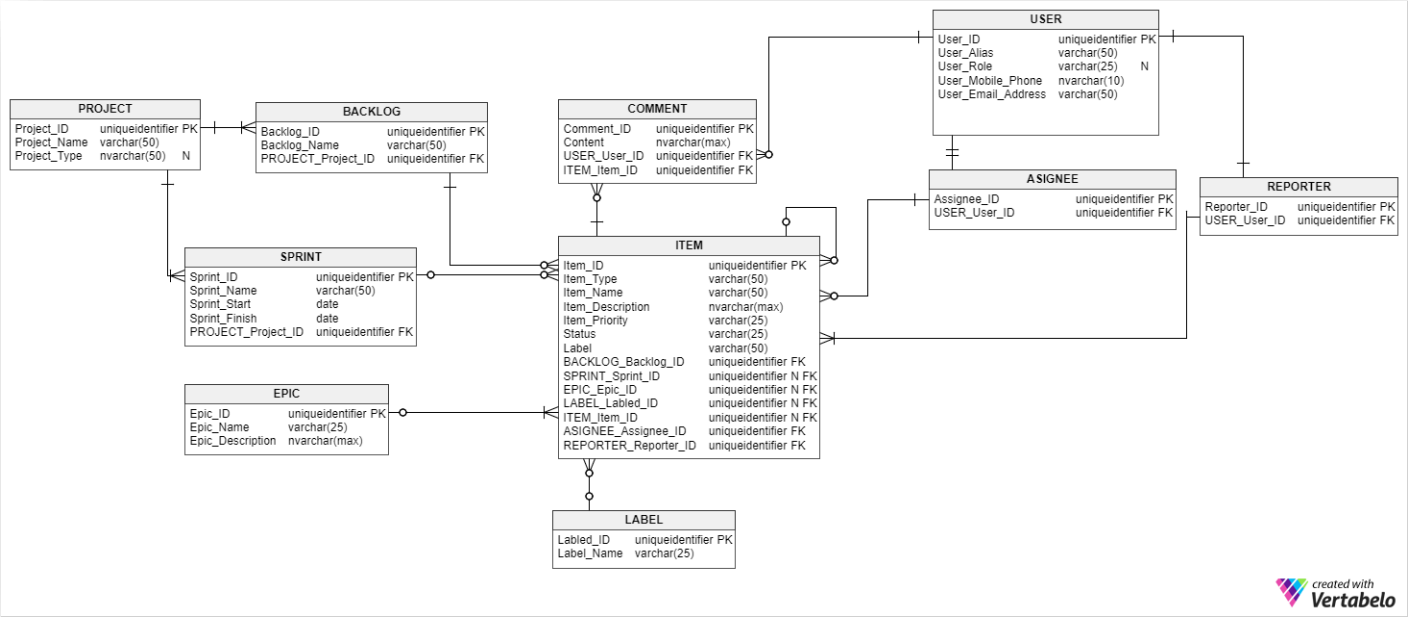
CONCEPTUAL MODEL

Created using Draw.IO



NORMALIZED LOGICAL MODEL

Created using Vertabelo



TABLES

TABLE PROJECT

COLUMNS

Column name	Type	Properties	Description
Project_ID	uniqueidentifier	PK	
Project_Name	varchar(50)		
Project_Type	nvarchar(50)	null	

TABLE BACKLOG

COLUMNS

Column name	Type	Properties	Description
Backlog_ID	uniqueidentifier	PK	
Backlog_Name	varchar(50)		
PROJECT_Project_ID	uniqueidentifier		

TABLE SPRINT

COLUMNS

Column name	Type	Properties	Description
Sprint_ID	uniqueidentifier	PK	
Sprint_Name	varchar(50)		
Sprint_Start	date		
Sprint_Finish	date		

PROJECT_Project_ID	uniqueidentifier		

TABLE COMMENT

COLUMNS

Column name	Type	Properties	Description
Comment_ID	uniqueidentifier	PK	
Content	nvarchar(max)		
USER_User_ID	uniqueidentifier		
ITEM_Item_ID	uniqueidentifier		

TABLE USER

COLUMNS

Column name	Type	Properties	Description
User_ID	uniqueidentifier	PK	
User_Alias	varchar(50)		
User_Role	varchar(25)	null	
User_Mobile_Phone	nvarchar(10)		
User_Email_Address	varchar(50)		

TABLE ITEM

COLUMNS

Column name	Type	Properties	Description
Item_ID	uniqueidentifier	PK	
Item_Type	varchar(50)		
Item_Name	varchar(50)		
Item_Description	nvarchar(max)		
Item_Priority	varchar(25)		
Status	varchar(25)		
Label	varchar(50)		
BACKLOG_Backlog_ID	uniqueidentifier		
SPRINT_Sprint_ID	uniqueidentifier	null	
EPIC_Epic_ID	uniqueidentifier	null	
LABEL_Labled_ID	uniqueidentifier	null	
ITEM_Item_ID	uniqueidentifier	null	
ASIGNEE_Assignee_ID	uniqueidentifier		
~~REPORTER_Reporter_ID~~	uniqueidentifier		

TABLE EPIC

COLUMNS

Column name	Type	Properties	Description
Epic_ID	uniqueidentifier	PK	
Epic_Name	varchar(25)		
Epic_Description	nvarchar(max)		

TABLE LABEL

COLUMNS

Column name	Type	Properties	Description
Labled_ID	uniqueidentifier	PK	
Label_Name	varchar(25)		

TABLE ASIGNEE

COLUMNS

Column name	Type	Properties	Description
Assignee_ID	uniqueidentifier	PK	
USER_User_ID	uniqueidentifier		

~~TABLE REPORTER~~

~~COLUMNS~~

~~Column name~~	~~Type~~	~~Properties~~	~~Description~~
~~Reporter_ID~~	~~uniqueidentifier~~	~~PK~~	
~~USER_User_ID~~	~~uniqueidentifier~~		

