

# The LEGO® Database

-- Johnathan Clementi --

-- Prof: Alan Labouseur --

-- Date: 12/4/2017 --

-- Assignment: Design Project LegoDB --

-- File name: legoDB\_documentation --

# Table of Contents

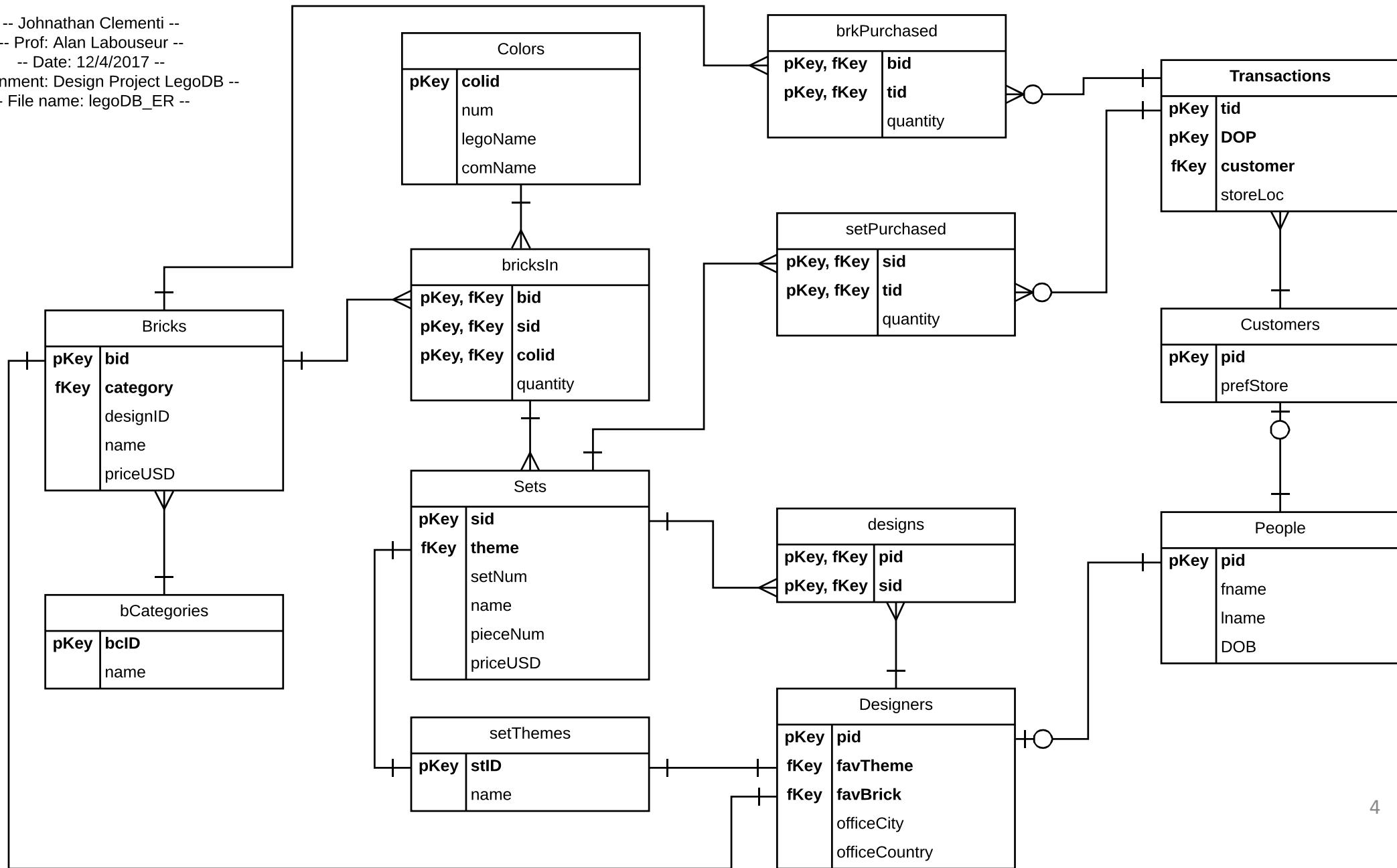
- Click to edit Master text styles
  - Second level
    - Third level
      - Fourth level
        - Fifth level

# Executive Summary

In preparation for the 2017 Christmas season, the Lego® Group was looking for an upgrade to their database. To do this, they hired Daniel Craig, a relational database designer from London in early September of 2017. Unfortunately, Mr. Craig has gone missing, and the Lego® Group needs this database as soon as possible so they can keep track of their holiday sales. Instead, The Lego® Group has hired a student from Marist College in Poughkeepsie, NY – Mr. Johnathan Clementi. This is Mr. Clementi's documentation for his Lego DB.

