



Parts Finder Application

Database Management: Project Report

Group Members: JiaHong Yu, Matthew Conway, Josh Doherty,
David Dunlavey, Kenneth Mende

APEX Workspace: LEGGO

Username: VICTORYU963@GMAIL.COM

Password: JYuPnv3n8txh

Chapter 1: Problem Description

One of the world's most famous and constructive toys continues to increase in demand throughout several decades. This trendy toy is the lovable LEGO. LEGO's started production in the 1930's and have reached revenue of over \$37.9 billion at the end of the 2016 fiscal year. Throughout the years, LEGO has developed countless brick styles that go along with millions of styles and themes. Many of the LEGO box designs use the same brick parts to build a different creation. One of the largest problems that a LEGO consumer faces are finding the parts they need to create a new desired set.

Our group has built a database application using Oracle's APEX to fix this problem. Our online database will allow consumers to input the set that they have wished to create, and the application will deliver various amounts of parts that they need to acquire. Our program will use several attributes of the LEGO including the part, color, and sets that the user wishes to create. This information with then is used within our database application to find the parts that the user needs to build a new set. Our site will also include the difficulty of the sets that the user is trying to create.

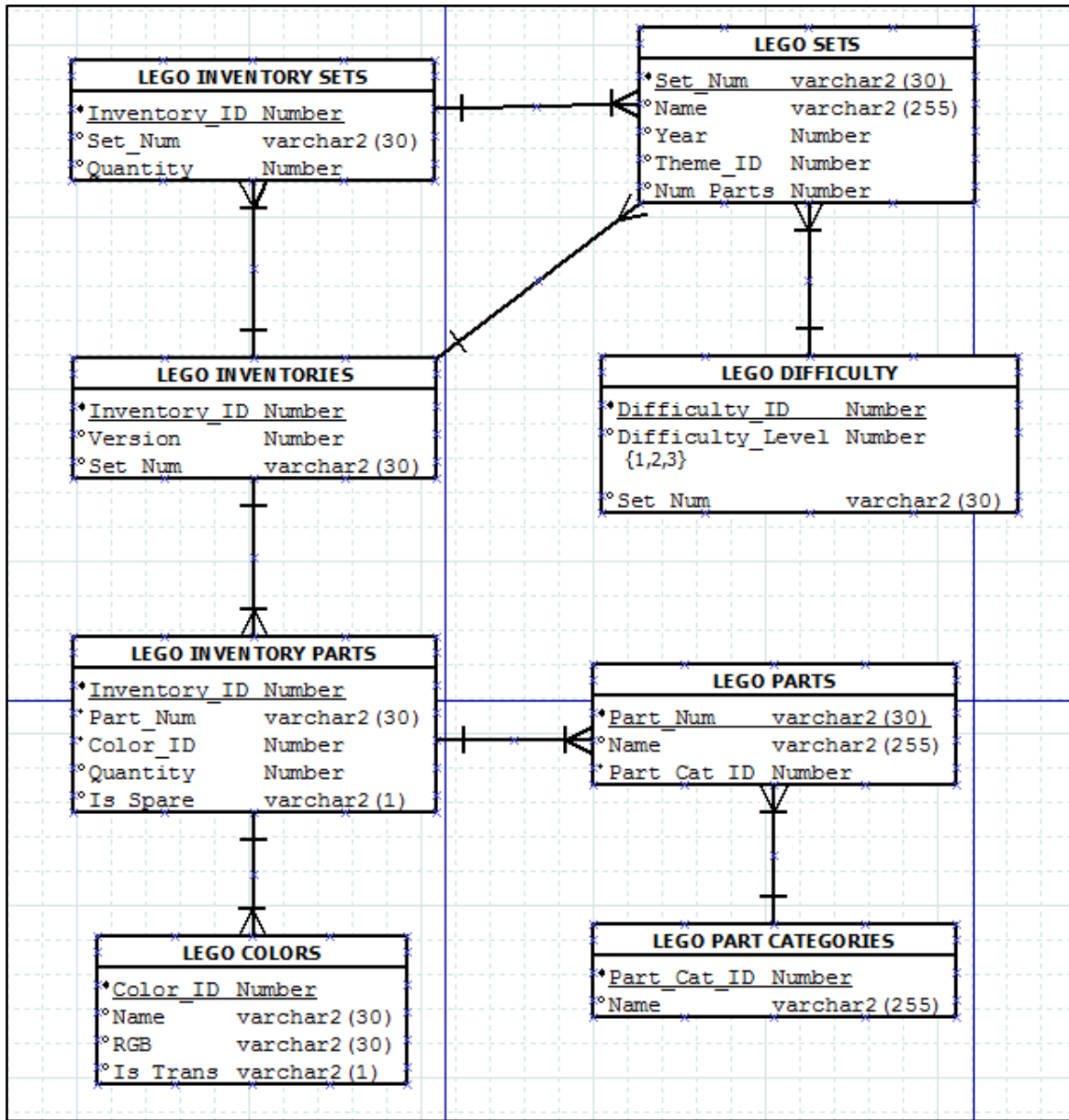
Some of the business rules for our database application include that the consumer must provide a detailed idea of the set in which they wish to create. For the application to launch an output, the prior information must be included. We used real data from Rebrickable.com to develop our database application. From the data that we pulled, we developed an additional table called the "Difficulty table" that gave the user an opportunity to see the difficulty of the set they were pursuing to build.

