SportStock: Database Application

Arpan Bhattacharya

INFO 605 Summer 2013

Final Term Project

August 28th, 2013

Table of Contents

==========

I certify that:

- **This paper/project/exam is entirely my own work.**
- **I have not quoted the words of any other person from a printed source or a website without indicating what has been quoted and providing an appropriate citation.**
- **I have not submitted this paper / project to satisfy the requirements of any other course.**

Signature___Arpan Bhattacharya_____
Date ___8/28/2013_____

==========

I. Problem Domain: SportStock

The problem domain will be a mobile application called SportStock that allows users to purchase different propositional contracts based on live sporting events. Presented with two associated contracts for each sporting event, the user will have the option to purchase either side. If the user purchases the correct contract, they will be rewarded $0.50 for each owned contract. If they pick the wrong contract, the cost of the contracts will be deducted from their account balance.

The user experience of this application will be the following, a mock-up can be found in the appendix (p.28):

If the Orioles are playing the Yankees and the Yankees are winning 3-2 in 9th inning, a user could login to the application and find the following open contracts and their associated cost:

Yankees v. Orioles (3-2)

1. Yankees win the game $1.50
2. Orioles win the game $0.50

If the user purchases 2 contracts of proposition 1 "Yankees win the game" and the Yankees do indeed win the game, their account would be credited $4, composed of a $3 contract premium and $1 for the contract reward. If the Yankees lose, then their account would be debited $3.00.

All live sporting events are modeled as these associated binary propositional pairs, i.e. out of the two associated contracts only one can represent the correct outcome. An algorithm will adjust the cost of each contract associated with the proposition based on the total quantity purchased on each side. A contract will always be worth between $0.01 and $1.99 dollars and the sum of the associated contract prices must always equal $2.00. Users will only be able to purchase 10 contracts at a time in order to create an elastic market price.

Since this is a proof of concept, a dollar amount has been chosen to represent the value of each contract. Due to federal laws prohibiting sports gambling, if this application were to be implemented, a point system would be used to replace the dollar amount. The user would sign up for free and start off with a certain set of points which they could then use to compete with other players. In addition, instead of just offering contracts based on the overall outcome of games, contracts based on individual player performance and specific events within the game could also be offered.

II. Requirements

The user will have a unique user ID, a username, a password, an email, and an account balance. A record of the account balance must be maintained for each user along with the corresponding transaction. For each transaction created by the user, the date, the time, and a unique transaction ID is required. The transaction should also record the contracts bought, the price they were bought at, and the quantity purchased.

For each event, we will want to classify the type of sport, a unique event ID, the date of the event, and time of the event. Each event can have many propositions, which will consist of a unique proposition ID and a description. Each proposition will be composed of two contracts. Each contract will have a unique contract ID, a description of the outcome, and the status of the outcome ('WIN, 'LOSS', or 'OPEN'). In addition, each contract will need a history of its price as it changes over time.

The following data management functions (based on the UX mockup displayed in the Appendix on p.28) are required:

- Buy Contracts
    - Display all open contracts for different events.
    - Create a transaction for a user purchasing a number of contracts.
    - Update the user's account balance based on the transaction.
- Check Open Contracts
    - Display all open contracts a user owns.
    - Display the price history of a contract (price versus time stock chart).
    - Update the user's account balance when open contracts are closed.
    - Update the price of a contract.
- User History
    - Display all transactions made by a user, sort newest to oldest.
    - Display the account balance based on each transaction for a user.
    - Create a pair of contracts for a proposition for an event.

