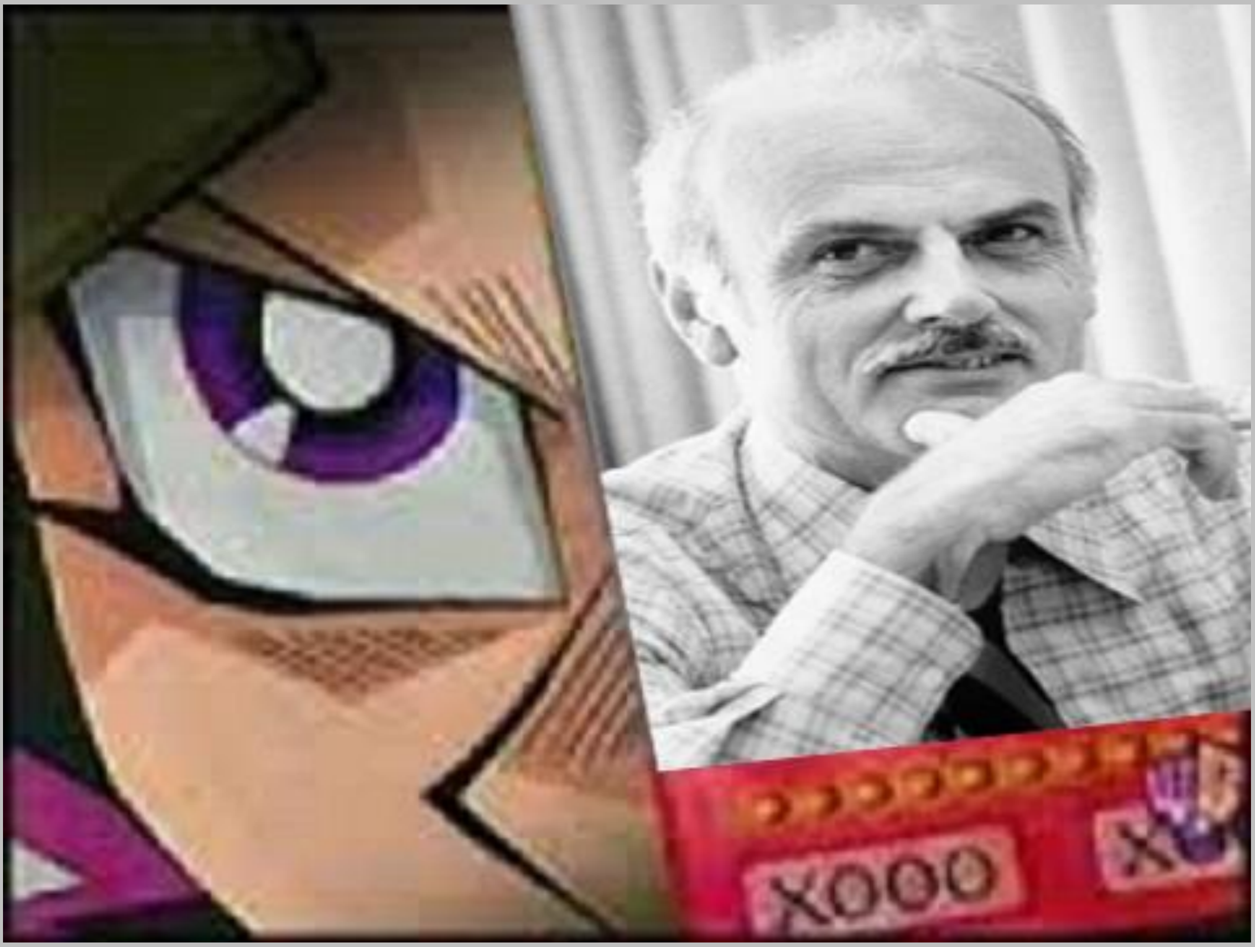
YugiohDB



By Graham Burek

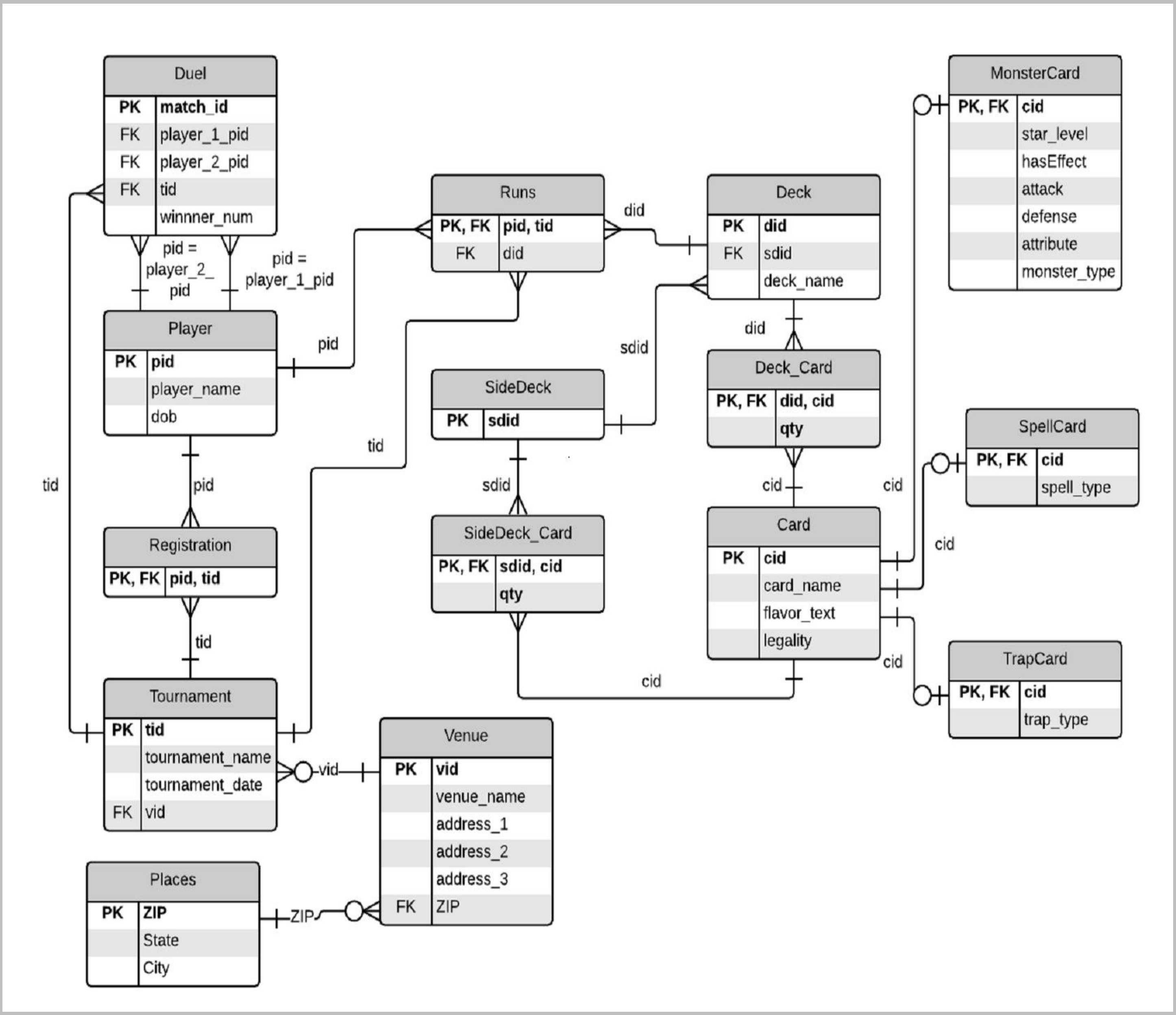
April 26, 2016

Table of Contents

A common complaint from serious card game players is that card companies often do a poor job of balancing their card games. Because of this, players create decks from a small group of cards that are so effective that they define the playstyle of the game. This can eventually cause the game to become stale, since everybody runs a twist on the dominant deck. YugiohDB is a database built to help remedy this problem.

YugiohDB is a database built to manage Yu-Gi-Oh! card game tournaments. It serves as a centralized way to manage tournament registration for official tournament events all over the world. It has the structure necessary to analyze card usage so that card balancing decisions can be made based on empirical results.

An outline of the database is presented in the following pages. Tested on PostgreSQL 9.5.


Duel
PK match_id