Designed in PostgreSQL, a relational database management system, the database contains information pertinent to running the Lego® Group such as bricks, sets, designers, customers, and transaction information. This paper explains the implementation the database: First is an entity relation (ER) diagram which graphically explains how the database's tables are related to one another. Next is a detailed listing of each table as well as queries to retrieve information (data with context) from such tables. After this stored procedures, views, reports, and triggers to enhance usability of the database are found. Finally, known problems and future enhancements are addressed.

-- Johnathan Clementi --  
 -- Prof: Alan Labousseur --  
 -- Date: 12/4/2017 --  
 -- Assignment: Design Project LegoDB --  
 -- File name: legoDB\_ER --



# Table declarations and explanations



# Brick Categories Table

- This table contains 17 categories of Lego® bricks
- Functional Dependencies:
  - $(bcID) \rightarrow name$

```
CREATE TABLE bCategories (
    bcID  CHAR(4) PRIMARY KEY NOT NULL,
    name  TEXT NOT NULL
);
```

	bcid character (4)	name text
1	bc01	Beams
2	bc02	Bricks
3	bc03	Bricks w/ Bows and Arc...
4	bc04	Bricks, special
5	bc05	Bricks, Technic 4.85
6	bc06	Connectors
7	bc07	Decoration Elements
8	bc08	Figure parts
9	bc09	Figure Clothing
10	bc10	Figure Heads and Masks
11	bc11	Figure Accessories
12	bc12	Frames, Windows, Wall...
13	bc13	Functional Elements
14	bc14	Plates
15	bc15	Plates, special
16	bc16	Signs, Flags, and Poles
17	bc17	Transportation

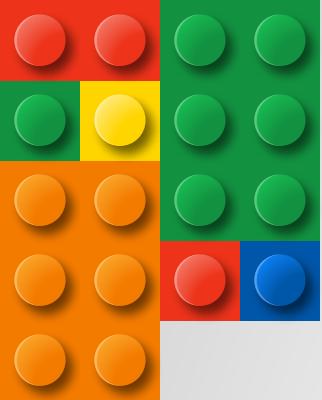


# Colors Table

- This table contains 19 of the most common colors of Lego® bricks. Colid is an artificial key, num is a Lego® number assigned to the color, legoName is the official name given to the color by Lego®, and comName is the common name
- Functional Dependencies:
  - $(\text{colid}) \rightarrow \text{num}, \text{legoName}, \text{comName}$

```
CREATE TABLE Colors (
    colid      CHAR(3) PRIMARY KEY NOT NULL,
    num        VARCHAR(3) NOT NULL,
    legoName   TEXT NOT NULL,
    comName    TEXT
);
```

	colid character (3)	num character varying (3)	legoName text	comname text
1	c01	1	White	White
2	c02	5	Brick Yellow	Tan
3	c03	18	Nougat	Flesh
4	c04	21	Bright Red	Red
5	c05	23	Bright Blue	Blue
6	c06	24	Bright Yell...	Yellow
7	c07	26	Black	Black
8	c08	28	Dark Green	Green
9	c09	323	Aqua	Unikitty Bl...
10	c10	326	Spring Yell...	Unikitty Gr...
11	c11	40	Transparent	[null]
12	c12	41	Transpare...	[null]



# Bricks Table

- This table contains 65 examples of Lego® bricks. Includes Lego®'s numbering system (designID), the category in which the brick belongs, the name of the brick, and the price of 1 individual brick
- (bid) → designID, category, name, priceUSD

```
CREATE TABLE Bricks (
    bid          VARCHAR(7) PRIMARY KEY NOT NULL,
    designID    VARCHAR(7) NOT NULL, -- designID is Lego's identifier for a type of brick --
    category    CHAR(4) NOT NULL REFERENCES bCategories(bcID),
    name        TEXT,
    priceUSD    NUMERIC(10,2)
);
```