III. ER Model

SportsStock

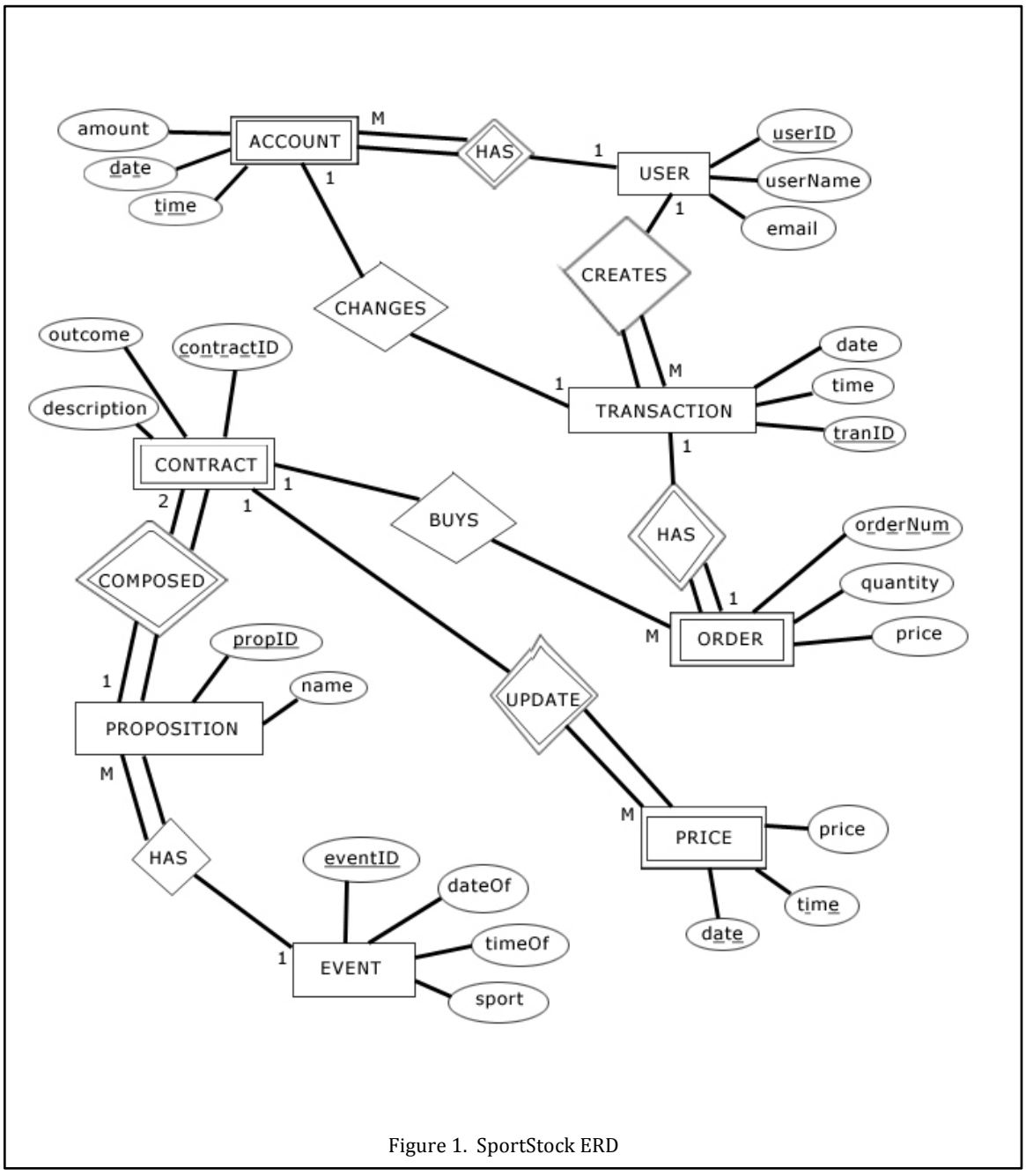
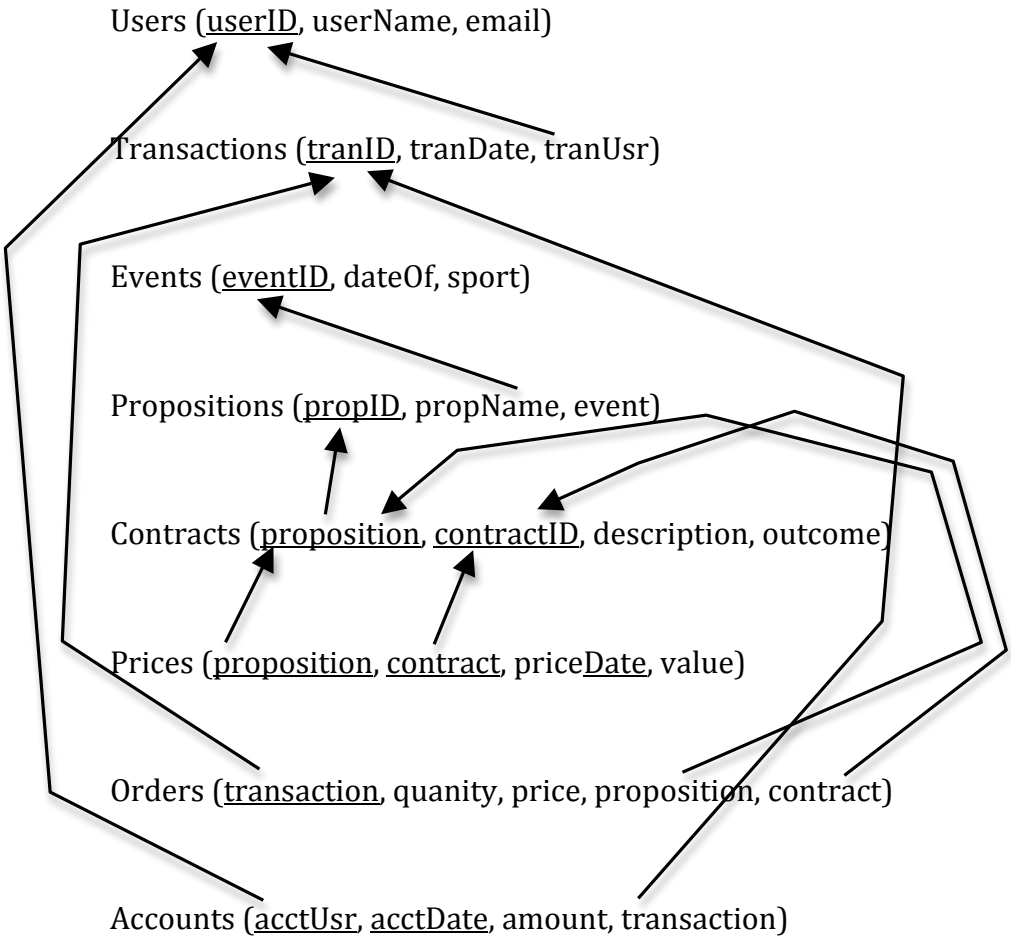