The goal of our database application is to provide a parts output for any given set that a consumer could use to recreate the set. This goal is only possible if the consumer already has a high count of LEGO inventory that they can rely on to build with.

However, if the user does not have a large LEGO inventory already, our application will pick out the few parts that they need to acquire rather than purchasing a whole new set. We want to help the consumer recycle their current parts while decreasing their financial consumptions of LEGO.

Chapter 2: Conceptual Data Model

_____ Below is the ER-Diagram that we have created for our given data. Each relationship between the tables is significant to the consistency of our queries. We were able to develop a Lego Difficulty table by scoring the number of sets in the Lego Sets table. With the creation of this ER-Diagram, we were able to develop several business rules regarding the data that we used.

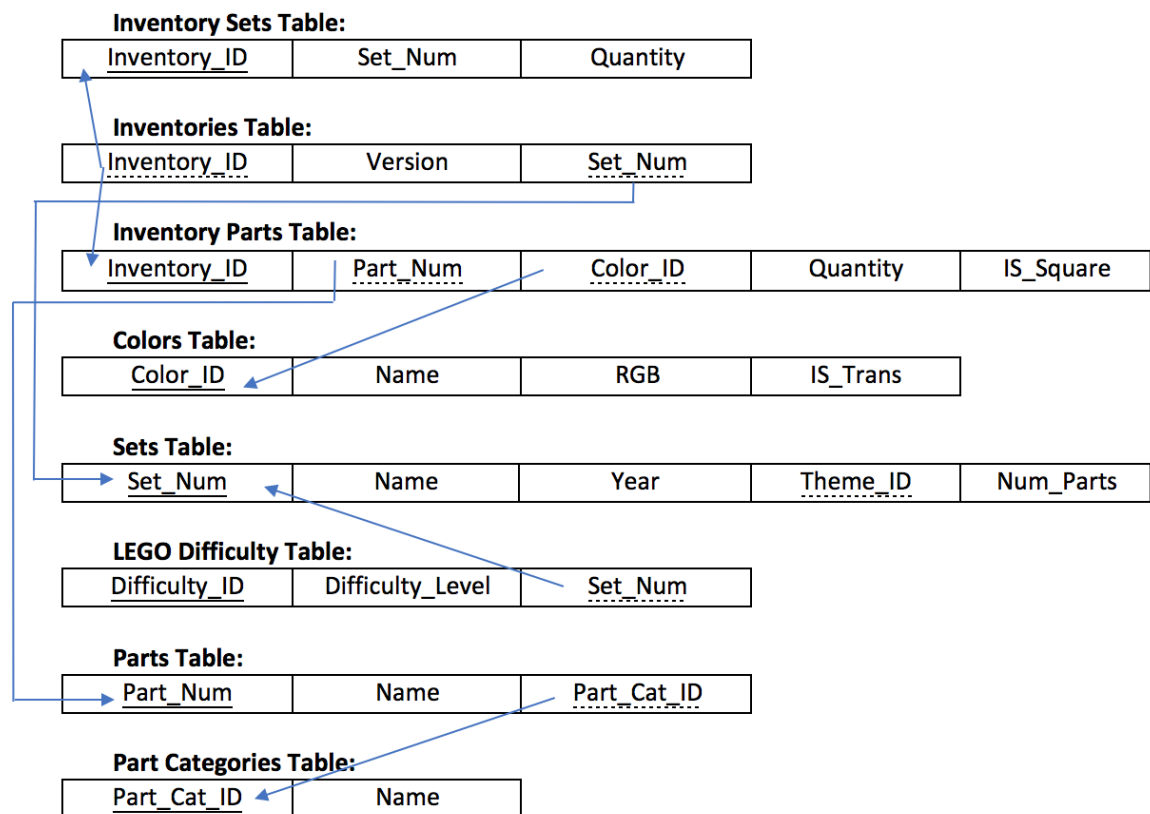


Many of the business rules are visible by reviewing our ER-Diagram. Some of these business rules include that the “Inventory_ID” must be a number and that the “Set_Num” is a text that can have up to thirty characters. One of the most interesting business rules is in regards to the “Difficulty_Level.” This business rule explains that each set can be classified as a 1, 2, or 3 meaning that it is easy, medium, or hard.

Most of the relationships in ER-Diagram are one to many. These are the most logical relationships that can be formed for the given data. For example, the relationship between the LEGO Inventories and LEGO Sets table is a one to many. This means that the user must have one inventory that contains one or more LEGO sets.

Chapter 3: Relational Schema

Below is an image of the relational schema that we have built for our given data:



The relational schema offers a vision of how all of the data categories and tables are connected. The relational schema has been created in third normal form, so none of the attributes in any of the seven tables have a partial relationship nor a transitive relationship. Given all the attributes in the tables above, our team was able to develop