FK player_1_pid
FK player_2_pid
FK tid
winnner_num
pid = player_2_pid
pid = player_1_pid
Player
PK pid
player_name
dob
pid
Registration
PK, FK pid, tid
tid
Tournament
PK tid
tournament_name
tournament_date
FK vid
vid
Venue
PK vid
venue_name
address_1
address_2
address_3
FK ZIP
ZIP
Places
PK ZIP
State
City
Runs
PK, FK pid, tid
FK did
did
tid
Deck
PK did
FK sdid
deck_name
sdid
SideDeck
PK sdid
sdid
SideDeck_Card
PK, FK sdid, cid
qty
Deck_Card
PK, FK did, cid
qty
cid
Card
PK cid
card_name
flavor_text
legality
MonsterCard
PK, FK cid
star_level
hasEffect
attack
defense
attribute
monster_type
SpellCard
PK, FK cid
spell_type
TrapCard
PK, FK cid
trap_type

E/R Diagram

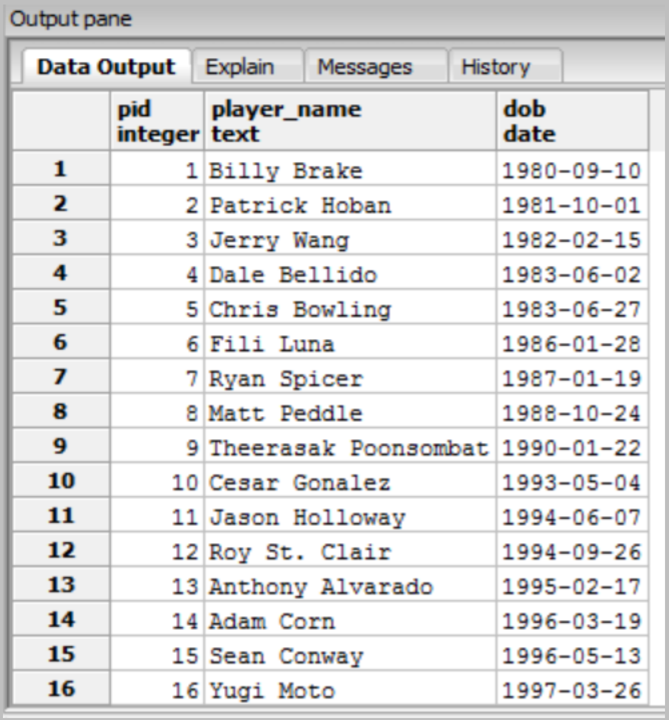
Player Table

A table that keeps track of players that have attended/registered a tournament.

```
CREATE TABLE Player
(
pid INT NOT NULL UNIQUE,
player_name TEXT NOT NULL,
dob DATE NOT NULL,
PRIMARY KEY(pid)
);
```

Dependencies: pid → player_name, dob

Sample Data:



Output pane

Data Output | Explain | Messages | History

	pid integer	player_name text	dob date
1	1	Billy Brake	1980-09-10
2	2	Patrick Hoban	1981-10-01
3	3	Jerry Wang	1982-02-15
4	4	Dale Bellido	1983-06-02
5	5	Chris Bowling	1983-06-27
6	6	Fili Luna	1986-01-28
7	7	Ryan Spicer	1987-01-19
8	8	Matt Peddle	1988-10-24
9	9	Theerasak Poonsombat	1990-01-22
10	10	Cesar Gonalez	1993-05-04
11	11	Jason Holloway	1994-06-07
12	12	Roy St. Clair	1994-09-26
13	13	Anthony Alvarado	1995-02-17
14	14	Adam Corn	1996-03-19
15	15	Sean Conway	1996-05-13
16	16	Yugi Moto	1997-03-26

Table: Player

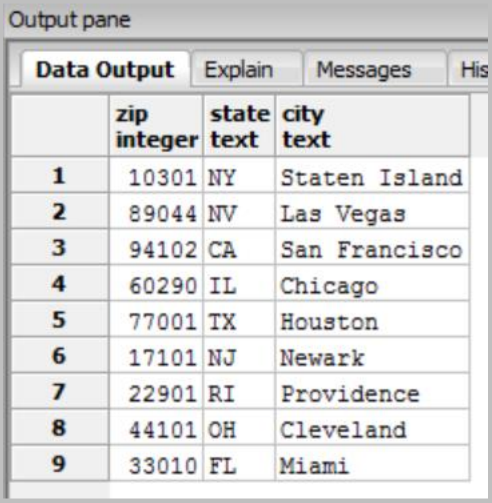
Places Table

A table that keeps track of ZIP codes and their associated states and cities.

```
CREATE TABLE Places
(
ZIP INT NOT NULL UNIQUE,
state TEXT NOT NULL,
city TEXT NOT NULL,
PRIMARY KEY(ZIP)
);
```

Dependencies: ZIP → state, city

Sample Data:



Output pane

Data Output | Explain | Messages | His

	zip integer	state text	city text
1	10301	NY	Staten Island
2	89044	NV	Las Vegas
3	94102	CA	San Francisco
4	60290	IL	Chicago
5	77001	TX	Houston
6	17101	NJ	Newark
7	22901	RI	Providence
8	44101	OH	Cleveland
9	33010	FL	Miami

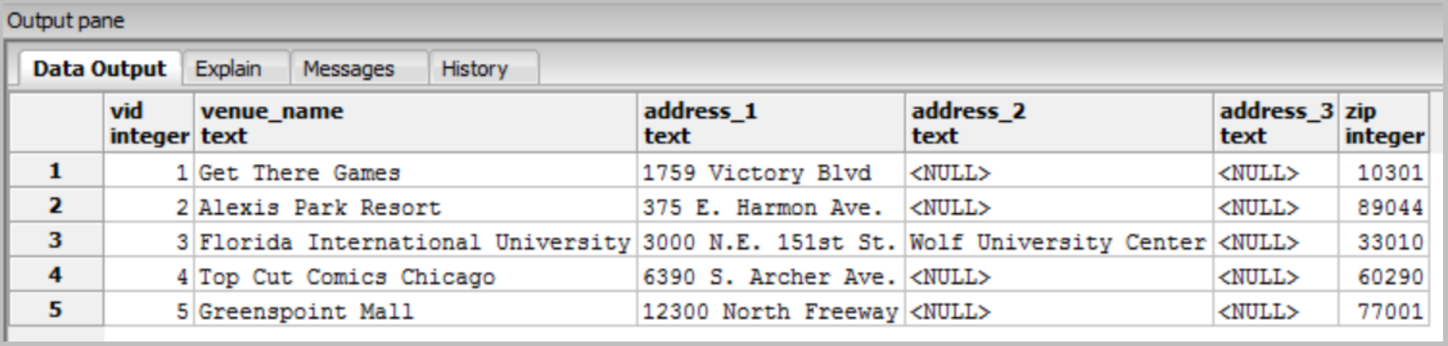
Venue Table

A table that keeps track of tournament locations.

```
CREATE TABLE Venue
(
vid INT NOT NULL UNIQUE,
venue_name TEXT NOT NULL,
address_1 TEXT NOT NULL,
address_2 TEXT,
address_3 TEXT,
ZIP INT NOT NULL references Places(ZIP),
PRIMARY KEY(vid)
);
```

Dependencies: vid → venue_name, address_1, 2, 3, ZIP

Sample Data:



Output pane

Data Output | Explain | Messages | History

	vid integer	venue_name text	address_1 text	address_2 text	address_3 text	zip integer
1	1	Get There Games	1759 Victory Blvd	<NULL>	<NULL>	10301
2	2	Alexis Park Resort	375 E. Harmon Ave.	<NULL>	<NULL>	89044
3	3	Florida International University	3000 N.E. 151st St.	Wolf University Center	<NULL>	33010
4	4	Top Cut Comics Chicago	6390 S. Archer Ave.	<NULL>	<NULL>	60290
5	5	Greenspoint Mall	12300 North Freeway	<NULL>	<NULL>	77001

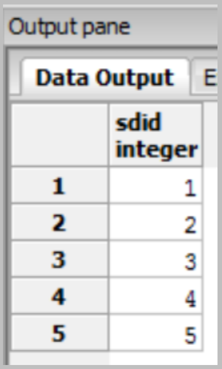
SideDeck Table

A representative table for a side deck.

```
CREATE TABLE SideDeck
(
sdid INT NOT NULL UNIQUE,
PRIMARY KEY(sdid)
);
```