Figure 1. SportStock ERD

IV. Relational Schema



The ERD, displayed in Figure 1, was mapped with total/partial translation. The translation was verified to meet 3NF criteria. All attributes in each table are functionally dependent on the PK.

OrderNum was not required for the Orders table since there is a 1 to 1 relationship with the Transasctions table and the tranID can serve as the FK PK. Since the DATE datatype can store the date and the time to the second, the separate time attribute was eliminated from the Transaction, Event, Price, and Account tables. This should suffice for a fairly dynamic user experience where transactions can be occur every second. If more granularity is required, a TIMESTAMP datatype could be used to make transactions by the fractional second.

V. Data Dictionaries

Users: Contains information about each person playing the game.						
Attribute Name	Description	Datatype	Domain	Nullable	PK	FK
userID	Unique marker used to identify specific user	Number(8)	0 -- 99999999	No	Yes	No
userName	The handle the user chooses to use	Varchar2(12)	All	No	No	No
email	The user's email address	Varchar(20)	Must contain '@'	No	No	No

Transactions: Contains information about every transaction that affects the user's account balance.						
Attribute Name	Description	Datatype	Domain	Nullable	PK	FK
tranID	Unique marker used to identify specific transactions	Number(8)	0 -- 99999999	No	Yes	No
tranDate	The date and time of the transaction	Date	All	No	No	No
tranUsr	The userID associated with the transaction	Number(8)	0 -- 99999999	Yes	No	Yes

Events: Contains information about all events, which propositional contracts are based on.						
Attribute Name	Description	Datatype	Domain	Nullable	PK	FK
eventID	Unique marker used to identify specific events	Number(8)	0 -- 99999999	No	Yes	No
dateOf	The date and time of the event	Date	All	No	No	No
sport	The name of the sporting event	Varchar2(140)	All	No	No	No

Propositions: Contains information about different propositions, which can be made for different events.						
Attribute Name	Description	Datatype	Domain	Nullable	PK	FK
propID	Unique marker used to identify specific propositions	Number(8)	0 -- 99999999	No	Yes	No
propName	The name of the proposition	Varchar2(140)	All	No	No	No
event	The event the proposition is associated with	Number(8)	0 -- 99999999	No	No	Yes

Contracts: Contains information detailing both sides of the proposition.						
Attribute Name	Description	Datatype	Domain	Nullable	PK	FK
proposition	The propID associated with the contract	Number(8)	0 -- 99999999	No	Yes	Yes
contractID	Unique marker used to identify a specific contract	Number(8)	0 -- 99999999	No	Yes	No
description	The details of one side of the proposition	Varchar2(140)	All	No	No	No
outcome	Determines the status of the contract	Varchar2(4)	“WIN”, “LOSS”, “OPEN”	No	No	No

Prices: Contains the price history of a specific contract.						
Attribute Name	Description	Datatype	Domain	Nullable	PK	FK
proposition	The propID for the contract	Number(8)	0 -- 99999999	No	Yes	Yes
contract	The contractID associated with the price	Number(8)	0 -- 99999999	No	Yes	Yes
priceDate	The date and time for the current price	Date	All	No	Yes	No
value	The amount each contract costs at this time	Number(3,2)	0.01 -- 1.99	No	No	No