an application that could use the data to help an individual user find the additional LEGO parts that they would need to complete certain set.

One of the most important tables from those listed above is the Sets table. To solve the problem that our application is trying to fix, the user must key into the app the set that they wish to build. Once they have imported the set they're seeking to build; the application will provide all of the parts that are needed to build this set. For example, the user might search a "plane," and the application will give the user an output of parts such as a grey brick, or a windshield part.

Some of the other tables that we have included in our relational schema include an inventory sets, inventories, and inventory parts table. Each of these tables provides different information and attributes about the data. The colors table allows the user of the application to search for specific colors that they want to include in their set. This table refines the search and makes the output more customer oriented by assisting to help customization in their next build. For the plane example above, the colors table would allow the user of the application to choose the color plane that they would like to build (blue, green, etc.). Each table enables the user of the application to have a more specific search.

Chapter 4: Database Implementation

We created a total of nine tables and used the data workshop under the 'Utilities' tag to upload the CSV files we retrieved into Oracle APEX. We set up the constraints of primary and foreign keys based on the schema we drew previously. Our table creation commands are shown below:

CREATE TABLE "LEGO_COLORS"

```
(  "ID" NUMBER,  
    "NAME" VARCHAR2(30),  
    "RGB" VARCHAR2(30),  
    "IS_TRANS" VARCHAR2(1),  
    CONSTRAINT "LEGO_COLORS_PK" PRIMARY KEY ("ID")  
    USING INDEX ENABLE  
)  
/
```