Dependencies: sdid →

Sample Data:



	sdid integer
1	1
2	2
3	3
4	4
5	5

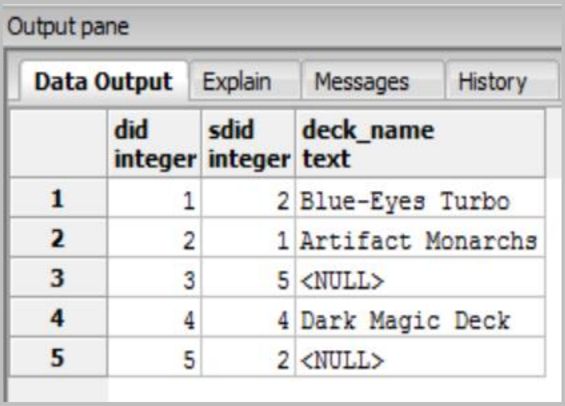
Deck Table

A representative table for a deck.

```
CREATE TABLE Deck
(
did INT NOT NULL UNIQUE,
sdid INT NOT NULL references SideDeck(sdid),
deck_name TEXT,
PRIMARY KEY(did)
);
```

Dependencies: did → sdid, deck_name

Sample Data:



	did integer	sdid integer	deck_name text
1	1	2	Blue-Eyes Turbo
2	2	1	Artifact Monarchs
3	3	5	<NULL>
4	4	4	Dark Magic Deck
5	5	2	<NULL>

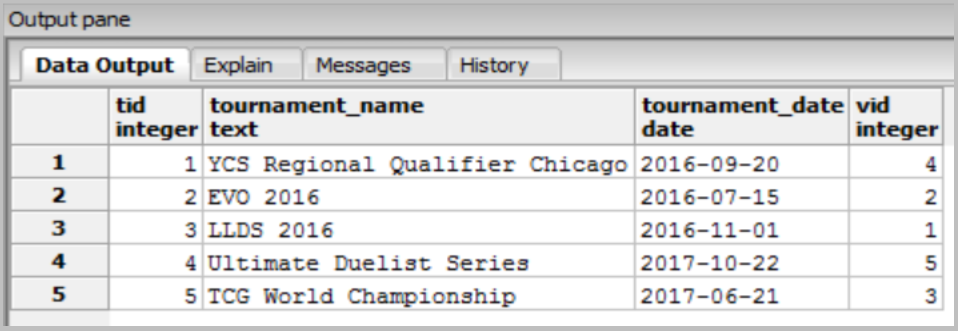
Tournament Table

A table that keeps track of tournaments.

```
CREATE TABLE Tournament
(
tid INT NOT NULL UNIQUE,
tournament_name TEXT NOT NULL,
tournament_date DATE NOT NULL CHECK(tournament_date > now()),
vid INT NOT NULL references Venue(vid),
PRIMARY KEY(tid)
);
```

Dependencies: tid → tournament_name, date, vid

Sample Data:



	tid integer	tournament_name text	tournament_date date	vid integer
1	1	YCS Regional Qualifier Chicago	2016-09-20	4
2	2	EVO 2016	2016-07-15	2
3	3	LLDS 2016	2016-11-01	1
4	4	Ultimate Duelist Series	2017-10-22	5
5	5	TCG World Championship	2017-06-21	3

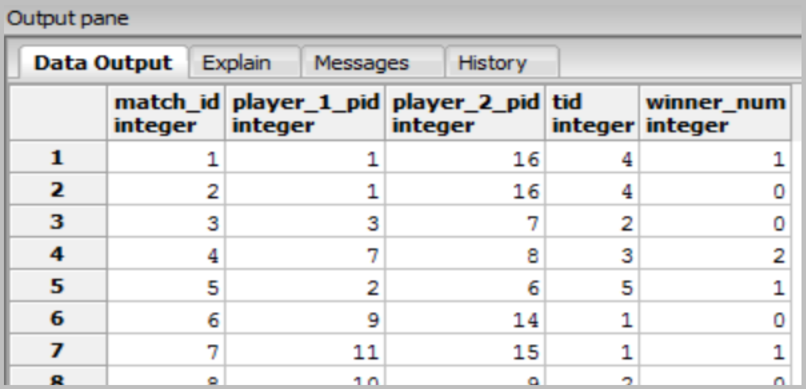
Duel Table

A table that keeps track of all the duels a player has in a tournament.

```
CREATE TABLE Duel
(
match_id INT NOT NULL UNIQUE,
player_1_pid INT NOT NULL references Player(pid),
player_2_pid INT NOT NULL references Player(pid),
tid INT NOT NULL references Tournament(tid),
winner_num INT NOT NULL CHECK(winner_num IN(0,1,2)),
PRIMARY KEY(match_id)
);
```

Dependencies: match_id → player_1_pid, player_2_pid, tid, winner_num

Sample Data:



	match_id integer	player_1_pid integer	player_2_pid integer	tid integer	winner_num integer
1	1	1	16	4	1
2	2	1	16	4	0
3	3	3	7	2	0
4	4	7	8	3	2
5	5	2	6	5	1
6	6	9	14	1	0
7	7	11	15	1	1
8	8	10	9	2	0

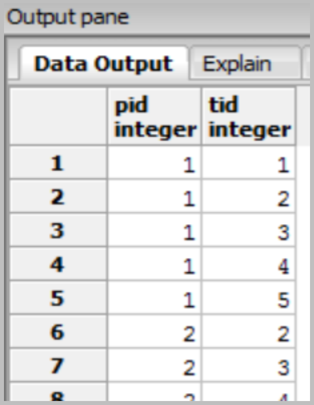
Registration Table

A table that keeps track of what tournaments every player is registered for.

```
CREATE TABLE Registration
(
pid INT NOT NULL references Player(pid),
tid INT NOT NULL references Tournament(tid),
PRIMARY KEY(pid,tid)
);
```