	<b>bid</b> character varying (7)	<b>designid</b> character varying (7)	<b>category</b> character (4)	<b>name</b> text	<b>priceusd</b> numeric (10,2)
1	b001	3005	bc02	Brick 1x1	0.07
2	b002	3004	bc02	Brick 1x2	0.10
3	b003	3622	bc02	Brick 1x3	0.14
4	b004	3010	bc02	Brick 1x4	0.15
5	b005	3009	bc02	Brick 1x6	0.24



# *Set Themes Table*

- This table contains 27 themes (currently in production) of Lego® sets. These are the themes that are currently in production.
- (stID) → name

```
CREATE TABLE SetThemes (
    stID  CHAR(4) PRIMARY KEY NOT NULL,
    name  TEXT NOT NULL
);
```

	stid character (4)	name text
1	st01	Architecture
2	st02	The Lego Movie
3	st03	Boost
4	st04	City
5	st05	City: Octan
6	st06	City: Vehicles
7	st07	Classic
8	st08	Creator



# Sets Table

- This table contains 30 examples of Lego® sets. setNum is a Lego® assigned unique value for each of their sets.
- (sid) → setNum, theme, name, pieceNum, priceUSD

```
CREATE TABLE Sets (
    sid          CHAR(4) PRIMARY KEY NOT NULL,
    setNum       VARCHAR(7) NOT NULL, -- Lego's set identifier --
    theme        CHAR(4) NOT NULL REFERENCES setThemes(stID),
    name         TEXT NOT NULL,
    pieceNum     INT NOT NULL,
    priceUSD    NUMERIC(10,2) NOT NULL
);
```

```
SELECT s.sid, s.setNum, st.name AS setTheme, s.name, s.pieceNum, s.priceUSD
FROM Sets s INNER JOIN setThemes st on s.theme = st.stID;
```

	<b>sid</b> character	<b>setnum</b> character	<b>settheme</b> text	<b>name</b> text	<b>pieceum</b> integer	<b>priceusd</b> numeric (10,2)
1	s001	10179	Lego Star Wars	Millennium Falcon	5195	3899.99
2	s002	10221	Lego Star Wars	Super Star Destroyer	3152	969.99
3	s003	10256	Architecture	Taj Mahal	5922	4499.99
4	s004	10214	Creator	Tower Bridge	4295	239.99

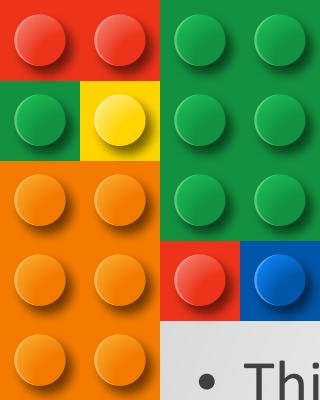


# People Table

- This table contains data for people involved with Lego® in some fashion (both customers and designers)
- (pid) → fname, lname, DOB

```
CREATE TABLE People (
    pid          CHAR(4) PRIMARY KEY NOT NULL,
    fname        TEXT NOT NULL,
    lname        TEXT NOT NULL,
    DOB          DATE NOT NULL
);
```

	<b>pid</b> character (4)	<b>fname</b> text	<b>lname</b> text	<b>dob</b> date
1	p001	Johnathan	Clementi	1996-05-12
2	p002	Alan	Labouseur	1970-01-15
3	p003	Megan	Clementi	1968-11-15
4	p004	Paul	Clementi	1966-01-25
5	p005	Kathryn	Rivera	1998-11-12



# Designers Table

- This table contains 12 examples of designers of Lego® sets. A designer is a person, thus this table is a subset of the people table. FavTheme and FavBrick are foreign keys to setThemes and bricks respectively. Finally the table has information about the designer's office location.
- (pid) → favTheme, favBrick, officeCity, officeCountry

```
CREATE TABLE Designers (
    pid                         CHAR(4) NOT NULL REFERENCES People(pid),
    favTheme                     CHAR(4) REFERENCES setThemes(stID),
    favBrick                     CHAR(4) REFERENCES bricks(bid),
    officeCity                   TEXT,
    officeCountry                TEXT,
    PRIMARY KEY(pid)
);
```