CREATE TABLE "LEGO_DIFFICULTY"

```
(  "SET_NUM" VARCHAR2(30),  
    "ID" NUMBER,  
    "DIFFICULTY_LEVEL" NUMBER,  
    CONSTRAINT "LEGO_DIFFICULTY_PK" PRIMARY KEY ("ID")  
    USING INDEX ENABLE  
)  
/  
ALTER TABLE "LEGO_DIFFICULTY" ADD CONSTRAINT "LEGO_DIFFICULTY_FK"  
FOREIGN KEY ("SET_NUM")  
    REFERENCES "LEGO_SETS" ("SET_NUM") ON DELETE CASCADE ENABLE  
/
```

CREATE TABLE "LEGO_INVENTORIES"

```
(  "ID" NUMBER,  
    "VERSION" NUMBER,  
    "SET_NUM" VARCHAR2(30),  
    CONSTRAINT "LEGO_INVENTORIES_PK" PRIMARY KEY ("ID")  
    USING INDEX ENABLE  
)  
/  
ALTER TABLE "LEGO_INVENTORIES" ADD CONSTRAINT "LEGO_INVENTORIES_CON"  
FOREIGN KEY ("SET_NUM")  
    REFERENCES "LEGO_SETS" ("SET_NUM") ON DELETE CASCADE ENABLE  
/
```

CREATE TABLE "LEGO_INVENTORY_PARTS"

```
(  "INVENTORY_ID" NUMBER NOT NULL ENABLE,  
    "PART_NUM" VARCHAR2(30) NOT NULL ENABLE,  
    "COLOR_ID" NUMBER NOT NULL ENABLE,  
    "QUANTITY" NUMBER,
```

```

        "IS_SPARE" VARCHAR2(1),
        CONSTRAINT "LEGO_INVENTORY_PARTS_CON" PRIMARY KEY
("INVENTORY_ID", "PART_NUM", "COLOR_ID")
        USING INDEX ENABLE
    )
/
ALTER TABLE "LEGO_INVENTORY_PARTS" ADD CONSTRAINT
"LEGO_INVENTORY_PARTS_CON_1" FOREIGN KEY ("INVENTORY_ID")
        REFERENCES "LEGO_INVENTORIES" ("ID") ON DELETE CASCADE ENABLE
/
ALTER TABLE "LEGO_INVENTORY_PARTS" ADD CONSTRAINT
"LEGO_INVENTORY_PARTS_CON_2" FOREIGN KEY ("COLOR_ID")
        REFERENCES "LEGO_COLORS" ("ID") ON DELETE CASCADE ENABLE
/
ALTER TABLE "LEGO_INVENTORY_PARTS" ADD CONSTRAINT
"LEGO_INVENTORY_PARTS_CON_3" FOREIGN KEY ("PART_NUM")
        REFERENCES "LEGO_PARTS" ("PART_NUM") ON DELETE CASCADE ENABLE
/

```

CREATE TABLE "LEGO_INVENTORY_SETS"

```

(
    "INVENTORY_ID" NUMBER,
    "SET_NUM" VARCHAR2(30),
    "QUANTITY" NUMBER,
    CONSTRAINT "LEGO_INVENTORY_SETS_PK" PRIMARY KEY ("INVENTORY_ID",
"SET_NUM")
    USING INDEX ENABLE
)
/
ALTER TABLE "LEGO_INVENTORY_SETS" ADD CONSTRAINT "FOREIGN_KEY_1"
FOREIGN KEY ("INVENTORY_ID")
        REFERENCES "LEGO_INVENTORIES" ("ID") ON DELETE CASCADE ENABLE
/
ALTER TABLE "LEGO_INVENTORY_SETS" ADD CONSTRAINT "FOREIGN_KEY_2"
FOREIGN KEY ("SET_NUM")
        REFERENCES "LEGO_SETS" ("SET_NUM") ON DELETE CASCADE ENABLE
/