Orders: Contains details about each transaction.						
Attribute Name	Description	Datatype	Domain	Nullable	PK	FK
transaction	The tranID associated with the order	Number(8)	0 -- 99999999	No	Yes	Yes
quanitity	The amount of contracts ordered	Number(2)	1 -- 10	No	No	No
price	The price the order was filled at	Number(3,2)	0.01 – 1.99	No	No	No
Proposition	The propID for the contract	Number(8)	0 -- 99999999	No	No	Yes
Contract	The contractID associated with the order	Number(8)	0 -- 99999999	No	No	Yes

Accounts: Contains the account balance history for each user.						
Attribute Name	Description	Datatype	Domain	Nullable	PK	FK
acctUsr	The userID associated with the account	Number(8)	0 -- 99999999	No	Yes	Yes
acctDate	The date and time associated with the account balance	Date	All	No	Yes	No
amount	The amount of money in the user's balance at this specific moment in time	Number(5,2)	0 -- 999.99	No	No	No
transaction	The tranID associated with this account update	Number(8)	0 -- 99999999	Yes	No	Yes

VI. DDL

```
CREATE TABLE Users
(
  userID NUMBER(8) CONSTRAINT user_pk PRIMARY KEY,
  userName VARCHAR2(12) CONSTRAINT user_nn_usrname NOT NULL
    CONSTRAINT user_uq_username UNIQUE,
  email VARCHAR2(20) CONSTRAINT user_nn_email NOT NULL
    CONSTRAINT user_ck_email CHECK (email LIKE ('%@%'))
)
```

```
CREATE TABLE Transactions
(
  tranID NUMBER(8) CONSTRAINT tran_pk PRIMARY KEY,
  trandate DATE CONSTRAINT tran_nn_date NOT NULL,
  tranusr NUMBER(8)
    CONSTRAINT tran_nn_user NOT NULL
    CONSTRAINT tran_fk_user REFERENCES Users(userID)
)
```

```
CREATE TABLE Events
(
  eventID NUMBER(8) CONSTRAINT events_pk PRIMARY KEY,
  dateOf DATE CONSTRAINT events_nn_date NOT NULL,
  sport VARCHAR2(140) CONSTRAINT events_nn_sport NOT NULL
)
```

```
CREATE TABLE Propositions
(
  propID NUMBER(8) CONSTRAINT prop_pk PRIMARY KEY,
  propName VARCHAR2(140) CONSTRAINT prop_nn_name NOT NULL,
  event NUMBER(8) CONSTRAINT prop_nn_event NOT NULL
    CONSTRAINT prop_fk_event REFERENCES Events(eventID)
)
```

```
CREATE TABLE Contracts
(
  proposition NUMBER(8)
    CONSTRAINT contract_fk_prop REFERENCES Propositions(propID),
  contractID NUMBER(8),
  CONSTRAINT contract_pk
  PRIMARY KEY (proposition, contractID),
  description VARCHAR2(140) CONSTRAINT contract_nn_desc NOT NULL,
  outcome VARCHAR2(24) CONSTRAINT contract_nn_outcome NOT NULL,
    CONSTRAINT contract_ck_outcome
    CHECK (outcome IN ('WIN', 'LOSS', 'OPEN'))
)
```

```
CREATE TABLE Prices
(
  proposition NUMBER(8),
  contract NUMBER(8),
  priceDate DATE,
  CONSTRAINT price_pk PRIMARY KEY (proposition, contract, priceDate),
  CONSTRAINT price_fk_contract FOREIGN KEY (proposition, contract)
    REFERENCES Contracts(proposition, contractID),
  value NUMBER(3,2) CONSTRAINT price_nn_price NOT NULL
    CONSTRAINT price_ck_value CHECK (value BETWEEN 0.01 AND 1.99)
)
```

```
CREATE TABLE Orders
(
  transaction NUMBER(8) CONSTRAINT order_fk_tran
    REFERENCES Transactions(tranID)
    CONSTRAINT order_pk_tran PRIMARY KEY,
  quantity NUMBER(2) CONSTRAINT order_nn_quantity NOT NULL,
  price NUMBER(3,2) CONSTRAINT order_nn_price NOT NULL
    CONSTRAINT order_ck_price
    CHECK (price BETWEEN 0.01 AND 1.99),
  proposition NUMBER(8) CONSTRAINT order_nn_prop NOT NULL,
  contract NUMBER(8) CONSTRAINT order_nn_contract NOT NULL,
  CONSTRAINT order_fk_contract FOREIGN KEY (proposition, contract)
    REFERENCES Contracts(proposition, contractID)
)
```

```
CREATE TABLE Accounts
 (
  acctUsr NUMBER(8) CONSTRAINT account_fk_user REFERENCES Users(userID),
  acctDate DATE,
  CONSTRAINT account_pk PRIMARY KEY (acctUsr, acctDate),
  amount NUMBER(5,2) CONSTRAINT account_nn_amt NOT NULL,
  transaction NUMBER(8) CONSTRAINT account_fk_tran
    REFERENCES Transactions(tranID)
)
```