# Designers Table (cont.)

```
SELECT p.fname, p.lname, st.name as favTheme, b.name as favBrick, d.officeCity, d.officeCountry  
FROM Designers d INNER JOIN People p ON d.pid = p.pid  
    INNER JOIN setThemes st ON d.favTheme = st.stID  
    INNER JOIN bricks b ON d.favBrick = b.bid;
```

	<b>fname</b> text	<b>lname</b> text	<b>favtheme</b> text	<b>favbrick</b> text	<b>officecity</b> text	<b>officecountry</b> text
1	Alan	Labouseur	The Lego Movie	Plate 2x12	London	England
2	Megan	Clementi	Lego Star Wars	Brick 1x1	Prague	Czech Republic
3	Paul	Clementi	Technic	Plate 4x12	Singapore	Singapore
4	Jorgen Vig	Knudstorp	Duplo	Plate 1x2	Philadelphia	United States ...



# Customers Table

- This table contains 12 examples of customers of Lego® sets. A customer is a person, thus this table is a subset of the people table. PrefStore refers to the customer's preferred store, or the store they shop at the most.
- $(pid) \rightarrow prefStore$

```
CREATE TABLE Customers (
    pid          CHAR(4) NOT NULL REFERENCES People(pid),
    prefStore    TEXT NOT NULL, -- Customer's preferred store --
    PRIMARY KEY(pid)
);
```

```
SELECT p.fname, p.lname, prefStore
FROM Customers c INNER JOIN People p on c.pid = p.pid;
```

	fname text	lname text	prefstore text
1	Johnathan	Clementi	King of Prussia
2	Megan	Clementi	San Fransisco
3	Kathryn	Rivera	New York City
4	Alex	Antaki	Shanghai
5	Matthew	Oakley	Dallas



# Transactions Table

- This table contains all of the transactions to purchase Lego® bricks and sets
- (tid) → customer, storeLoc, DOP

```
CREATE TABLE transactions (
    tid           CHAR(4) PRIMARY KEY NOT NULL,
    customer     CHAR(4) NOT NULL REFERENCES customers(pid),
    storeLoc      TEXT NOT NULL,
    DOP           DATE NOT NULL -- Date of Purchase --
);
```

```
SELECT t.tid, p.fname, p.lname, t.storeLoc, t.DOP
FROM transactions t INNER JOIN customers c on t.customer = c.pid
                     INNER JOIN people p on c.pid = p.pid;
```

	<b>tid</b> character	<b>fname</b> text	<b>lname</b> text	<b>storeloc</b> text	<b>dop</b> date
1	t001	Johnathan	Clementi	King of Prussia	2017-12-01
2	t002	Megan	Clementi	San Francisco	2017-12-02
3	t003	Megan	Clementi	New York City	2017-12-03
4	t004	Kathryn	Rivera	Shanghai	2017-12-04
5	t005	Alex	Antaki	Shanghai	2017-12-01
6	t006	Matthew	Oakley	Moscow	2017-12-02



# Sets Purchased Table

- This represents the Lego® sets purchased, and is therefore a subset of the transaction table. Because there can be many sets purchased in the same transaction, there must be an associative entity (the setsPurchased table) to map every intersection while maintaining uniqueness and referential integrity.
- $(sid, tid) \rightarrow quantity$

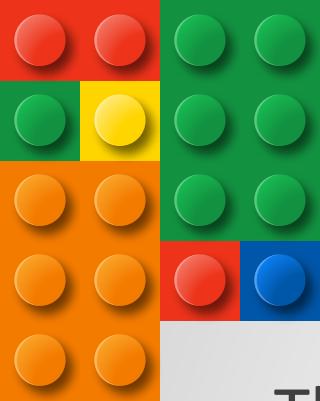
```
CREATE TABLE setPurchased (
    sid          CHAR(4) NOT NULL REFERENCES sets(sid),
    tid          CHAR(4) NOT NULL REFERENCES transactions(tid),
    quantity     INT NOT NULL
);
```