```

CREATE TABLE "LEGO_PARTS"

```

(
    "PART_NUM" VARCHAR2(30),
    "NAME" VARCHAR2(255),
    "PART_CAT_ID" NUMBER,
    CONSTRAINT "LEGO_PARTS_PK" PRIMARY KEY ("PART_NUM")

```



```

        USING INDEX ENABLE
    )
/
ALTER TABLE "LEGO_PARTS" ADD CONSTRAINT "LEGO_PARTS_CON" FOREIGN KEY
("PART_CAT_ID")
        REFERENCES "LEGO_PART_CATEGORIES" ("ID") ON DELETE CASCADE
ENABLE
/

```

```

CREATE TABLE "LEGO_PART_CATEGORIES"
(
    "ID" NUMBER,
    "NAME" VARCHAR2(255),
    CONSTRAINT "LEGO_PART_CATEGORIES_PK" PRIMARY KEY ("ID")
    USING INDEX ENABLE
)
/

```

```

CREATE TABLE "LEGO_SETS"
(
    "SET_NUM" VARCHAR2(30),
    "NAME" VARCHAR2(255),
    "YEAR" NUMBER,
    "THEME_ID" NUMBER,
    "NUM_PARTS" NUMBER,
    CONSTRAINT "LEGO_SETS_PK" PRIMARY KEY ("SET_NUM")
    USING INDEX ENABLE
)
/
ALTER TABLE "LEGO_SETS" ADD CONSTRAINT "LEGO_SETS_CON" FOREIGN KEY
("THEME_ID")
        REFERENCES "LEGO_THEMES" ("ID") ON DELETE CASCADE ENABLE
/

```

```

CREATE TABLE "LEGO_THEMES"
(
    "ID" NUMBER,
    "NAME" VARCHAR2(30),
    "PARENT_ID" NUMBER,
    CONSTRAINT "LEGO_THEMES_PK" PRIMARY KEY ("ID")
    USING INDEX ENABLE
)
/

```

Note: we created the 'LEGO_DIFFICULTY' table on our own to better understand the usage of Lego sets. We created the difficulty level using the following calculations: We decided to label the difficulty using 1 for easy sets, 2 for medium and 3 for hard sets. Easy sets have a total number of parts below the average number of parts per sets. Hard sets have more parts than the average number plus two times the standard deviation and medium sets are between the average number of parts and the average number plus two times the standard deviation. We calculated these values using excel, and we exported the new excel table into a CSV file and uploaded that file to Oracle Apex.





There was a terrible error happened during our table creation process. When we tried to set up the foreign keys of Table Inventory_parts, because there was a part_num in Table Inventory_parts but not in Table Parts, the system failed to connect these two tables by part_num. To fix this error, our team add the missing part_num into Table Parts with two null attributes:

```
SELECT PART_NUM
FROM LEGO_INVENTORY_PARTS
WHERE PART_NUM NOT IN (SELECT PART_NUM
                       FROM LEGO_PARTS);
```




```
INSERT INTO LEGO_PARTS
VALUES ('3650', NULL, NULL);
```

After the constraint error was cleaned up, our tables connected to each other to form our relational schema in Chapter 3. Here are a few samples of each data table that we created and populated to our application:




1.LEGO_COLORS

EDIT	ID	NAME	RGB	IS_TRANS
	-1	Unknown	0033B2	f
	0	Black	05131D	f
	1	Blue	0055BF	f
	2	Green	237841	f