VII. DML

Table: Users

Adding new user accounts (Johny, David, Frank, and Susan)

```
INSERT INTO Users (userID, userName, email) VALUES (1, 'johnny', 'johnny@gmailcom');
INSERT INTO Users (userID, userName, email) VALUES (2, 'david', 'david@gmail.com');
INSERT INTO Users (userID, userName, email) VALUES (3, 'frank', 'frank@gmail.com');
INSERT INTO Users (userID, userName, email) VALUES (4, 'susan', 'susan@gmail.com');
```

Updating a user's email address (Frank changed his email)

```
UPDATE Users
  SET email = 'frank@yahoo.com'
WHERE userID = 3;
```

Susan deleted her account

```
DELETE FROM Users
  WHERE userID = 4;
```

Table: Transactions

Adding a new transaction

```
/* Johnny makes a transaction */
INSERT INTO Transactions (tranID, tranDate, tranUsr)
VALUES (1, to_date('01-MAY-2014 15:05:00', 'dd-mon-yyyy hh24:mi:ss'), 1);

/* Frank makes a transaction at the same time */
INSERT INTO Transactions (tranID, tranDate, tranUsr)
VALUES (2, to_date('01-MAY-2014 15:05:00', 'dd-mon-yyyy hh24:mi:ss'), 3);

/* Johnny makes another transaction 5 minutes later */
INSERT INTO Transactions (tranID, tranDate, tranUsr)
VALUES (3, to_date('01-MAY-2014 15:10:00', 'dd-mon-yyyy hh24:mi:ss'), 1);

/* Frank makes another transaction */
INSERT INTO Transactions (tranID, tranDate, tranUsr)
VALUES (4, to_date('01-MAY-2014 15:11:00', 'dd-mon-yyyy hh24:mi:ss'), 3);

/* David makes a transaction at the same time*/
INSERT INTO Transactions (tranID, tranDate, tranUsr)
VALUES (5, to_date('01-MAY-2014 15:11:00', 'dd-mon-yyyy hh24:mi:ss'), 2);

/* David makes another transaction 3 seconds later */
INSERT INTO Transactions (tranID, tranDate, tranUsr)
VALUES (6, to_date('01-MAY-2014 15:11:03', 'dd-mon-yyyy hh24:mi:ss'), 2);
```

Transactions will never be modified or removed since they keep a log of all orders and account adjustments

Table: Events

Adding a few events

```
INSERT INTO Events (eventID, dateOf, sport)
VALUES (1, to_date('01-MAY-2014 15:00:00', 'dd-mon-yyyy hh24:mi:ss'), 'YANKEES VS ORIOLES');

INSERT INTO Events (eventID, dateOf, sport)
VALUES (2, to_date('02-MAY-2014 18:00:00', 'dd-mon-yyyy hh24:mi:ss'), 'PHILLIES VS PIRATES');

INSERT INTO Events (eventID, dateOf, sport)
VALUES (3, to_date('05-MAY-2014 18:00:00', 'dd-mon-yyyy hh24:mi:ss'), 'PACKERS VS GIANTS');

INSERT INTO Events (eventID, dateOf, sport)
VALUES (4, to_date('11-MAY-2014 20:00:00', 'dd-mon-yyyy hh24:mi:ss'), 'RAIDERS VS BRONCOS');
```

Modifying an event

```
/* Phillies Pirates game got rained out and postponed to the following day */
UPDATE Events
  SET dateOf = to_date('03-MAY-2014 18:00:00', 'dd-mon-yyyy hh24:mi:ss')
WHERE eventID = 2;
```

Removing an event (rare but could happen)

```
/* There are rumors that say that the Raiders vs. Broncos game is fixed.  We don't want to offer any
propositions on this game and we want to remove it from our events */
DELETE FROM Events
  WHERE eventID = 4;
```

Table: Propositions

Adding a few propositions based on the ‘Yankees vs. Orioles’, ‘Phillies vs. Pirates’, and ‘Packers vs. Giants’ events