Dependencies: (pid,tid) →

Sample Data:



	pid integer	tid integer
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	2	2
7	2	3

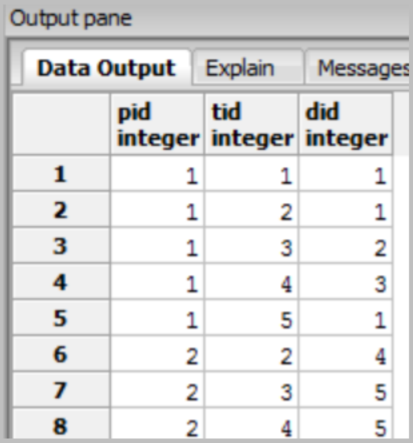
Runs Table

A table that keeps track of the decks players run in a given tournament.

```
CREATE TABLE Runs
(
pid INT NOT NULL references Player(pid),
tid INT NOT NULL references Tournament(tid),
did INT NOT NULL references Deck(did),
PRIMARY KEY(pid,tid)
);
```

Dependencies: (pid,tid) → did

Sample Data:



	pid integer	tid integer	did integer
1	1	1	1
2	1	2	1
3	1	3	2
4	1	4	3
5	1	5	1
6	2	2	4
7	2	3	5
8	2	4	5

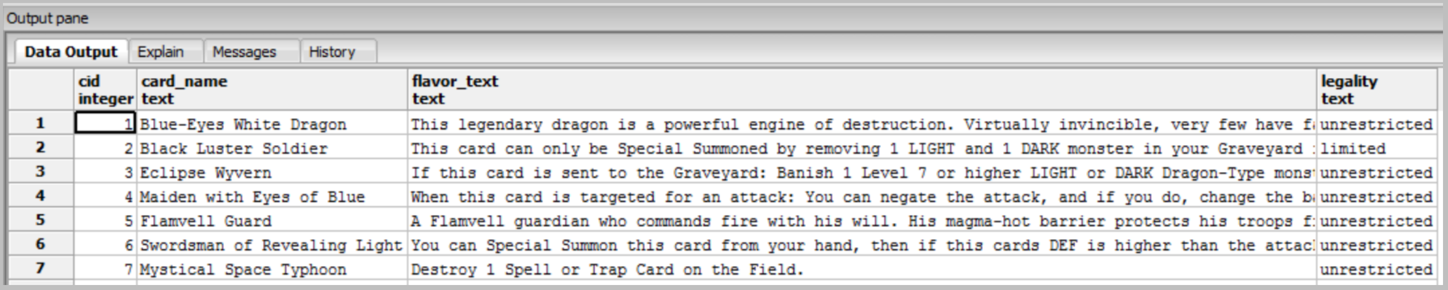
Card Table

A table that contains basic information common to all YuGiOh cards.

```
CREATE TABLE Card
(
cid INT NOT NULL UNIQUE,
card_name TEXT NOT NULL UNIQUE,
flavor_text TEXT NOT NULL,
legality TEXT NOT NULL CHECK(legality IN('unrestricted','semi-
limited','limited','forbidden')),
PRIMARY KEY(cid)
);
```

Dependencies: cid → card_name, flavor_text, legality

Sample Data:



Output pane

Data Output | Explain | Messages | History

	cid integer	card_name text	flavor_text text	legality text
1	1	Blue-Eyes White Dragon	This legendary dragon is a powerful engine of destruction. Virtually invincible, very few have f	unrestricted
2	2	Black Luster Soldier	This card can only be Special Summoned by removing 1 LIGHT and 1 DARK monster in your Graveyard	limited
3	3	Eclipse Wyvern	If this card is sent to the Graveyard: Banish 1 Level 7 or higher LIGHT or DARK Dragon-Type mons	unrestricted
4	4	Maiden with Eyes of Blue	When this card is targeted for an attack: You can negate the attack, and if you do, change the b	unrestricted
5	5	Flamvell Guard	A Flamvell guardian who commands fire with his will. His magma-hot barrier protects his troops f	unrestricted
6	6	Swordsman of Revealing Light	You can Special Summon this card from your hand, then if this cards DEF is higher than the attac	unrestricted
7	7	Mystical Space Typhoon	Destroy 1 Spell or Trap Card on the Field.	unrestricted

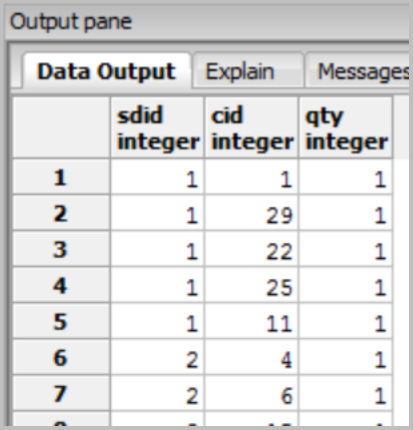
SideDeck_Card Table

A table that keeps track of what cards are in what side deck.

```
CREATE TABLE SideDeck_Card
(
sdid INT NOT NULL references SideDeck(sdid),
cid INT NOT NULL references Card(cid),
qty INT NOT NULL CHECK(qty IN(1,2,3)),
PRIMARY KEY(sdid,cid)
);
```

Dependencies: (sdid,cid) → qty

Sample Data:



Output pane

Data Output | Explain | Messages

	sdid integer	cid integer	qty integer
1	1	1	1
2	1	29	1
3	1	22	1
4	1	25	1
5	1	11	1
6	2	4	1
7	2	6	1