2.LEGO_DIFFICULTY

EDIT	SET_NUM	ID	DIFFICULTY_LEVEL
	21304-1	1513	2
	21305-1	1514	2
	21306-1	1515	2




3. LEGO_INVENTORIES

EDIT	ID	VERSION	SET_NUM
	1624	1	8404-1
	1625	1	8877-1
	1627	1	7306-1




4.LEGO_INVENTORY_PARTS

EDIT	INVENTORY_ID	PART_NUM	COLOR_ID	QUANTITY	IS_SPARE
	489	2376	0	1	f
	491	2417	2	2	f
	492	27bc01	4	6	f




5.LEGO_INVENTORY_SETS

EDIT	INVENTORY_ID	SET_NUM	QUANTITY
	3271	71007-10	1
	3271	71007-11	1
	3271	71007-12	1




6.LEGO_PARTS

EDIT	PART_NUM	NAME	PART_CAT_ID
	10305	Minifig Helmet with Front Prongs [Plain]	27
	10305pr0001	Minifig Helmet with Front Prongs and Dark Purple Print (Magneto)	27
	1030b1	Set 1030 Activity Booklet 1 - Parts Tray Organizer Card	17




7. LEGO_PART_CATEGORIES

EDIT	ID	NAME
	1	Baseplates
	2	Bricks Printed
	3	Bricks Sloped

8. LEGO SETS

EDIT	SET_NUM	NAME	YEAR	THEME_ID	NUM_PARTS
	10592-1	Fire Truck	2015	504	26
	10593-1	Fire Station	2015	504	104
	10594-1	Sofia the First™ Royal Stable	2015	504	38

9.LEGO THEMES

EDIT	ID	NAME	PARENT_ID
	1	Technic	-
	2	Arctic Technic	1
	3	Competition	1

Chapter 5: Web Design and Data Analysis

For one of our overviews, we have created a dropdown list to select the year. This list allows us to look at the number of colors, the number of sets, and the number of parts produced for each year. To create this in APEX, we used the following SQL commands:

List of values :

```
select distinct(year) d, year r
from lego_sets
Order By year asc
```

Sets source SQL:

```
Select null link,
lego_sets.year label,
count(set_num) value
FROM lego_sets
WHERE (year = :P12_NEW OR :P12_NEW IS NULL)
group by lego_sets.year
```

Colors source SQL:

```
select NULL LINK,
count(distinct(lego_colors.id)) Value,
lego_sets.year Label
FROM lego_colors, lego_inventory_parts, lego_inventories,
lego_sets
WHERE lego_colors.id=lego_inventory_parts.color_id
AND lego_inventory_parts.inventory_id=lego_inventories.id
AND lego_inventories.set_num=lego_sets.set_num
AND (year = :P12_NEW OR :P12_NEW IS NULL)
GROUP BY lego_sets.year
```

Parts source SQL:

```
select NULL LINK,
count(distinct(lego_parts.part_num)) Value,
lego_sets.year Label
FROM lego_colors, lego_inventory_parts, lego_inventories,
lego_sets, lego_parts
WHERE lego_colors.id=lego_inventory_parts.color_id
AND lego_inventory_parts.inventory_id=lego_inventories.id
AND lego_inventories.set_num=lego_sets.set_num
```

```
AND lego_parts.part_num=lego_inventory_parts.part_num
AND (year = :P12_NEW OR :P12_NEW IS NULL)
GROUP BY lego_sets.year
```