REFERENCES

REFERENCE BACKLOG_PROJECT

PROJECT	**1..***	**BACKLOG**
Project_ID	<->	PROJECT_Project_ID

REFERENCE SPRINT_PROJECT

PROJECT	**1..***	**SPRINT**
Project_ID	<->	PROJECT_Project_ID

REFERENCE COMMENT_USER

USER	**0..***	**COMMENT**
User_ID	<->	USER_User_ID

REFERENCE ITEM_BACKLOG

BACKLOG	**0..***	**ITEM**
Backlog_ID	<->	BACKLOG_Backlog_ID

REFERENCE ITEM_SPRINT

SPRINT	**0..***	**ITEM**
Sprint_ID	<->	SPRINT_Sprint_ID

REFERENCE ITEM_EPIC

EPIC	**1..***	**ITEM**
Epic_ID	<->	EPIC_Epic_ID

REFERENCE ITEM_LABEL

LABEL	**0..***	**ITEM**
Labled_ID	<->	LABEL_Labled_ID

REFERENCE ASIGNEE_USER

USER	**1..1**	**ASIGNEE**
User_ID	<->	USER_User_ID

REFERENCE COMMENT_ITEM

ITEM	**0..***	**COMMENT**

Item_ID	<->	ITEM_Item_ID

REFERENCE ITEM_ITEM

ITEM	**0..***	**ITEM**
Item_ID	<->	ITEM_Item_ID

REFERENCE ITEM_ASIGNEE

ASIGNEE	**0..***	**ITEM**
Assignee_ID	<->	ASIGNEE_Assignee_ID

PHYSICAL DATABASE DESIGN

DDL CODE: TABLES

```
-- Table: PROJECT
CREATE TABLE PROJECT (
    Project_ID INT IDENTITY(1,1)  NOT NULL,
    Project_Name varchar(50)  NOT NULL,
    Project_Type nvarchar(50)  NULL,
    CONSTRAINT PROJECT_pk PRIMARY KEY  (Project_ID)
);

-- Table: SPRINT
CREATE TABLE SPRINT (
    Sprint_ID INT IDENTITY(1,1)  NOT NULL,
    Sprint_Name varchar(50)  NOT NULL,
    Sprint_Start date  NOT NULL,
    Sprint_Finish date  NOT NULL,
    --PROJECT_Project_ID INT (1,1)  NOT NULL,
    CONSTRAINT SPRINT_pk PRIMARY KEY  (Sprint_ID)
);

/*I wasn't sure about adding FK constraints. They were in the code Vertabelo created, but
I omitted FK columns upon initial table creation. Once I realized I the FK constraint did
not create the column, I had to alter tables to add them. */

ALTER TABLE dbo.SPRINT
ADD Project_ID INT  NOT NULL;

-- Table: BACKLOG
CREATE TABLE BACKLOG (
    Backlog_ID INT IDENTITY(1,1)  NOT NULL,
    Backlog_Name varchar(50)  NOT NULL,
    --PROJECT_Project_ID INT (1,1)  NOT NULL,
    CONSTRAINT BACKLOG_pk PRIMARY KEY  (Backlog_ID)

);

ALTER TABLE dbo.BACKLOG

ADD Project_ID INT NOT NULL;

-- Reference: BACKLOG_PROJECT (table: BACKLOG)
ALTER TABLE BACKLOG ADD CONSTRAINT FK_BACKLOG_PROJECT
    FOREIGN KEY (Project_ID)

    REFERENCES PROJECT(Project_ID);

-- Reference: SPRINT_PROJECT (table: SPRINT)
ALTER TABLE SPRINT ADD CONSTRAINT FK_SPRINT_PROJECT
    FOREIGN KEY (Project_ID)

    REFERENCES PROJECT(Project_ID);

--have any of my tables been created?
SELECT *

FROM AgileFamily.INFORMATION_SCHEMA.TABLES;
```

```sql
-- Table: ITEM
CREATE TABLE ITEM (
    Item_ID INT IDENTITY(1,1)  NOT NULL,
    Item_Type varchar(50)  NOT NULL,
    Item_Name varchar(50)  NOT NULL,
    Item_Description nvarchar(max)  NOT NULL,
    Item_Priority varchar(25)  NOT NULL,
    Item_Status varchar(25)  NOT NULL,
    Label varchar(50)  NOT NULL,
    Backlog_ID INT  NOT NULL,
    Sprint_ID INT NULL,
    Epic_ID INT NULL,
    Lable_ID INT NULL,
    Linked_Item_ID INT NULL,
    ASIGNEE_Assignee_ID INT NOT NULL,
    REPORTER_Reporter_ID INT NOT NULL,
    CONSTRAINT ITEM_pk PRIMARY KEY  (Item_ID)
);

--I don't like the way Assignee_ID is named and I don't need Reporter
sp_rename 'item.ASIGNEE_Assignee_ID', 'Assignee_ID', 'COLUMN';

ALTER TABLE ITEM
DROP COLUMN REPORTER_Reporter_ID;

-- Table: EPIC
CREATE TABLE EPIC (
    Epic_ID INT IDENTITY(1,1)  NOT NULL,
    Epic_Name varchar(25)  NOT NULL,
    Epic_Description nvarchar(max)  NOT NULL,
    CONSTRAINT EPIC_pk PRIMARY KEY  (Epic_ID)
);

-- Table: COMMENT
CREATE TABLE COMMENT (
    Comment_ID INT IDENTITY(1,1)  NOT NULL,
    Content nvarchar(max)  NOT NULL,
    USER_User_ID INT NOT NULL,
    ITEM_Item_ID INT NOT NULL,
    CONSTRAINT COMMENT_pk PRIMARY KEY  (Comment_ID)
);

--Didn't like the way Vertebelo named my columns
exec sp_rename 'COMMENT.USER_User_ID', 'User_ID', 'COLUMN';

exec sp_rename 'COMMENT.ITEM_Item_ID', 'Item_ID', 'COLUMN';

-- Table: LABEL
CREATE TABLE LABEL (
    Labled_ID INT IDENTITY(1,1)  NOT NULL,
    Label_Name varchar(25)  NOT NULL,
    CONSTRAINT LABEL_pk PRIMARY KEY  (Labled_ID)
);
exec sp_rename 'LABEL.Labled_ID', 'Label_ID', 'COLUMN';

-- Table: USER
```

```sql
CREATE TABLE "USER" (
    User_ID INT IDENTITY(1,1)  NOT NULL,
    User_Alias varchar(50)  NOT NULL,
    User_Role varchar(25)  NULL,
    User_Mobile_Phone nvarchar(10)  NOT NULL,
    User_Email_Address varchar(50)  NOT NULL,
    CONSTRAINT USER_pk PRIMARY KEY  (User_ID)
);

-- Table: ASIGNEE
CREATE TABLE ASSIGNEE (
    Assignee_ID INT IDENTITY(1,1)  NOT NULL,
    USER_User_ID INT NOT NULL,
    CONSTRAINT ASIGNEE_pk PRIMARY KEY  (Assignee_ID)
);

exec sp_rename 'ASSIGNEE.USER_User_ID', 'User_ID', 'COLUMN';

--I hope I've been creating stuff...
SELECT *
FROM AgileFamily.INFORMATION_SCHEMA.TABLES;

--What FK constraints have I created already?
SELECT * FROM sys.objects
WHERE type_desc LIKE '%CONSTRAINT'
ORDER by type;
```