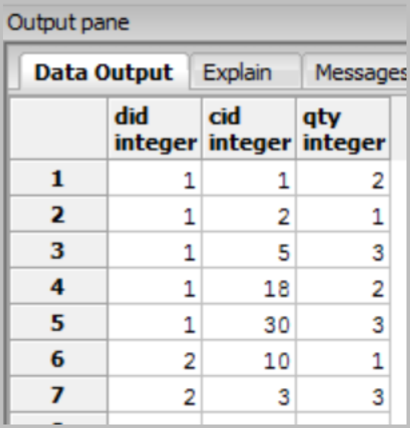
Deck_Card Table

A table that keeps track of what cards are in what deck.

```
CREATE TABLE Deck_Card
(
did INT NOT NULL references Deck(did),
cid INT NOT NULL references Card(cid),
qty INT NOT NULL CHECK(qty IN(1,2,3)),
PRIMARY KEY(did,cid)
);
```

Dependencies: (did,cid) → qty

Sample Data:



	did integer	cid integer	qty integer
1	1	1	2
2	1	2	1
3	1	5	3
4	1	18	2
5	1	30	3
6	2	10	1
7	2	3	3

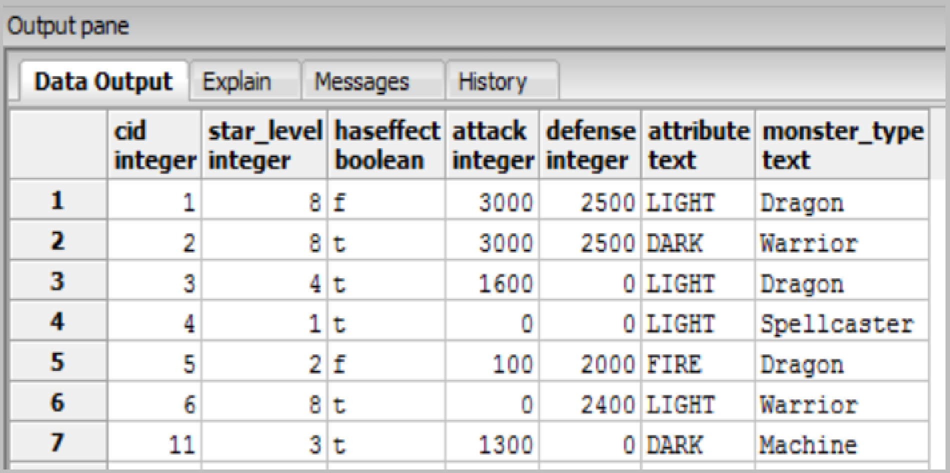
MonsterCard Table

A table that keeps track of specific information about monster cards.

```
CREATE TABLE MonsterCard
(
cid INT NOT NULL references Card(cid),
star_level INT NOT NULL,
hasEffect BOOLEAN NOT NULL,
attack INT NOT NULL,
defense INT NOT NULL,
attribute TEXT NOT NULL,
monster_type TEXT NOT NULL,
PRIMARY KEY(cid)
);
```

Dependencies: cid → star_level, hasEffect, attack, defense, attribute, monster_type

Sample Data:



Output pane

Data Output | Explain | Messages | History

	cid integer	star_level integer	haseffect boolean	attack integer	defense integer	attribute text	monster_type text
1	1	8	f	3000	2500	LIGHT	Dragon
2	2	8	t	3000	2500	DARK	Warrior
3	3	4	t	1600	0	LIGHT	Dragon
4	4	1	t	0	0	LIGHT	Spellcaster
5	5	2	f	100	2000	FIRE	Dragon
6	6	8	t	0	2400	LIGHT	Warrior
7	11	3	t	1300	0	DARK	Machine

Table: MonsterCard

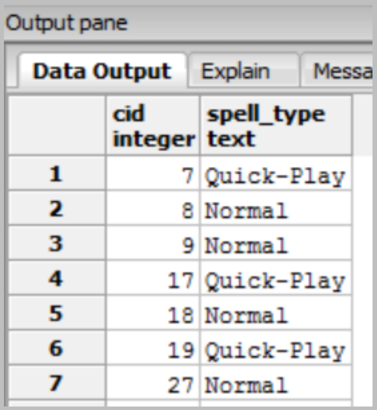
SpellCard Table

A table that keeps track of specific information about spell cards.

```
CREATE TABLE SpellCard
(
cid INT NOT NULL references Card(cid),
spell_type TEXT NOT NULL,
PRIMARY KEY(cid)
);
```

Dependencies: cid → spell_type

Sample Data:



	cid integer	spell_type text
1	7	Quick-Play
2	8	Normal
3	9	Normal
4	17	Quick-Play
5	18	Normal
6	19	Quick-Play
7	27	Normal

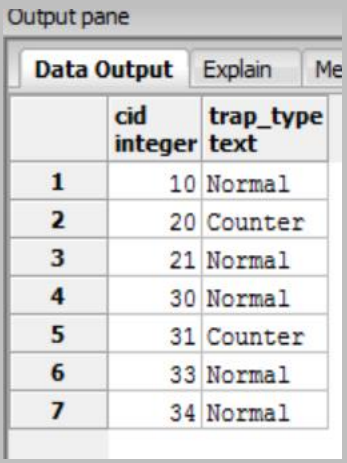
TrapCard Table

A table that keeps track of specific information about trap cards.

```
CREATE TABLE TrapCard
(
cid INT NOT NULL references Card(cid),
trap_type TEXT NOT NULL,
PRIMARY KEY(cid)
);
```

Dependencies: cid → trap_type

Sample Data:



	cid integer	trap_type text
1	10	Normal
2	20	Counter
3	21	Normal
4	30	Normal
5	31	Counter
6	33	Normal
7	34	Normal

CheckLegality Trigger

A trigger that checks if a new card placed in Deck_Card or SideDeck_Card has an acceptable value.

```
CREATE OR REPLACE FUNCTION checkLegality() RETURNS trigger AS
$$
DECLARE
    currentRecord text;
BEGIN
    FOR currentRecord IN SELECT legality FROM Card WHERE NEW.cid
= Card.cid LOOP
        IF  currentRecord = 'forbidden' THEN
            RAISE NOTICE 'Cid % is a forbidden card and cant be
used.',NEW.cid;
            RETURN NULL;
        END IF;
    END LOOP;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Check_Deck_Size Trigger

A trigger that checks if a new card can be placed in or removed from Deck_Card.