# *Sets Purchased Table (cont.)*

```
SELECT sp.tid, s.name AS setName, p.fname, p.lname, t.storeLoc, t.DOP, sp.quantity
FROM setPurchased sp INNER JOIN sets s on sp.sid = s.sid
    INNER JOIN transactions t on sp.tid = t.tid
    INNER JOIN customers c on t.customer = c.pid
    INNER JOIN people p on c.pid = p.pid;
```

	<b>tid</b> character	<b>setname</b> text	<b>fname</b> text	<b>lname</b> text	<b>storeloc</b> text	<b>dop</b> date	<b>quantity</b> integer
1	t001	Millennium Falcon	Johnathan	Clementi	King of Prussia	2017-12-01	1
2	t002	Airport Rescue Vehicle	Megan	Clementi	San Francisco	2017-12-02	2
3	t006	Death Star	Matthew	Oakley	Moscow	2017-12-02	2
4	t007	Princess: Ariels Magica...	Eric	Seltzer	Singapore	2017-12-03	1



# Bricks Purchased Table

- This represents the Lego® bricks purchased, and is therefore a subset of the transaction table. For the same reason that sets purchased table is an associative entity, the bricks purchased table is an associative entity.
- $(bid, tid) \rightarrow quantity$

```
CREATE TABLE brkPurchased (
    bid          CHAR(4) NOT NULL REFERENCES bricks(bid),
    tid          CHAR(4) NOT NULL REFERENCES transactions(tid),
    quantity     INT NOT NULL
);
```



# Bricks Purchased Table (cont.)

```
SELECT bkp.tid, b.name AS brickName, p.fname, p.lname, t.storeLoc, t.DOP, bkp.quantity  
FROM brkPurchased bkp      INNER JOIN bricks b on bkp.bid = b.bid  
                           INNER JOIN transactions t on bkp.tid = t.tid  
                           INNER JOIN customers c on t.customer = c.pid  
                           INNER JOIN people p on c.pid = p.pid;
```

	<b>tid</b> character (4)	<b>brickname</b> text	<b>fname</b> text	<b>lname</b> text	<b>storeloc</b> text	<b>dop</b> date	<b>quantity</b> integer
1	t002	FFlat Tile 1x3	Megan	Clementi	San Francisco	2017-12-02	20
2	t003	Brick 2x4	Megan	Clementi	New York City	2017-12-03	185
3	t004	Brick 1x6	Kathryn	Rivera	Shanghai	2017-12-04	34
4	t005	Brick 1x8	Alex	Antaki	Shanghai	2017-12-01	95



# Bricks In Table

- This table is an associative entity that explains which bricks are in which sets and which color they appear in. Because the same brick can appear in the same set, but in a different color, the primary key of this table is the composite of bid, sid, and colid.
- (bid, sid, colid) → quantity

```
CREATE TABLE bricksIn (
    bid          CHAR(4) NOT NULL REFERENCES bricks(bid),
    sid          CHAR(4) NOT NULL REFERENCES sets(sid),
    colid        CHAR(4) NOT NULL REFERENCES colors(colid),
    quantity     INT NOT NULL,
PRIMARY KEY(bid, sid, colid)
);
```



# Bricks In Table (cont.)

```
SELECT s.name AS setName, b.name AS brickName, c.legoName AS Color, bi.quantity  
FROM bricksIn bi INNER JOIN sets s ON bi.sid = s.sid  
INNER JOIN bricks b ON bi.bid = b.bid  
INNER JOIN colors c ON bi.colid = c.colid;
```

	<b>setname</b> text	<b>brickname</b> text	<b>color</b> text	<b>quantity</b> integer
1	Millennium Falcon	Brick 1x4	White	2
2	Millennium Falcon	Brick 1x6	Brick Yellow	234
3	Millennium Falcon	Brick 1x8	Nougat	13
4	Super Star Destroyer	Plate 2x16	Bright Red	52
5	Super Star Destroyer	Brick W/ B...	Bright Blue	65
6	Super Star Destroyer	Mini Leg	Bright Yellow	24
7	Taj Mahal	Plate 4x4	Black	54
8	Taj Mahal	Plate 2x2	Dark Green	4
9	Taj Mahal	Brick 1x3	Aqua	23
10	Tower Bridge	Brick 1x10	Spring Yello...	67
11	Tower Bridge	Brick 1x2x2	Transparent	73
12	Tower Bridge	Brick W/ Ar...	Transparent..	33