CONSTRAINTS

```sql
-- foreign keys
-- Reference: ASIGNEE_USER (table: ASSIGNEE)
ALTER TABLE ASSIGNEE ADD CONSTRAINT ASSIGNEE_USER
    FOREIGN KEY (User_ID)
    REFERENCES "USER" (User_ID);

/*Already created
-- Reference: BACKLOG_PROJECT (table: BACKLOG)
ALTER TABLE BACKLOG ADD CONSTRAINT BACKLOG_PROJECT
    FOREIGN KEY (PROJECT_Project_ID)
    REFERENCES PROJECT (Project_ID);
*/

-- Reference: COMMENT_ITEM (table: COMMENT)
ALTER TABLE COMMENT ADD CONSTRAINT COMMENT_ITEM
    FOREIGN KEY (Item_ID)
    REFERENCES ITEM (Item_ID);

-- Reference: COMMENT_USER (table: COMMENT)
ALTER TABLE COMMENT ADD CONSTRAINT COMMENT_USER
    FOREIGN KEY (User_ID)
    REFERENCES "USER" (User_ID);

-- Reference: ITEM_ASIGNEE (table: ITEM)
ALTER TABLE ITEM ADD CONSTRAINT ITEM_ASSIGNEE
    FOREIGN KEY (Assignee_ID)
    REFERENCES ASSIGNEE (Assignee_ID);
```

```sql
-- Reference: ITEM_BACKLOG (table: ITEM)
ALTER TABLE ITEM ADD CONSTRAINT ITEM_BACKLOG
    FOREIGN KEY (Backlog_ID)
    REFERENCES BACKLOG (Backlog_ID);

-- Reference: ITEM_EPIC (table: ITEM)
ALTER TABLE ITEM ADD CONSTRAINT ITEM_EPIC
    FOREIGN KEY (Epic_ID)
    REFERENCES EPIC (Epic_ID);

/*Not sure what to do with this...
-- Reference: ITEM_ITEM (table: ITEM)
ALTER TABLE ITEM ADD CONSTRAINT ITEM_ITEM
    FOREIGN KEY (ITEM_Item_ID)
    REFERENCES ITEM (Item_ID);
*/

-- Reference: ITEM_LABEL (table: ITEM)
ALTER TABLE ITEM ADD CONSTRAINT ITEM_LABEL
    FOREIGN KEY (Label_ID)
    REFERENCES LABEL (Label_ID);

/*I keep getting the error message:
Msg 1769, Level 16, State 1, Line 243
Foreign key 'ITEM_LABEL' references invalid column 'Label_ID' in referencing table
'ITEM'.
Msg 1750, Level 16, State 0, Line 243
Could not create constraint or index. See previous errors.
*/

ALTER TABLE LABEL
DROP CONSTRAINT LABEL_pk;

ALTER TABLE LABEL
ADD CONSTRAINT LABEL_pk PRIMARY KEY (Label_ID); --not the problem

exec sp_rename 'ITEM.Lable_ID', 'Label_ID', 'COLUMN';

/*I had some typos in my create statements for the tables
LABEL and ITEM. Naturally, when I went to create the FKs,
I was getting errors because the DMBS couldn't find the column(s)
in the table(s).
I did need to rename the PK and column Label_ID for the sake of accuracy,
but I'm not sure I needed to drop and re-create the PK constraint.
I did need to rename the Label_ID column in the ITEM table though.*/


-- Reference: ITEM_SPRINT (table: ITEM)
ALTER TABLE ITEM ADD CONSTRAINT ITEM_SPRINT
    FOREIGN KEY (Sprint_ID)
    REFERENCES SPRINT (Sprint_ID);

/*Already created
-- Reference: SPRINT_PROJECT (table: SPRINT)
ALTER TABLE SPRINT ADD CONSTRAINT SPRINT_PROJECT
    FOREIGN KEY (PROJECT_Project_ID)
    REFERENCES PROJECT (Project_ID);
```

```
*/

GO

--Are all of my tables and constraints in good shape?
SELECT * FROM sys.objects
WHERE type_desc LIKE '%CONSTRAINT'
ORDER by type;

--Turns out I have not consistently named my FKs
GO

-- Return the current Foreign Key constraints.
SELECT name, SCHEMA_NAME(schema_id) AS schema_name, type_desc
FROM sys.objects
WHERE type = 'F';

GO

sp_rename 'dbo.ASSIGNEE_USER' , 'FK_ASSIGNEE_USER';
exec sp_rename 'dbo.COMMENT_ITEM' , 'FK_COMMENT_ITEM';
exec sp_rename 'dbo.COMMENT_USER' , 'FK_COMMENT_USER';
exec sp_rename 'dbo.ITEM_ASSIGNEE' , 'FK_ITEM_ASSIGNEE';
exec sp_rename 'dbo.ITEM_BACKLOG' , 'FK_ITEM_BACKLOG';
exec sp_rename 'dbo.ITEM_EPIC' , 'FK_ITEM_EPIC';
exec sp_rename 'dbo.ITEM_LABEL' , 'FK_ITEM_LABEL';
exec sp_rename 'dbo.ITEM_SPRINT' , 'FK_ITEM_SPRINT';

GO
-- Return the current Foreign Key constraints to confirm updates.
SELECT name, SCHEMA_NAME(schema_id) AS schema_name, type_desc
FROM sys.objects
WHERE type = 'F';
```

100 %

Results | Messages

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc
1	FK_ASSIGNEE_USER	610101214	NULL	1	1333579789	F	FOREIGN_KEY_CONSTRAINT
2	FK_BACKLOG_PROJECT	1013578649	NULL	1	965578478	F	FOREIGN_KEY_CONSTRAINT
3	FK_COMMENT_ITEM	626101271	NULL	1	1237579447	F	FOREIGN_KEY_CONSTRAINT
4	FK_COMMENT_USER	642101328	NULL	1	1237579447	F	FOREIGN_KEY_CONSTRAINT
5	FK_ITEM_ASSIGNEE	98099390	NULL	1	18099105	F	FOREIGN_KEY_CONSTRAINT
6	FK_ITEM_BACKLOG	114099447	NULL	1	18099105	F	FOREIGN_KEY_CONSTRAINT
7	FK_ITEM_EPIC	130099504	NULL	1	18099105	F	FOREIGN_KEY_CONSTRAINT
8	FK_ITEM_ITEM	146099561	NULL	1	18099105	F	FOREIGN_KEY_CONSTRAINT
9	FK_ITEM_LABEL	418100530	NULL	1	18099105	F	FOREIGN_KEY_CONSTRAINT
10	FK_ITEM_SPRINT	658101385	NULL	1	18099105	F	FOREIGN_KEY_CONSTRAINT
11	FK_SPRINT_PROJECT	1029578706	NULL	1	933578364	F	FOREIGN_KEY_CONSTRAINT