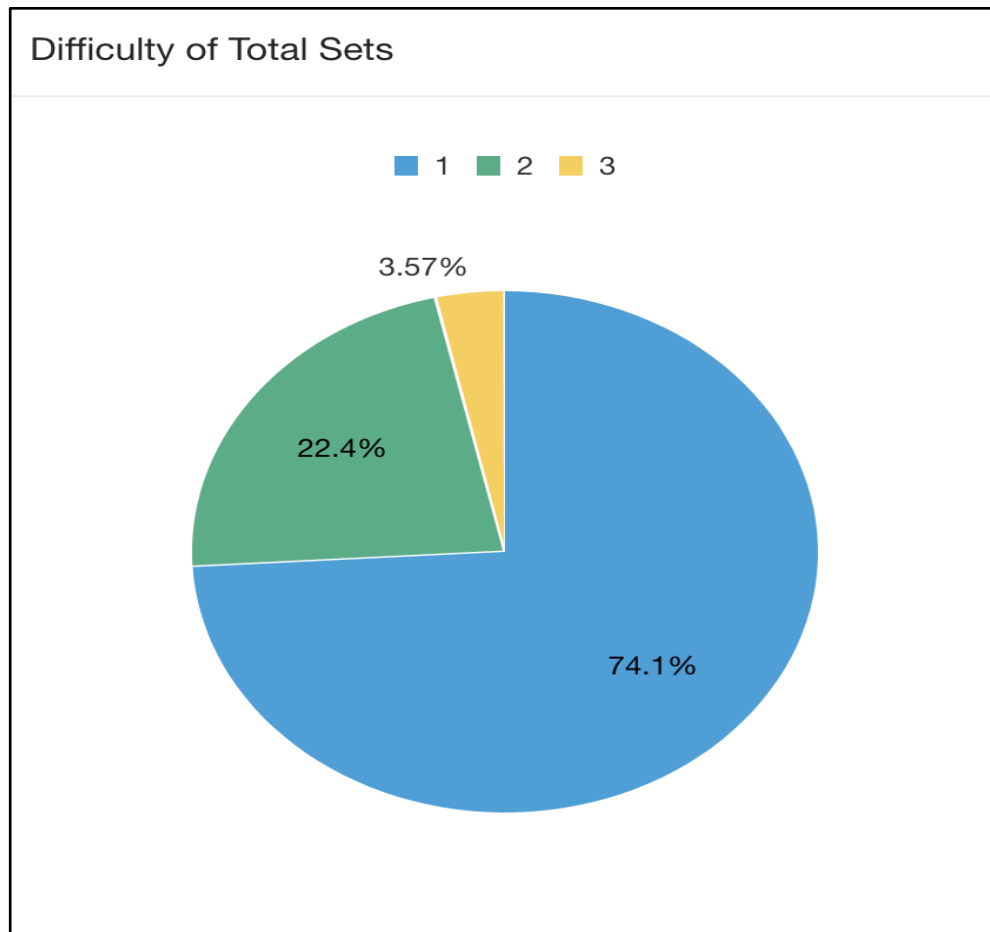
The results of this code are presented in the per_year tab.

The queries to answer our original questions are not as complicated.