---

```
/* Adding the winning team proposition to the Yankees vs. Orioles event */
INSERT INTO Propositions (propID, propName, event)
VALUES (1, 'WINNER', 1);

/* Adding the total score proposition to the Yankees vs. Orioles event */
INSERT INTO Propositions (propID, propName, event)
VALUES (2, 'TOTAL SCORE', 1);

/* Adding the total hits proposition to the Phillies vs. Pirates event */
INSERT INTO Propositions (propID, propName, event)
VALUES (3, 'TOTAL HITS', 2);

/* Adding the total score proposition to the Packers vs. Giants event */
INSERT INTO Propositions (propID, propName, event)
VALUES (4, 'TOTAL SCORE', 3);
```

Removing the ‘Total Score’ proposition for Yankees vs. Orioles

---

```
DELETE FROM Propositions
  WHERE propID = 2;
```

Updating the ‘Total Score’ proposition for the Packers vs. Giants to ‘First to Score’

```
UPDATE Propositions
  SET propName = 'FIRST TO SCORE'
WHERE propID = 4;
```

Table: Contracts

Adding associated contracts based on the each proposition

```
/* Yankees vs. Orioles Winning Team */

INSERT INTO Contracts (proposition, contractID, description, outcome)
VALUES (1, 1, 'YANKEES WIN', 'OPEN');

INSERT INTO Contracts (proposition, contractID, description, outcome)
VALUES (1, 2, 'ORIOLES WIN', 'OPEN');

/* Phillies vs. Pirates Total Hits */

INSERT INTO Contracts (proposition, contractID, description, outcome)
VALUES (3, 1, 'TOTAL HITS GREATER THAN 7.5', 'OPEN');

INSERT INTO Contracts (proposition, contractID, description, outcome)
VALUES (3, 2, 'TOTAL HITS LESS THAN 7.5', 'OPEN');

/* Packers vs. Giants First to Score */

INSERT INTO Contracts (proposition, contractID, description, outcome)
VALUES (4, 1, 'PACKERS SCORE FIRST', 'OPEN');

INSERT INTO Contracts (proposition, contractID, description, outcome)
VALUES (4, 2, 'GIANTS SCORE FIRST', 'OPEN');
```

Updating the status of a contract once there is an outcome (Yankees Won!)

```
UPDATE Contracts
  SET outcome = 'WIN' WHERE proposition = 1 AND contractID = 1;
UPDATE Contracts
  SET outcome = 'LOSS' WHERE proposition = 1 AND contractID = 2;
```

Table: Prices

This table stores the price of each contract. The price will be updated based on an algorithm (which is detailed in the Appendix (p.20). The prices of associated contracts will always be updated at the same time. A record of the historical prices of each contract is needed, thus this table will never need any rows modified or removed.

```
/* Proposition and associated contracts initially created, 5 minutes before the event starts */

/* Prices start at $1 for both contracts @ 01-MAY-2014 14:55:00 */

INSERT INTO Prices (proposition, contract, priceDate, value)

VALUES (1, 1, to_date('01-MAY-2014 14:55:00', 'dd-mon-yyyy hh24:mi:ss'), 1.00);

INSERT INTO Prices (proposition, contract, priceDate, value)

VALUES (1, 2, to_date('01-MAY-2014 14:55:00', 'dd-mon-yyyy hh24:mi:ss'), 1.00);

/* Price update @ 01-MAY-2014 14:55:30 */

INSERT INTO Prices (proposition, contract, priceDate, value)

VALUES (1, 1, to_date('01-MAY-2014 14:55:30', 'dd-mon-yyyy hh24:mi:ss'), 1.30);

INSERT INTO Prices (proposition, contract, priceDate, value)

VALUES (1, 2, to_date('01-MAY-2014 14:55:30', 'dd-mon-yyyy hh24:mi:ss'), 0.70);

/* Price update @ 01-MAY-2014 14:56:15 */

INSERT INTO Prices (proposition, contract, priceDate, value)

VALUES (1, 1, to_date('01-MAY-2014 14:56:15', 'dd-mon-yyyy hh24:mi:ss'), 1.00);

INSERT INTO Prices (proposition, contract, priceDate, value)

VALUES (1, 2, to_date('01-MAY-2014 14:56:15', 'dd-mon-yyyy hh24:mi:ss'), 1.00);

/* Price update @ 01-MAY-2014 14:56:17*/

INSERT INTO Prices (proposition, contract, priceDate, value)

VALUES (1, 1, to_date('01-MAY-2014 14:56:17', 'dd-mon-yyyy hh24:mi:ss'), 0.86);

INSERT INTO Prices (proposition, contract, priceDate, value)

VALUES (1, 2, to_date('01-MAY-2014 14:56:17', 'dd-mon-yyyy hh24:mi:ss'), 0.14);
```