DATA CREATION

INSERT, ALTER TABLE, UPDATE, AND DELETE STATEMENTS

```sql
INSERT INTO "USER"
(User_Alias, User_Role, User_Mobile_Phone, User_Email_Address)
VALUES
('Loki', 'Mischief Maker', '(777)-777-7777', 'loki@donttrustme.com'),
('Odin', 'Allfather', '(444)-444-4444', 'odin@wednesdaysinasgard.com')

SELECT *
FROM "USER"; --am I actually inserting data?

--I kept getting error msg 8152 level 16 state 4 which I determined was an issue
--with my nchar length on the User_Mobile_Phone column
ALTER TABLE "USER"
ALTER COLUMN User_Mobile_Phone NCHAR (14);

GO

INSERT INTO "USER"
(User_Alias, User_Role, User_Mobile_Phone, User_Email_Address)
VALUES
('Frigg', 'Aesir Queen', '(123)-456-7777', 'frigg@ipracticeseidr.com'),
('Freya', 'Aesir-Vanir', '(456)-789-4444', 'freya@ipracticeseidr.com'),
('Freyr', 'Aesir-Vanir', '(123)-789-4444', 'freyr@Alfheim.net'),
('Thor', 'Sparkle Fingers', '(123)-789-4444', 'thor@ilikebighammers.net')
;
GO

UPDATE "USER"
SET User_Mobile_Phone = '(123)-333-4567'
WHERE User_ID = 9; --Thor and Freya probably don't have the same digits

GO

/**I need some data truer to life than what is offered by Norse mythology.
Otherwise, I'd spend more time creating a narraitve around Loki's takeover of the Midgard
than I would actually coding the DB and data*/
INSERT INTO "USER"
(User_Alias, User_Role, User_Mobile_Phone, User_Email_Address)
VALUES
('Pat', 'Mom', '(717)-654-7777', 'pqstudent@syr.edu'),
('Tony', 'Dad', '(412)-607-7777', 'daddio@email.com'),
('Kiddo1', DEFAULT, DEFAULT, 'kid1@email.com'),
('Kiddo2', DEFAULT, DEFAULT, 'kid2@email.com')
;

SELECT *
FROM "USER";

GO

SELECT *
FROM PROJECT;

INSERT INTO PROJECT
(Project_Name, Project_Type)
VALUES
('Take Over Midgard', 'Loki'),
```

```
('Grad School at SU', 'Pat'),
('Backyard', 'Outdoor Home'),
('Living Room', 'Indoor Home'),
('Fourth Grade', 'Kiddo1'),
('Potty Trianing', 'Kiddo2')

--I ran my entire script instead of one statement
--Now I have duplicate records
--I should probably some conditions to stored procedures...

DELETE FROM "USER"
WHERE User_ID > 13;

SELECT *
FROM "USER";

SELECT *
FROM PROJECT;

DELETE FROM PROJECT
WHERE Project_ID > 6;

INSERT INTO BACKLOG
(Backlog_Name, Project_ID)
SELECT Project_Type as Backlog_Name, Project_ID
FROM PROJECT;

INSERT INTO ASSIGNEE
(USER_ID)
SELECT USER_ID
FROM "USER";
```

Results | Messages

	Assignee_ID	User_ID
1	1	46
2	2	47
3	3	48
4	4	49
5	5	50
6	6	51
7	7	52
8	8	53
9	9	54
10	10	55
11	11	56
12	12	57

I've accidentally run my entire script several times now. I blame this on my roots in Oracle. Depending on the tool, the user isn't bound to one function key to run the entire script OR one statement. I think it's F8 or F9 in SQL Developer.

```
INSERT INTO EPIC
(Epic_Name, Epic_Description)
VALUES
('First Term', 'Student 1st Term at SU'),
```

```
('Second Term', 'Student 2nd Term at SU'),
('Third Term', 'Student 3rd Term at SU')
GO
```

[Image Omitted for Confidentiality Reasons]

```
INSERT INTO COMMENT
(Content, User_ID, Item_ID)
VALUES
('I think all of my tables are populated now.' , 53, 12),
('I used some straight insert statements, but wrapped SPs around my insert statements.',
53, 12),
('I need to think about this a little bit more.. Maybe a mindmap would help me frame what
I need to do to complete this task', 53, 13),
('I should probably go back and create some update statements and wrap them in SPs', 53,
13),
('We are about done. Just editing and then we plan to submit tomorrow.', 53, 14);
GO
SELECT * FROM COMMENT;
GO
```

Results | Messages

	Comment_ID	Content	User_ID	Item_ID
1	3	I think all of my tables are populated now.	53	12
2	4	I used some straight insert statements, but wrappe...	53	12
3	5	I need to think about this a little bit more.. Maybe a ...	53	13
4	6	I should probably go back and create some update...	53	13
5	7	We are about done. Just editing and then we plan ...	53	14

STORED PROCEDURES, VIEWS, ETC.

