YugiohDB

By Graham Burek

April 26, 2016
Table of Contents

Table of contents ........................................................................................................................................... 2
Executive Summary .................................................................................................................................................. 3
E/R Diagram ......................................................................................................................................................... 4
Tables .................................................................................................................................................................. 5-19
Triggers ............................................................................................................................................................... 20-25
Views .................................................................................................................................................................. 26
Stored Procedures ............................................................................................................................................... 27-28
Example Reports ............................................................................................................................................... 29
Roles .................................................................................................................................................................. 30-32
Implementation Notes ....................................................................................................................................... 33
Known Problems ............................................................................................................................................... 34
Future Enhancements ....................................................................................................................................... 35
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.
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:
**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:**

![Sample Data](https://example.com/sample_data.png)
**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 $\rightarrow$ venue_name, address_1, 2, 3, ZIP

**Sample Data:**

![Sample Data Table](image)
**SideDeck Table**

A representative table for a side deck.

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

**Dependencies:** sdid →

**Sample Data:**

[Table: SideDeck]

<table>
<thead>
<tr>
<th>sdid</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
**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 $\rightarrow$ sdid, deck_name

**Sample Data:**

```
<table>
<thead>
<tr>
<th>did</th>
<th>sdid</th>
<th>deck_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Blue-Eyes Turbo</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Artifact Monarchs</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>&lt;NULL&gt;</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Dark Magic Deck</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>&lt;NULL&gt;</td>
</tr>
</tbody>
</table>
```
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 \( \rightarrow \) tournament_name, date, vid

Sample Data:

<table>
<thead>
<tr>
<th>tid</th>
<th>tournament_name text</th>
<th>tournament_date</th>
<th>vid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YCS Regional Qualifier Chicago</td>
<td>2016-09-20</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>EVO 2016</td>
<td>2016-07-15</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>LLDS 2016</td>
<td>2016-11-01</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Ultimate Duelist Series</td>
<td>2017-10-22</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>TCG World Championship</td>
<td>2017-06-21</td>
<td>3</td>
</tr>
</tbody>
</table>
**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:

![Sample Data Table](image)
**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:**

![Sample Data Table](image-url)
**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) \(\rightarrow\) did

**Sample Data:**

![Table: Runs](image)
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 \(\rightarrow\) card_name, flavor_text, legality

Sample Data:

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

**Sample Data:**

![Sample Data Table](image-url)
**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:** $(did,cid) \rightarrow qty$

**Sample Data:**

![Sample Data Table]

*Table: Deck_Card*
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 \rightarrow star_level, hasEffect, attack, defense, attribute, monster_type

Sample Data:

<table>
<thead>
<tr>
<th>cid</th>
<th>star_level</th>
<th>hasEffect</th>
<th>attack</th>
<th>defense</th>
<th>attribute</th>
<th>monster_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>f</td>
<td>3000</td>
<td>2500</td>
<td>LIGHT</td>
<td>Dragon</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>f</td>
<td>3000</td>
<td>2500</td>
<td>DARK</td>
<td>Warrior</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>f</td>
<td>1600</td>
<td>0</td>
<td>LIGHT</td>
<td>Dragon</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>t</td>
<td>0</td>
<td>0</td>
<td>LIGHT</td>
<td>Spellcaster</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>f</td>
<td>100</td>
<td>2000</td>
<td>FIRE</td>
<td>Dragon</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>t</td>
<td>0</td>
<td>2400</td>
<td>LIGHT</td>
<td>Warrior</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>t</td>
<td>1300</td>
<td>0</td>
<td>DARK</td>
<td>Machine</td>
</tr>
</tbody>
</table>
**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:**

Table: SpellCard

<table>
<thead>
<tr>
<th>cid</th>
<th>spell_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7 Quick-Play</td>
</tr>
<tr>
<td>2</td>
<td>8 Normal</td>
</tr>
<tr>
<td>3</td>
<td>9 Normal</td>
</tr>
<tr>
<td>4</td>
<td>17 Quick-Play</td>
</tr>
<tr>
<td>5</td>
<td>18 Normal</td>
</tr>
<tr>
<td>6</td>
<td>19 Quick-Play</td>
</tr>
<tr>
<td>7</td>
<td>27 Normal</td>
</tr>
</tbody>
</table>
**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:**

![Sample Data Table](image)

**Table:** TrapCard
**CheckLegality Trigger**

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

```sql
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 $$
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;

Triggers: Check_Dueling_Players
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,attack,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.

```sql
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;
```

*Stored Procedures: getTournaments()*
Example Reports

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

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