Table: Orders

Orders coming in on the Yankees vs. Orioles Winner Propositional Contracts

---

```
/* First set of 100 orders at initial offering of $1.00 */
INSERT INTO Orders (transaction, quantity, price, proposition, contract)
VALUES (1, 65, 1.00, 1, 1);
INSERT INTO Orders (transaction, quantity, price, proposition, contract)
VALUES (2, 35, 1.00, 1, 2);

/* Second set of orders at new adjusted price of $1.30 and $0.70 */
INSERT INTO Orders (transaction, quantity, price, proposition, contract)
VALUES (3, 50, 1.30, 1, 1);
INSERT INTO Orders (transaction, quantity, price, proposition, contract)
VALUES (4, 50, 0.70, 1, 2);

/* Third set of orders at new adjusted price of $1.00 and $1.00 */
INSERT INTO Orders (transaction, quantity, price, proposition, contract)
VALUES (5, 43, 1.00, 1, 1);
INSERT INTO Orders (transaction, quantity, price, proposition, contract)
VALUES (6, 57, 1.00, 1, 2);

/* Orders won't come in exactly at 100 contract blocks but that is what this example is showing. */
/* There will be a max order amount per transaction per user to facilitate an elastic market price.*/
```

An order will never need to be modified or deleted since it is mapped 1 to 1 with transactions and all transactions are saved.

Table: Accounts

Account creation for Johnny and Frank with an initial balance of $200

```
/*  No tranID for non-transaction updates.  Johnny's account created with initial $200 balance */
INSERT INTO Accounts (acctUsr, acctDate, amount)
VALUES (1, to_date('01-MAY-2014 00:01:00', 'dd-mon-yyyy hh24:mi:ss'), 200.00);
/* Frank's account created with initial $200 balance */
INSERT INTO Accounts (acctUsr, acctDate, amount)
VALUES (3, to_date('01-MAY-2014 00:01:15', 'dd-mon-yyyy hh24:mi:ss'), 200.00);
```

Accounts will never be modified or deleted since they keep a history of the balance. Johnny and Frank create transactions, which creates an account update.

```
/* Update Johnny's account balance based on his first transaction.  $200 - $65 = $135 */
INSERT INTO Accounts (acctUsr, acctDate, amount, transaction)
VALUES (1, to_date('01-MAY-2014 15:05:00', 'dd-mon-yyyy hh24:mi:ss'), 135.00, 1);
/* Update Frank's account balance based on his first transaction. $200 - $35 = $165  */
INSERT INTO Accounts (acctUsr, acctDate, amount, transaction)
VALUES (3, to_date('01-MAY-2014 15:05:00', 'dd-mon-yyyy hh24:mi:ss'), 165.00, 2);
/* Update Johnny's account balance based on his second transaction. $135 - $65 = $70  */
INSERT INTO Accounts (acctUsr, acctDate, amount, transaction)
VALUES (1, to_date('01-MAY-2014 15:10:00', 'dd-mon-yyyy hh24:mi:ss'), 70.00, 3);
```

After the Yankees win, Johnny gets paid out on his contracts

```
/* After Johnny's transactions he owned a total of 115 contracts at an average price of $1.13. */
/* This allows Johnny to win $57.5 contract reward, and $130 in contract premium, totaling $187.50. */
/* Johnny's last balance was $70.  So his new balance will be $70 + $187.50 = $257.5 */
INSERT INTO Accounts (acctUsr, acctDate, amount)
VALUES (1, to_date('01-MAY-2014 18:00:00', 'dd-mon-yyyy hh24:mi:ss'), 257.50);
```

VIII. Queries

Display all open propositional contracts for each associated event:

```
SELECT sport AS Game,
       propName AS Proposition,
       description AS Options,
       outcome AS Status
FROM Contracts
  JOIN Propositions
  ON propID = contracts.proposition
  JOIN Events
  ON eventID = Propositions.event
WHERE outcome = 'OPEN';
```

Output:

GAME	PROPOSITION	OPTIONS	STATUS
PHILLIES VS PIRATES	TOTAL HITS	TOTAL HITS GREATER THAN 7.5	OPEN
PHILLIES VS PIRATES	TOTAL HITS	TOTAL HITS LESS THAN 7.5	OPEN
PACKERS VS GIANTS	FIRST TO SCORE	PACKERS SCORE FIRST	OPEN
PACKERS VS GIANTS	FIRST TO SCORE	GIANTS SCORE FIRST	OPEN

Display the price history for a contract:

```
/* We are displaying the price history for the YANKEES WIN contract */

SELECT c.proposition, c.contractID,
  p.value AS Price,
  p.priceDate AS DateTime
FROM Contracts c, Prices p
WHERE c.contractID = p.contract
  AND p.proposition = c.proposition
  AND p.contract = 1
  AND p.proposition = 1
ORDER BY p.priceDate;
```