SP TO CREATE OR UPDATE A USER

```
GO
--SP to create and/or update a user
--in the interface, the user will need a message warning them of the following:
--If they enter an existing user name, they will over write a current record
CREATE PROCEDURE spCreateUser
       @User_Alias varchar(50),
       @User_Role varchar(25),
       @User_Mobile_Phone nchar(14),
       @User_Email_Address varchar (50)
AS
BEGIN
       DECLARE @ExistingUserAlias varchar(50)
       SELECT @ExistingUserAlias = User_Alias FROM "USER"
       WHERE User_Alias = @User_Alias
IF EXISTS (SELECT * FROM "USER" WHERE User_Alias = @ExistingUserAlias)
       BEGIN
              UPDATE "USER"
              SET User_Alias = @User_Alias, User_Role = @User_Role, User_Mobile_Phone =
@User_Mobile_Phone,
                     User_Email_Address = @User_Email_Address
              WHERE @ExistingUserAlias = @User_Alias
       END
ELSE
```

```
        BEGIN
                INSERT INTO "USER"
                (User_Alias, User_Role, User_Mobile_Phone, User_Email_Address)
                VALUES
                (@User_Alias, @User_Role, @User_Mobile_Phone, @User_Email_Address)
        END
        RETURN @@IDENTITY
END;

GO
--attempt to execute with an existing user
SELECT *
FROM "USER";

EXEC spCreateUser
@User_Alias = 'Kiddo1',
@User_Role = 'Helper',
@User_Mobile_Phone = '555-555-5555',
@User_Email_Address = 'kiddo.girls0883@gmail.com';
```

The execution of my spCreateUser resulted in an update to all rows on the table. So, every row was set to the information in the EXEC statement above. To fix this, I re-ran my INSERT statements from earlier. Then, I ran delete statements like:

```
DELETE FROM "USER"
WHERE User_ID < 13;
```

Dropping spCreateUser to before I start debugging it

```
SELECT name AS procedure_name,
    SCHEMA_NAME(schema_id) AS schema_name,
    type_desc,
    create_date,
    modify_date
FROM sys.procedures;
```

Results | Messages

procedure_name	schema_name	type_desc	create_date	modify_date
spCreateUser	dbo	SQL_STORED_PROCEDURE	2017-12-02 18:54:48.270	2017-12-02 18:54:48.270

```
456
457 SELECT name AS procedure_name,
458     SCHEMA_NAME(schema_id) AS schema_name,
459     type_desc,
460     create_date,
461     modify_date
462 FROM sys.procedures;
463
464 DROP PROCEDURE spCreateUser;
465 GO
```

Results | Messages

procedure_name	schema_name	type_desc	create_date	modify_date

After dropping the procedure, and losing the locally created variable @ExistingUserAlias, I was able to run the procedure as an update and as a create statement.

```
--SP to create and/or update a user
--in the interface, the user will need a message warning them of the following:
--If they enter an existing user name, they will over write a current record
CREATE PROCEDURE spCreateUser
       @User_Alias varchar(50),
       @User_Role varchar(25),
       @User_Mobile_Phone nchar(14),
       @User_Email_Address varchar (50)
AS
BEGIN
IF EXISTS (SELECT * FROM "USER" WHERE User_Alias = @User_Alias) --If  User_Alias on the
User Table = parameter
       BEGIN
              UPDATE "USER"
              SET User_Alias = @User_Alias, User_Role = @User_Role, User_Mobile_Phone =
@User_Mobile_Phone,
                     User_Email_Address = @User_Email_Address
              WHERE User_Alias = @User_Alias
       END
ELSE
       BEGIN
              INSERT INTO "USER"
              (User_Alias, User_Role, User_Mobile_Phone, User_Email_Address)
              VALUES
              (@User_Alias, @User_Role, @User_Mobile_Phone, @User_Email_Address)
       END
       RETURN @@IDENTITY

END;
GO
```

```
--attempt to execute with an existing user
SELECT *
FROM "USER";

EXEC spCreateUser
@User_Alias = 'Kiddo1',
@User_Role = 'Helper',
@User_Mobile_Phone = '555-555-5555',
@User_Email_Address = 'joyner.girls0883@gmail.com'
;
--attempt to execute with a new existing user
GO
EXEC spCreateUser
@User_Alias = 'Eleven',
@User_Role = 'Adoptee',
@User_Mobile_Phone = '555-555-1111',
@User_Email_Address = 'el@hopperscabin.net'
;
GO
```

Results | Messages

User_ID	User_Alias	User_Role	User_Mobile_Phone	User_Email_Address
24	[illegible]	[illegible]	(717)-654-7777	mojoyner@syr.edu
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
26	Kiddo1	Helper	555-555-5555	joyner.girls0883@gmail.com
27	Kiddo2	NULL	NULL	kid2@email.com
28	Frigg	Aesir Queen	(123)-456-7777	frigg@ipracticeseidr.com
29	Freya	Aesir-Vanir	(456)-789-4444	freya@ipracticeseidr.com
30	Freyr	Aesir-Vanir	(123)-789-4444	freyr@Alfheim.net
31	Thor	Sparkle Fingers	(123)-789-4444	thor@likebighammers.net
32	Loki	Mischief Maker	(777)-777-7777	loki@donttrustme.com
33	Odin	Allfather	(444)-444-4444	odin@wednesdaysinasgard.com
34	Eleven	Adoptee	555-555-1111	el@hopperscabin.net

Omitted

```
GO
--I renamed the spCreateUser to spCreateUpdateUser
--verifying it works
EXEC spCreateUpdateUser
@User_Alias = 'Jane Eyre',
@User_Role = 'Governess',
@User_Mobile_Phone = '555-555-2222',
@User_Email_Address = 'jane@thornfieldhall.net'
;
GO
SELECT * FROM "USER";

--I had to log out and log back in to SSMS
```

SP TO CREATE OR UPDATE A PROJECT

```
--SP to create/update a project

SELECT *
FROM PROJECT;

GO

CREATE PROCEDURE spCreateUpdateProject
       @Project_Name varchar(50),
       @Project_Type nvarchar(50)
AS
BEGIN
IF EXISTS (SELECT * FROM PROJECT WHERE Project_Name = @Project_Name) --If  Project_Name
on the User Table = parameter
       BEGIN
```

```
                UPDATE "PROJECT"
                SET Project_Name = @Project_Name, Project_Type = @Project_Type
                WHERE Project_Name = @Project_Name
        END
ELSE
        BEGIN
                INSERT INTO PROJECT
                (Project_Name, Project_Type)
                VALUES
                (@Project_Name, @Project_Type)
        END
        RETURN @@IDENTITY

END;
```

```
EXEC spCreateUpdateProject
 @Project_Name = 'Bedroom Makeovers',
 @Project_Type = 'Indoor Home';

EXEC spCreateUpdateProject
 @Project_Name = 'Take Over Midgard',
 @Project_Type = 'Odin AND Loki';
```

Results | Messages

Project_ID	Project_Name	Project_Type
1	Take Over Midgard	Odin AND Loki
2	Grad School at SU	Monica
3	Backyard	Outdoor Home
4	Living Room	Indoor Home
5	Fourth Grade	Kiddo1
6	Potty Trianing	Kiddo2
13	Bedroom Makeovers	Indoor Home

```
EXEC spCreateUpdateProject
@Project_Name = 'Christmas',
@Project_Type = 'Family';
GO


SELECT * FROM PROJECT;

GO
```