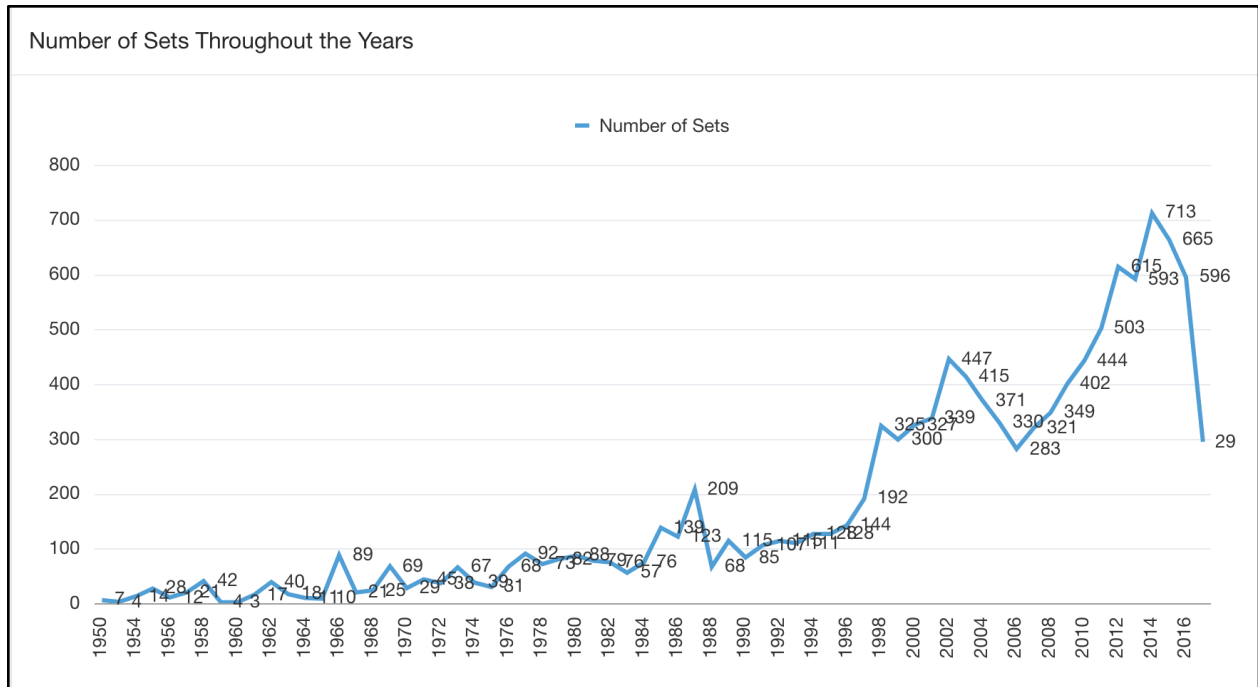
Sets with different difficulties

```
select null link,
lego_difficulty.difficulty_level label,
count(lego_difficulty.difficulty_level) value
From lego_sets,
Lego_difficulty
where lego_sets.set_num = lego_difficulty.set_num
group by null, lego_difficulty.difficulty_level
order by lego_difficulty.difficulty_level
```



❖ How many sets are produced per year, on average?

```
SELECT    count(distinct(set_num)), year
FROM      lego_sets
GROUP BY  year
```

What is the most complex set?

```
SELECT    num_parts, name
FROM      lego_sets
WHERE     num_parts = (SELECT max(num_parts)
                        FROM lego_sets)
```

Maximum number of parts	
Most parts ↑	Name
5922	Taj Mahal
1 - 1	

We also wrote some queries to find other interesting insights, for instance:

- What is the part category with most types of colors?

```
SELECT pc.id as part_category_id, pc.name as part_category_name, count(c.id)
as number_of_colors
```

```

FROM lego_part_categories pc, lego_inventory_parts ip, lego_parts p,
     lego_colors c
WHERE pc.id=p.part_cat_id and ip.part_num=p.part_num and ip.color_id=c.id
GROUP BY pc.id, pc.name
ORDER BY count(c.id) desc;

```

PART_CATEGORY_ID	PART_CATEGORY_NAME	NUMBER_OF_COLORS
11	Bricks	1432

- What parts were used in sets that were produced in 1965?
SELECT unique(p.part_num) as PART_NUM, p.name as PART_NAME, s.year
as Produced_Year
FROM lego_parts p, lego_inventory_parts ip, lego_inventories i, lego_sets s
WHERE p.part_num=ip.part_num and ip.inventory_id=i.id and
i.set_num=s.set_num
AND s.year=1965
ORDER BY s.year desc;

PART_NUM	PART_NAME	PRODUCED_YEAR
3001a	Brick 2 x 4 without Cross Supports	1965
3020	Plate 2 x 4	1965
3068a	Tile 2 x 2 without Groove	1965
3003	Brick 2 x 2	1965
132a	Tyre Smooth Old Style - Small	1965
wood03	Wooden Storage Box with Red Sliding Top	1965

Sources:

Kaggle Inc. (2017). LEGO Database. Retrieved from:
<https://www.kaggle.com/ratatman/lego-database>

LEGO (2017). The LEGO Group Reports Record Revenue in 2016. Retrieved from:
<https://www.lego.com/en-us/aboutus/news-room/2017/march/annual-results-2016>

Rebrickable (2017). What is Rebrickable? Retrieved from:
<https://rebrickable.com/about/>