# *Designs* Table

- This table is an associative entity that connects designers to the sets that they have designed. Because Lego® design can be a collaborative effort, thus many designers which design many sets, an associative entity is necessary.
- (pid, sid) →

```
CREATE TABLE designs (
    pid          CHAR(4) NOT NULL REFERENCES designers(pid),
    sid          CHAR(4) NOT NULL REFERENCES sets(sid),
    PRIMARY KEY(pid, sid)
);
```



# *Designs* Table (cont.)

```
CREATE TABLE designs (
    pid          CHAR(4) NOT NULL REFERENCES designers(pid),
    sid          CHAR(4) NOT NULL REFERENCES sets(sid),
    PRIMARY KEY(pid, sid)
);
```

	<b>fname</b> text	<b>Iname</b> text	<b>setname</b> text
1	Bob	Odenkirk	Millennium Falcon
2	Bob	Odenkirk	Super Star Destroyer
3	Bob	Odenkirk	Taj Mahal
4	Grant	Dixon	Tower Bridge
5	Alan	Labouseur	Grand Carousal
6	Alan	Labouseur	Sandcrawler



# Reports, Views, Stored Procedures, and Triggers

# Sets Without Bricks Report

- This query outputs a list of customers who have purchased a set(s) but have not purchased a brick(s)

```
SELECT DISTINCT p.pid, p.fname, p.lname
FROM people p INNER JOIN customers c ON p.pid = c.pid
WHERE c.pid IN (Select c1.pid
                 FROM setPurchased sp INNER JOIN transactions t1 ON sp.tid = t1.tid
                                         INNER JOIN customers c1 ON t1.customer = c1.pid
                 WHERE c1.pid NOT IN (Select c2.pid
                                         FROM brkPurchased bkp INNER JOIN transactions t2 ON bkp.tid = t2.tid
                                         INNER JOIN customers c2 ON t2.customer = c2.pid)
)
ORDER BY p.fname ASC;
```

	<b>pid</b> character (4)	<b>fname</b> text	<b>lname</b> text
1	p007	Alex	Mahlm...
2	p019	Bradley	Mobek
3	p017	Eric	Seltzer
4	p025	Walter...	White

# Bricks Without Sets Report

- This query outputs a list of customers who have purchased a brick(s) but have not purchased a set(s)

```
SELECT DISTINCT p.pid, p.fname, p.lname
FROM people p INNER JOIN customers c ON p.pid = c.pid
WHERE c.pid IN (Select c1.pid
                 FROM brkPurchased bkp INNER JOIN transactions t1 ON bkp.tid = t1.tid
                                         INNER JOIN customers c1 ON t1.customer = c1.pid
                 WHERE c1.pid NOT IN (Select c2.pid
                                         FROM setPurchased sp INNER JOIN transactions t2 ON sp.tid = t2.tid
                                         INNER JOIN customers c2 ON t2.customer = c2.pid)
)
ORDER BY p.fname ASC;
```

	<b>pid</b> character (4)	<b>fname</b> text	<b>lname</b> text
1	p005	Kathryn	Rivera

# Most Selling Stores View

- This view outputs the most selling store based on number of transactions.

```
CREATE OR REPLACE VIEW mostSellingStores AS
  SELECT t.storeloc, COUNT(t.tid)
    FROM transactions t
   GROUP BY t.storeloc
  HAVING (COUNT(t.tid) >= 0)
 ORDER BY COUNT(t.tid) DESC;
```

	storeloc text	count bigint
1	King of P...	3
2	Singapore	3
3	New Yor...	3
4	Shanghai	3
5	Moscow	2
6	Sydney	2
7	San Fran...	1
8	Philadelp...	1
9	Prague	1
10	San Fran...	1



# Known Problems / Future Enhancements



- For the purposes of this exercise, I created an artificial key for most of my tables. I know that Lego uses 'unique' identifiers for each of their bricks and sets, but I didn't want to use those to guarantee atomicity. I wonder if these identifiers could be utilized.  
(bricks.designID and sets.setNum)
- While the color table is accessible by means of the bricksIn table, this data can only be applied to sets. A quick fix would be to add a reference clause in the brkPurchased table that allows for the addition of color data.
- New subtypes of people could be implemented – such as store managers and sales associates