Results | Messages

	Project_ID	Project_Name	Project_Type
1	1	Take Over Midgard	Odin AND Loki
3	3	Backyard	Outdoor Home
4	4	Living Room	Indoor Home
5	5	Fourth Grade	Kiddo1
6	6	Potty Trianing	Kiddo2
7	13	Bedroom Makeovers	Indoor Home
8	20	Christmas	Family

SP TO CREATE OR UPDATE A BACKLOG

```
--SP to create/update backlog
--This should provide the means to answer the question:
--For which project would you like to create a backlog?
--Before running this, a user will need to obtain the projectID...
SELECT *
FROM BACKLOG;
GO
CREATE PROCEDURE spCreateUpdateBacklog
        @Backlog_Name varchar(50),
```

```sql
	@Project_ID int
AS
BEGIN
IF EXISTS (SELECT * FROM BACKLOG WHERE Project_ID = @Project_ID)
	BEGIN
		UPDATE BACKLOG
		SET Backlog_Name = @Backlog_Name
		WHERE Project_ID = @Project_ID
	END
ELSE
	BEGIN
		INSERT INTO BACKLOG
		(Backlog_Name, Project_ID)
		VALUES
		(@Backlog_Name, @Project_ID)
	END
	RETURN @@IDENTITY
END;
GO

--to create a new Backlog record
EXEC spCreateUpdateBacklog
@Backlog_Name = 'Bedrooms',
@Project_ID = 13
GO
--to update an existing Backlog record
EXEC spCreateUpdateBacklog
@Backlog_Name = '1st Floor Rooms',
@Project_ID = 4
GO
SELECT * FROM viewBacklogProject;
GO
--This inserted a new row instead of updating one...
--Drop and and implement fix
DROP PROCEDURE spCreateUpdateBacklog;
GO
DELETE BACKLOG
WHERE Backlog_Name = '1st Floor Rooms';
GO
--I'm not inserting or updating now
--Dropping and editing procedure again
```

```
550 SELECT *
551 FROM viewBacklogProject; --might alter to pull in projectID from PROJECT table
552 GO
553 --to create a new Backlog record
554 EXEC spCreateUpdateBacklog
555 @Backlog_Name = 'Bedrooms',
556 @Project_ID = 13
557 GO
558 --to update an existing Backlog record
559 EXEC spCreateUpdateBacklog
560 @Backlog_Name = '1st Floor Rooms',
561 @Project_ID = 4
562 GO
563 --This inserted a new row instead of updating one...
```

Results | Messages

	Backlog_Name	Project_ID	Project_Name	Project_Type
1	Loki	1	Take Over Midgard	Odin AND Loki
3	Outdoor Home	3	Backyard	Outdoor Home
4	1st Floor Rooms	4	Living Room	Indoor Home
5	Kiddo1	5	Fourth Grade	Kiddo1
6	Kiddo2	6	Potty Trianing	Kiddo2
7	Bedrooms	13	Bedroom Makeovers	Indoor Home

SP TO CREATE AN EPIC

```
CREATE PROCEDURE spCreateEpic
       @Epic_Name varchar(25),
       @Epic_Description nvarchar(max)
AS
BEGIN
       INSERT INTO EPIC
       (Epic_Name, Epic_Description)
       VALUES
       (@Epic_Name, @Epic_Description)
       RETURN @@IDENTITY
END;
GO
exec spCreateEpic
@Epic_Name = 'Fire Pit',
@Epic_Description = 'Like it sounds. We want a fire pit in the backyard'
GO
```

Results | Messages

	Epic_ID	Epic_Name	Epic_Description
1	1	First Term	1st Term at SU
2	2	Second Term	2nd Term at SU
3	3	Third Term	3rd Term at SU
4	4	Fire Pit	Like it sounds. We want a fire pit in the backyard

SP TO CREATE OR UPDATE A SPRINT

```
CREATE PROCEDURE spCreateUpdateSprint
       @Sprint_Name varchar(50),
       @Sprint_Start date,
       @Sprint_Finish date,
       @Project_ID int
AS
BEGIN
IF EXISTS (SELECT * FROM SPRINT WHERE Sprint_Name = @Sprint_Name)
       BEGIN
              UPDATE SPRINT
              SET Sprint_Start = @Sprint_Start, Sprint_Finish = @Sprint_Finish,
Project_ID = @Project_ID
              WHERE Sprint_Name = @Sprint_Name
```

```
        END
ELSE
        BEGIN
                INSERT INTO SPRINT
                (Sprint_Name, Sprint_Start, Sprint_Finish, Project_ID)
                VALUES
                (@Sprint_Name, @Sprint_Start, @Sprint_Finish, @Project_ID)
        END
        RETURN @@IDENTITY
END;
GO

EXEC spCreateUpdateSprint
@Sprint_Name = 'Sprint 1 - Grad School',
@Sprint_Start = '2017-10-05',
@Sprint_Finish = '2017-10-18',
@Project_ID = 2;

EXEC spCreateUpdateSprint
@Sprint_Name = 'Sprint 2 - Grad School',
@Sprint_Start = '2017-10-19',
@Sprint_Finish = '2017-11-01',
@Project_ID = 2;

EXEC spCreateUpdateSprint
@Sprint_Name = 'Sprint 3 - Grad School',
@Sprint_Start = '2017-11-02',
@Sprint_Finish = '2017-11-15',
@Project_ID = 2;

EXEC spCreateUpdateSprint
@Sprint_Name = 'Sprint 4 - Grad School',
@Sprint_Start = '2017-11-16',
@Sprint_Finish = '2017-11-29',
@Project_ID = 2;

EXEC spCreateUpdateSprint
@Sprint_Name = 'Sprint 5 - Grad School',
@Sprint_Start = '2017-11-30',
@Sprint_Finish = '2017-12-14',
@Project_ID = 2;

GO
```

100 %

Results | Messages

	Sprint_ID	Sprint_Name	Sprint_Start	Sprint_Finish	Project_ID
1	1	Sprint 1 - Grad School	2017-10-05	2017-10-18	2
2	2	Sprint 2 - Grad School	2017-10-19	2017-11-01	2
3	3	Sprint 3 - Grad School	2017-11-02	2017-11-15	2
4	4	Sprint 4 - Grad School	2017-11-16	2017-11-29	2
5	5	Sprint 5 - Grad School	2017-11-30	2017-12-14	2