```
CREATE OR REPLACE FUNCTION check_deck_size() RETURNS trigger AS
$$
DECLARE
    --deckID integer;
    totalCards integer := 0;
    currentRecord record;

BEGIN
    --deckID := NEW.did;
    FOR currentRecord IN SELECT Deck_Card.qty FROM Deck_Card
WHERE NEW.did = Deck_Card.did LOOP
        totalCards := totalCards + currentRecord.qty;
    END LOOP;
    IF totalCards > 15 THEN
        RAISE NOTICE 'The new deck is too big. It has % cards.',
totalCards;
        RETURN NULL;
    ELSIF totalCards < 10 THEN
        RAISE NOTICE 'The new deck is too small. It has %
cards.', totalCards;
        RETURN NULL;
    ELSE

        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Check_Side_Deck_Size Trigger

A trigger that checks if a new card to be placed in SideDeck_Card has an acceptable value.

```
CREATE OR REPLACE FUNCTION check_side_deck_size() RETURNS
trigger AS
$$
DECLARE
    --deckID integer;
    totalCards integer := 0;
    currentRecord record;

BEGIN
    --deckID := NEW.did;
    FOR currentRecord IN SELECT SideDeck_Card.qty FROM
SideDeck_Card WHERE NEW.sdid = SideDeck_Card.sdid LOOP
        totalCards := totalCards + currentRecord.qty;
    END LOOP;
    IF totalCards > 5 THEN
        RAISE NOTICE 'The new side deck is too big. It has %
cards.', totalCards;
        RETURN NULL;
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Check_Dueling_Players Trigger

A trigger that checks to make sure that players are registered for a tournament they duel in, and are not dueling themselves.

```
CREATE OR REPLACE FUNCTION check_dueling_players() RETURNS
trigger AS

$$

BEGIN

    IF NEW.player_1_pid = NEW.player_2_pid THEN

        RAISE NOTICE 'A player cannot duel his or herself!';

        RETURN NULL;

    ELSIF NEW.player_1_pid NOT IN(SELECT pid FROM Registration
WHERE NEW.tid = Registration.tid) THEN

        RAISE NOTICE 'Player 1 is not registered for that
tournament.';

        RETURN NULL;

    ELSIF NEW.player_2_pid NOT IN(SELECT pid FROM Registration
WHERE NEW.tid = Registration.tid) THEN

        RAISE NOTICE 'Player 2 is not registered for that
tournament.';

        RETURN NULL;

    ELSE

        RETURN NEW;

    END IF;

END;

$$ LANGUAGE plpgsql;
```

Check_Card_Type_Monster Trigger

A trigger that checks if a card is already a spell or trap card before adding it as a monster card.

```
CREATE OR REPLACE FUNCTION check_card_type_monster() RETURNS
trigger AS
$$
DECLARE
    multitypeCards integer;
BEGIN
    SELECT count(*) INTO multitypeCards FROM SpellCard, TrapCard
    WHERE NEW.cid = SpellCard.cid
    OR NEW.cid = TrapCard.cid;

    --RAISE NOTICE 'multitype cards: %', multitypeCards;
    IF multitypeCards > 0 THEN
        RAISE NOTICE 'Card is already a trap or spell.';
        RETURN NULL;
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

- **NOTE: Two very similar triggers perform the same functionality for monster cards and trap cards, and were omitted.**

Monster/Spell/TrapCardView Views

Views that consolidate all monster/spell/trap card information.

```
CREATE VIEW MonsterCardView AS
    SELECT
Card.cid,card_name,flavor_text,legality,star_level,hasEffect,att
ack,defense,attribute,monster_type
    FROM Card, MonsterCard
    WHERE MonsterCard.cid = Card.cid;

CREATE VIEW SpellCardView AS
    SELECT Card.cid,card_name,flavor_text,legality,spell_type
    FROM Card, SpellCard
    WHERE SpellCard.cid = Card.cid;

CREATE VIEW TrapCardView AS
    SELECT Card.cid,card_name,flavor_text,legality,trap_type
    FROM Card, TrapCard
    WHERE TrapCard.cid = Card.cid;
```

Stored Procedure getCardsInDeck(integer)

Given a deck's did, the procedure returns what cards are in the deck.

```
 CREATE OR REPLACE FUNCTION getCardsInDeck(integer) RETURNS
TABLE(card_name TEXT, qty INTEGER) AS
$$
DECLARE
    deckID ALIAS FOR $1;
BEGIN
     RETURN QUERY
     SELECT Card.card_name, Deck_Card.qty
     FROM Card, Deck, Deck_Card
     WHERE Card.cid = Deck_Card.cid
     AND Deck.did = Deck_Card.did
     AND Deck.did = deckID;
END;
$$ LANGUAGE plpgsql;
```

- **NOTE: One very similar procedure performs the same functionality for side decks.**

Stored Procedure getTournaments(integer)