Output:

PROPOSITION	CONTRACTID	PRICE	DATETIME
1	1	1	01-MAY-14
1	1	1.3	01-MAY-14
1	1	1	01-MAY-14
1	1	0.86	01-MAY-14

Display all transactions made by a specific user:

```
/* We are displaying the transaction history for user Johnny */

SELECT u.username, o.price, o.quantity, c.proposition, c.contractID, t.tranDate
FROM Users u, Transactions t, Orders o, Contracts c
WHERE u.userID = t.tranUsr
AND o.transaction = t.tranID
AND o.proposition = c.proposition
AND o.contract = c.contractID
AND u.userID = 1
ORDER BY t.tranDate;
```

Output:

USERNAME	PRICE	QUANTITY	PROPOSITION	CONTRACTID	DATETIME
Johnny	1	65	1	1	01-MAY-14
Johnny	1.3	50	1	1	01-MAY-14

Display all open contracts that a user owns:

```
/* Lets have Frank buy some contracts for the Packers scoring first when they are first offered */

/* The following tables are updated Transactions, Orders, and Accounts */

/* Frank had $165 in his account, it will be updated to $140 after this order */

INSERT INTO Transactions (tranID, tranDate, tranUsr)

VALUES (7, to_date('05-MAY-2014 17:55:00', 'dd-mon-yyyy hh24:mi:ss'), 3);

INSERT INTO Orders (transaction, quantity, price, proposition, contract)

VALUES (7, 15, 1.00, 4, 1);

INSERT INTO Accounts (acctUsr, acctDate, amount, transaction)

VALUES (3, to_date('05-MAY-2014 17:55:00', 'dd-mon-yyyy hh24:mi:ss'), 140.00, 7);

/* Frank now has contracts for both the Yankees and Packers, but only the Packers contracts are open */


SELECT u.username, o.price, o.quantity, o.proposition, o.contract, c.outcome

FROM Orders o, Users u, Transactions t, Contracts c

WHERE t.tranUsr = u.userID

AND o.transaction = t.tranID

AND o.proposition = c.proposition

AND o.contract = c.contractID

AND u.userID = 3

AND c.outcome = 'OPEN';
```

Output:

USERNAME	PRICE	QUANITITY	PROPOSITION	CONTRACT	OUTCOME
Frank	1	15	4	1	OPEN

Display the user's account balance by transaction

```
SELECT u.username, a.acctdate, a.amount AS balance,
  o.price, o.quantity, (quantity * price) AS debit
FROM Accounts a, Users u, Orders o, Transactions t
WHERE a.acctUsr = u.userID
  AND o.transaction = t.tranID
  AND t.tranUsr = a.acctUsr
  AND a.transaction = t.tranID
  AND u.userID = 1
 /* u.userID = 3 for Frank's account history */
ORDER BY a.acctDate;
```

Output (Johnny):

USERNAME	ACCTDATE	BALANCE	PRICE	QUANTITY	DEBIT
Johnny	01-MAY-14	135	1	65	65
Johnny	01-MAY-14	70	1.3	50	65

Output (Frank):

USERNAME	ACCTDATE	BALANCE	PRICE	QUANTITY	DEBIT
Frank	01-MAY-14	165	1	35	35
Frank	05-MAY-14	140	1	15	15

Appendix

Price Update Algorithm

For every 100 contracts opened for a proposition, update the price of both associated contracts.

The price will always stay between $0.01 and $1.99. The total cost of both associated contracts must be $2.00. For every 100 contracts of a proposition use the following ratio to calculate the new price of both contracts.

Assume each proposition can have contracts for outcome A or B. If all 100 contracts are on either side of A or side B of the proposition then the price for that side will be set to $1.99 and the price for the other side will be set to $0.01. Otherwise, the new prices will be found using this ratio:

(Contracts Opened A / (Contracts Opened B + Contracts Opened B)) * $2.00

Total Opened	Contract A	Contract B	Price A	Price B
0	0	0	$1.00	$1.00
100	65	35	$1.30	$0.70
200	50	50	$1.00	$1.00
300	43	57	$0.86	$1.14
400	40	60	$0.80	$1.20
500	35	65	$0.70	$1.30
600	20	80	$0.40	$1.60
700	19	81	$0.38	$1.62
800	23	77	$0.46	$1.54
900	70	30	$1.40	$0.60
1000	78	22	$1.56	$0.44

Figure 2. Example of Price Updates

User Experience Mock Up

Figure 3. UX Mockups