SP TO CREATE AN ITEM

```
-- sp to create an item
CREATE PROCEDURE spCreateItem
```

```
        @Item_Type varchar(50),
        @Item_Name varchar(50),
        @Item_Description nvarchar(max),
        @Item_Priority varchar(25),
        @Status varchar(25),
        @Label varchar(50),
        @Backlog_ID int,
        @Sprint_ID int,
        @Epic_ID int,
        @Label_ID int,
        @Related_Item_ID int,
        @Assignee_ID int
AS
BEGIN
        INSERT INTO ITEM (
                Item_Type,
                Item_Name,
                Item_Description,
                Item_Priority,
                "Status",
                Label,
                Backlog_ID,
                Sprint_ID,
                Epic_ID,
                Label_ID,
                Realated_Item_ID,
                Assignee_ID
                )
        VALUES(
                @Item_Type,
                @Item_Name,
                @Item_Description,
                @Item_Priority,
                @Status,
                @Label,
                @Backlog_ID,
                @Sprint_ID,
                @Epic_ID,
                @Label_ID,
                @Related_Item_ID,
                @Assignee_ID
                )
        RETURN @@IDENTITY
END;
GO

exec spCreateItem
@Item_Type = 'DB Project Task',
@Item_Name = 'Create Data',
@Item_Description = 'DB Table Population',
@Item_Priority = '5',
@Status = 'In Progress',
@Label = 'IST659',
@Backlog_ID = '2',
@Sprint_ID = '4',
@Epic_ID = '1',
@Label_ID = 1,
@Related_Item_ID = '',
```

```
@Assignee_ID = '8';

exec spCreateItem
@Item_Type = 'DB Project Task',
@Item_Name = 'Create User Views',
@Item_Description = 'Bring together disparate data points for user',
@Item_Priority = '4',
@Status = 'In Progress',
@Label = 'IST 659',
@Backlog_ID = '2',
@Sprint_ID = '4',
@Epic_ID = '1',
@Label_ID = 1,
@Related_Item_ID = '',
@Assignee_ID = '8';

exec spCreateItem
@Item_Type = 'Info Policy Project Task',
@Item_Name = 'Merge Partner Contributions',
@Item_Description = 'I need to combine sections written my partner and I respectively',
@Item_Priority = '5',
@Status = 'In Progress',
@Label = 'IST 618',
@Backlog_ID = '2',
@Sprint_ID = '4',
@Epic_ID = '1',
@Label_ID = 2,
@Related_Item_ID = '',
@Assignee_ID = '8';

exec spCreateItem
@Item_Type = 'Fire Pit Prep Task',
@Item_Name = 'Remove existing materials from grotto',
@Item_Description = 'o build a fire pit, we must first remove the exisitng structure.',
@Item_Priority = '2',
@Status = 'In Progress',
@Label = 'Fire Pit',
@Backlog_ID = '3',
@Sprint_ID = 3,
@Epic_ID = '4',
@Label_ID = 3,
@Related_Item_ID = '',
@Assignee_ID = '7';

GO
```

100 %

Results | Messages

	Item_ID	Item_Type	Item_Name	Item_Description	Item_Priority	Status	Label	Backlog_ID	Sprint_ID	Epic_ID	Label_ID	Realated_Item_ID	Assignee_ID
1	12	DB Project Task	Create Data	DB Table Population	5	In Progress	IST659	2	4	1	1	0	8
2	13	DB Project Task	Create User Views	Bring together disparate data points for user	4	In Progress	IST 659	2	4	1	1	0	8
3	14	Info Policy Project Task	Merge Partner Contributions	I need to combine sections written my partner a...	5	In Progress	IST 618	2	4	1	2	0	8
4	18	Fire Pit Prep Task	Remove existing materials from grotto	o build a fire pit, we must first remove the exisitn...	2	In Progress	Fire Pit	3	3	4	3	0	7

SP TO CREATE AN ASSIGNEE

```
/*We should not use this... ever
CREATE PROCEDURE spCreateAssignee
```

```
	@User_ID int
AS
BEGIN
	INSERT INTO ASSIGNEE
	(User_ID)
	VALUES(@User_ID)
	RETURN @@IDENTITY
END;
GO
*/
```

I created this SP in my frenzy to have a stored procedure for every table. That said, I think populating the ASSIGNEE table should probably be kicked off by a trigger. That is, upon the creation of a user, the ASSIGNEE table should be updated with the new User_ID.

SP TO CREATE A COMMENT

```
--SP to add a comment
CREATE PROCEDURE spCreateComment
	@Content nvarchar(max),
	@User_ID int,
	@Item_ID int
AS
BEGIN
	INSERT INTO COMMENT
	(Content, User_ID, Item_ID)
	VALUES(@Content, @User_ID, @Item_ID)
	RETURN @@IDENTITY
END;
GO

EXEC spCreateComment
@Content = 'I think we need a view for comments and items...',
@User_ID = 53,
@Item_ID = 13
;

EXEC spCreateComment
@Content = 'To create the view Chad requested, I think we do a three table join -- item,
comment, user on... Actually, it will be 4 tables. Item, Comment, User, and Assignee.',
@User_ID = 53,
@Item_ID = 13
;
GO
```

Results | Messages

	Comment_ID	Content	User_ID	Item_ID
1	3	I think all of my tables are populated now.	53	12
2	4	I used some straight insert statements, but wrapped SPs around my insert statements.	53	12
3	5	I need to think about this a little bit more.. Maybe a mindmap would help me frame what I need to do to complete this task	53	13
4	6	I should probably go back and create some update satements and wrap them in SPs	53	13
5	7	We are about done. Just editing and then we plan to submit tomorrow.	53	14
6	8	I think we need a view for comments and items...	53	13
7	9	To create the view Chad requested, I think we do a three tabel join -- item, comment, user on... Actually, it will be 4 tables. Item, Comment, User, and Assignee.	53	13

VIEW: BACKLOG & PROJECT TABLES

```
--View to see what's on the Backlog table that's also on the Project Table but not vice
versa
--This is useful information to the user as they create backlogs for their projects
CREATE VIEW viewBacklogProject AS
       SELECT BACKLOG.Backlog_Name, BACKLOG.Project_ID, PROJECT.Project_Name,
PROJECT.Project_Type
       FROM BACKLOG
       RIGHT OUTER JOIN PROJECT
       ON BACKLOG.Project_ID = PROJECT.Project_ID;
GO

SELECT * FROM viewBacklogProject;
GO
```

I need to clean up the tables. We shouldn't have two projects of the same ID and Name. This happened when I was playing with the stored procedure to create either a project or a backlog. Because I initially omitted logic to look for an existing record and update (instead of insert), I ended up with duplicates.