Given a player's pid, the procedure returns what tournaments the players have signed up for.

```
 CREATE OR REPLACE FUNCTION getTournaments(integer) RETURNS
TABLE(tournament_name TEXT) AS

$$

DECLARE

    playerID ALIAS FOR $1;

BEGIN

     RETURN QUERY

     SELECT Tournament.tournament_name

     FROM Tournament, Player, Registration

     WHERE Tournament.tid = Registration.tid

     AND Player.pid = Registration.pid

     AND Player.pid = playerID;

END;

$$ LANGUAGE plpgsql;
```

Example Reports

- **Sample report for balancing—see what cards are most used in professional decks.**

```
 SELECT card_name, count(card_name) AS occurences
FROM Deck, Card, Deck_Card
WHERE Deck.did = Deck_Card.did
AND Deck_Card.cid = Card.cid
GROUP BY card_name
ORDER BY occurences DESC;
```

- **Sample report for running tournament—see all the players that have registered for a tournament.**

```
SELECT player_name
FROM Player, Registration,Tournament
WHERE Player.pid = Registration.pid
AND Tournament.tid = Registration.tid
AND Tournament.tid = <<insert tid here>>
```

Roles

The database currently supports three kinds of roles: Admin, CheckIn, and Judge.

```
CREATE ROLE CheckIn;
CREATE ROLE Admin;
CREATE ROLE Judge;
```

- **Admin: Has administrative power over the entire database.**

```
GRANT SELECT, INSERT, UPDATE, DELETE ON Duel TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Player TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Registration TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Tournament TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Places TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Runs TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON SideDeck TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON SideDeck_Card TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Venue TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Deck TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Deck_Card TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Card TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON MonsterCard TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON SpellCard TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON TrapCard TO Admin;
```

- **CheckIn: Has the power to add, remove, or register players for a given tournament.**

```
REVOKE ALL PRIVILEGES ON Duel FROM CheckIn;
REVOKE ALL PRIVILEGES ON Player FROM CheckIn;
REVOKE ALL PRIVILEGES ON Registration FROM CheckIn;
REVOKE ALL PRIVILEGES ON Tournament FROM CheckIn;
REVOKE ALL PRIVILEGES ON Places FROM CheckIn;
REVOKE ALL PRIVILEGES ON Runs FROM CheckIn;
REVOKE ALL PRIVILEGES ON SideDeck FROM CheckIn;
REVOKE ALL PRIVILEGES ON SideDeck_Card FROM CheckIn;
REVOKE ALL PRIVILEGES ON Venue FROM CheckIn;
REVOKE ALL PRIVILEGES ON Deck FROM CheckIn;
REVOKE ALL PRIVILEGES ON Deck_Card FROM CheckIn;
REVOKE ALL PRIVILEGES ON Card FROM CheckIn;
REVOKE ALL PRIVILEGES ON MonsterCard FROM CheckIn;
REVOKE ALL PRIVILEGES ON SpellCard FROM CheckIn;
REVOKE ALL PRIVILEGES ON TrapCard FROM CheckIn;

GRANT SELECT, INSERT, UPDATE, DELETE ON Registration TO CheckIn;
GRANT SELECT, INSERT, UPDATE, DELETE ON Player TO CheckIn;
GRANT SELECT ON Tournament TO CheckIn;
```

- **Judge: Has the final say about the outcome of a duel. Can disqualify a player for cheating or poor conduct.**

```
REVOKE ALL PRIVILEGES ON Duel FROM Judge;
REVOKE ALL PRIVILEGES ON Player FROM Judge;
REVOKE ALL PRIVILEGES ON Registration FROM Judge;
REVOKE ALL PRIVILEGES ON Tournament FROM Judge;
REVOKE ALL PRIVILEGES ON Places FROM Judge;
REVOKE ALL PRIVILEGES ON Runs FROM Judge;
REVOKE ALL PRIVILEGES ON SideDeck FROM Judge;
REVOKE ALL PRIVILEGES ON SideDeck_Card FROM Judge;
REVOKE ALL PRIVILEGES ON Venue FROM Judge;
REVOKE ALL PRIVILEGES ON Deck FROM Judge;
REVOKE ALL PRIVILEGES ON Deck_Card FROM Judge;
REVOKE ALL PRIVILEGES ON Card FROM Judge;
REVOKE ALL PRIVILEGES ON MonsterCard FROM Judge;
REVOKE ALL PRIVILEGES ON SpellCard FROM Judge;
REVOKE ALL PRIVILEGES ON TrapCard FROM Judge;

GRANT SELECT, INSERT, UPDATE, DELETE ON Duel TO Judge;
GRANT SELECT, DELETE ON Player TO Judge;
GRANT SELECT ON Registration TO Judge;
GRANT SELECT, DELETE ON Runs TO Judge;
GRANT SELECT ON SideDeck TO Judge;
GRANT SELECT ON SideDeck_Card TO Judge;
GRANT SELECT ON Deck TO Judge;
GRANT SELECT ON Deck_Card TO Judge;
GRANT SELECT ON Card TO Judge;
```

Implementation Notes

- **For simplicity, decks and side decks in the database were defined as being between 10 and 15 and 0 and 5 cards respectively. In official YuGiOh rules, decks consist of 40-60 cards, and side decks are 0-15 cards.**

- **The SideDeck table is necessary, even though it is only a primary key. It is needed as a way to refer to the group of 15 cards that it is made of.**

Known Problems

- **Another view might be helpful to make tournament info easier to access.**

Future Enhancements

- **Account for Pendulum-type monsters (both spell and monster types).**
- **Create trigger that flags cards with an extremely high deck inclusion rate.**