	Backlog_Name	Project_ID	Project_Name	Project_Type
1	Loki	1	Take Over Midgard	Odin AND Loki
2	Odin AND Loki	1	Take Over Midgard	Odin AND Loki
3	[illegible]	2	Grad School at SU	[illegible]
4	Outdoor Home	3	Backyard	Outdoor Home
5	1st Floor Rooms	4	Living Room	Indoor Home
6	Kiddo1	5	Fourth Grade	Kiddo1
7	Kiddo2	6	Potty Trianing	Kiddo2
8	Bedrooms	13	Bedroom Makeovers	Indoor Home
9	Winter	20	Christmas	Family

VIEW: USER USER AND ASSIGNEE TABLES

```
CREATE VIEW viewUserAssignee AS
SELECT ASSIGNEE.Assignee_ID, "USER".User_ID, "USER".User_Alias
FROM ASSIGNEE
JOIN "USER" ON ASSIGNEE.User_ID = "USER".User_ID;
GO

SELECT * FROM viewUserAssignee;
```

Results | Messages

	Assignee_ID	User_ID	User_Alias
1	1	46	Jane Eyre
2	2	47	Loki
3	3	48	Odin
4	4	49	Frigg
5	5	50	Freya
6	6	51	Freyr
7	7	52	Thor
8	8	53	Monica
9	9	54	Tony
10	10	55	Kiddo1
11	11	56	Kiddo2
12	12	57	Eleven

VIEW: UNIQUE USER VIEW

```
--Chad's request
CREATE VIEW viewUserCommentItem AS
```

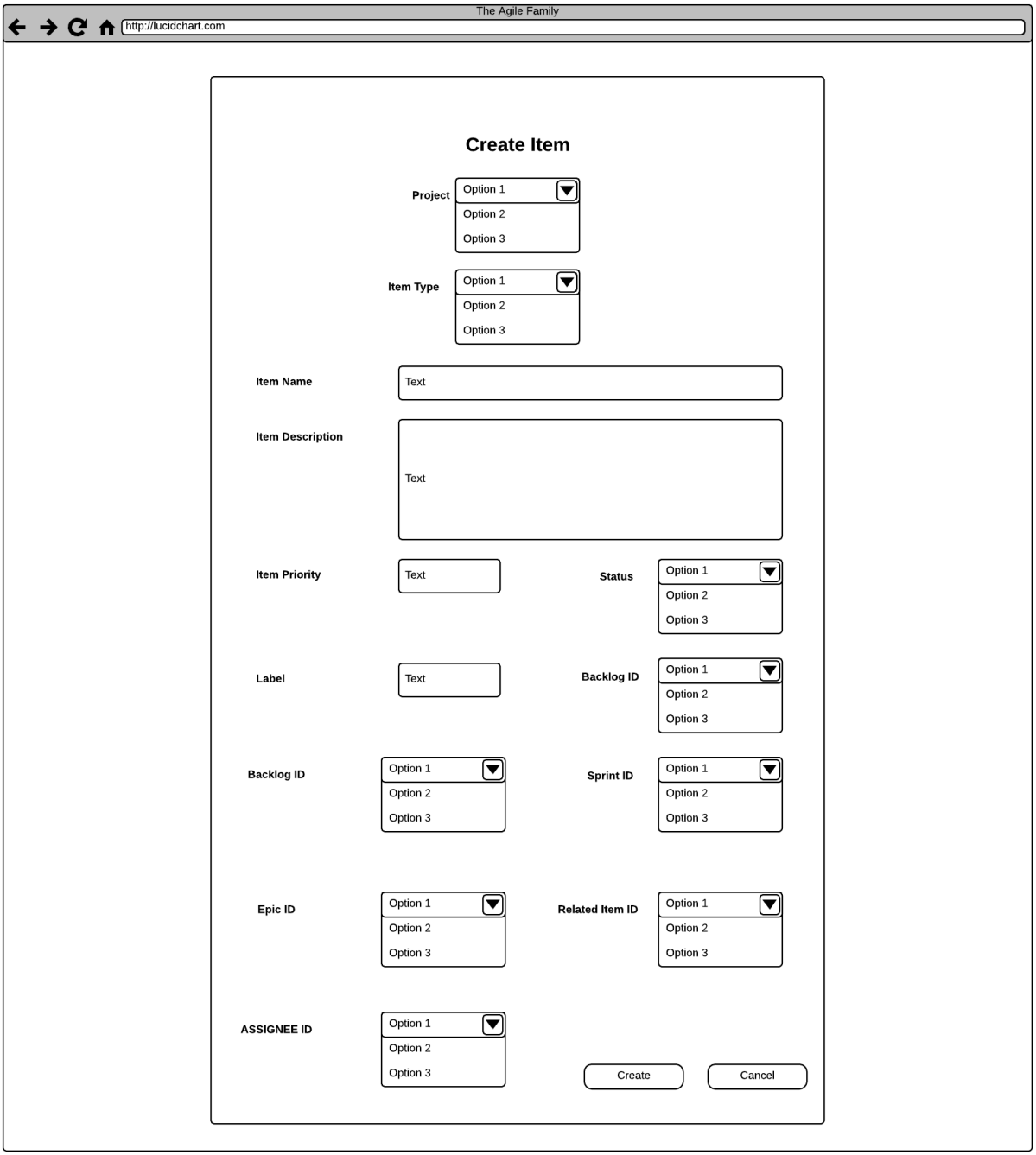
```
        SELECT ITEM.Assignee_ID,
                     ITEM.Item_ID,
                     ITEM.Item_Type,
                     ITEM.Item_Name,
                     ITEM.Item_Description,
                     ITEM.Item_Priority,
                     ITEM."Status",
                     ITEM.Label,
                     ITEM.Backlog_ID,
                     ITEM.Sprint_ID,
                     ITEM.Epic_ID,
                     ITEM.Label_ID,
                     ITEM.Realated_Item_ID,
                     COMMENT.Content,
                     "USER".User_Alias
        FROM   ITEM
        JOIN   COMMENT ON ITEM.Item_ID = COMMENT.Item_ID
        JOIN   "USER" ON COMMENT.User_ID = "USER".User_ID
        WHERE  "USER".User_ID = 53;
GO

SELECT * FROM viewUserCommentItem;
```

[Omitted for Anonymity: Screen captures of the view results]

IMPLEMENTATION

WIREFRAME: CREATE ITEM

Created with Lucidchart



WIREFRAME: UNIQUE USER VIEW

Created with Lucidchart