|  |
| --- |
| CMPS 342 Database Project |
| Fall 2010 |
| Nick Kott |

[Phase I: Fact-Finding, Information Gathering, and Conceptual Database Design 3](#_Toc278744485)

[1. Fact-Finding Techniques and Information Gathering 3](#_Toc278744486)

[1.1 Fact-Finding Techniques 3](#_Toc278744487)

[1.2 Introduction to Enterprise/Organization 4](#_Toc278744488)

[1.3 Structure of the Enterprise 4](#_Toc278744489)

[1.4 Itemized Description of Major Objects 5](#_Toc278744490)

[1.5 Data Views and Operations for User Groups 5](#_Toc278744491)

[2. Conceptual Database Design 6](#_Toc278744492)

[2.1 Entity Set Description 6](#_Toc278744493)

[2.2 Relationship Set Description 10](#_Toc278744494)

[2.3 Related Entity Set 12](#_Toc278744495)

[2.4 E-R Diagram 14](#_Toc278744496)

[Phase II: Relational Database Model 15](#_Toc278744497)

[1. E-R Model and Relational Model 15](#_Toc278744498)

[1.1 Description 15](#_Toc278744499)

[1.2 Comparison 15](#_Toc278744500)

[1.3 Conversion from E-R Model to Relational Model. 16](#_Toc278744501)

[1.4 Constraints 18](#_Toc278744502)

[2. E-R Database to Relational Database Conversion 19](#_Toc278744503)

[3. E-R Database to Relational Database Conversion 27](#_Toc278744504)

[4. Queries 34](#_Toc278744505)

[5. Query Representation 34](#_Toc278744506)

[Phase III: Implementation of Relational Database 40](#_Toc278744507)

[1. SQL\*PLUS 40](#_Toc278744508)

[2. Oracle Schema Objects 40](#_Toc278744509)

[3. Relation Schemas and Instances 42](#_Toc278744510)

[4. SQL Queries 55](#_Toc278744511)

[Phase IV: Implementation of Relational Database 59](#_Toc278744512)

[1. Common Features in Oracle PL/SQL and MS Trans-SQL 59](#_Toc278744513)

[2. Oracle PL/SQL 59](#_Toc278744514)

[3. Oracle PL/SQL Subprogram 64](#_Toc278744515)

[Phase V: GUI Design and Implementation 67](#_Toc278744516)

[1. Daily User Activities 67](#_Toc278744517)

[Gamblers 67](#_Toc278744518)

[Bookies 67](#_Toc278744519)

[2. Relations, Views, and Subprograms 68](#_Toc278744520)

[3. Application Screen Shots 69](#_Toc278744521)

[4. Code Description 74](#_Toc278744522)

[5. Development Process 77](#_Toc278744523)

[6. Conclusion 78](#_Toc278744524)

# 

# Phase I: Fact-Finding, Information Gathering, and Conceptual Database Design

1. **Fact-Finding Techniques and Information Gathering**

### Fact-Finding Techniques

Fact-finding is a formal process by which information is acquired via various methods in order to better articulate the requirements, and intricacies of a potential database system. In order for the subsequent steps of database design to be completed properly, it is imperative that the fact-finding step be carried out thoroughly so as to ensure complete understanding by the developer. The following techniques were used in the fact-finding process of this project.

* **Interviewing** – A number of casinos and a few end users were interviewed to provide understanding of the database system. The structure of the interviews was unstructured at first, allowing the interviewee to provide direction to the interview. Later, questions were formed in order to fill in any apparent gaps left in understanding. It is worth noting that interviewing provided supplemental information that the other techniques could not provide.
* **Research** – A majority of this project’s end-user understanding was acquired via research as there is a multitude of information available. Casinos both physical, and on-line provided guides to their system, while blogs and other end-user documentation was available and surprisingly detailed on their perspective of the betting system. Research also provided some clarity to answers given in the interview process by giving fundamental information that some interviewees assumed in their responses.
* **Questionnaires** – A structured series of open-ended questions were used to give structure of the entity types and attributes. Like the research, this provided a broad understanding that was later clarified through the interviewing process.
  1. Introduction to Enterprise/Organization

Gambling has been around just about as long as any form of currency was passing through hands. It is by no stretch of the imagination to assume that sports gambling has been around as long as their respective sports. Football was created around 1900 C.E. and the NFL was founded in 1920. Money line betting is the original, most basic form of sport betting: you bet which team will win. The problem arose that a vast majority would bet for one team (the team expected to win; or *favorite*) which did not provide the balance that the bookmakers (people who took bets) desired. Thus the invention of the point spread came about. The point spread assigns a handicap to the favorite team so that they must win by a certain amount of points. After observing that more types of betting resulted in more gamblers, another form of betting was created, called totals. For totals betting, gamblers bet on the summated amount of points in a game, without regard to which team wins or loses. Recently, more forms of betting opportunities have arose, such as parlays and teasers.

* 1. Structure of the Enterprise

The basic structure of sports betting consists of three main parts: The oddsmaker, sportsbook, and gambler. The oddsmaker controls the given odds on a game. In a money line bet, they provide an opinion on who they think will win the game. However in point spread gambling, they provided the handicap by which the favorite team must win by. These handicaps are formulated through complex and exhaustive algorithms, which are not shared with the public. The sportsbook is any entity that takes bets from gamblers. They get their odds and point spreads from the oddsmakers and offer these to the gamblers. Gamblers place their bets, and receive any winnings from the sportsbook. Typically, a casino acts as both a sportsbook and oddsmaker. The gambler is perhaps the most obvious of the three: the one who places bets.

It is a fact that the sportsbook has an inherent advantage over gamblers and the bets they place. It is called the 11/10 vigorish, which means for every 11 units that a gambler bets, they have a potential to gain 10 units (a winning 11 unit bet will result in a 21 unit return). This means that a gambler has to win 52.38 percent of their bets just to break even.

* 1. Itemized Description of Major Objects

A gambler is the person that drives this enterprise. The gambler will have basic information stored, a username, name, and password as well as contact information such as an address. A gambler has a relationship with a bet in that the gambler *places* the bet.

The bet is a simple object with the attribute of amount. However it will have relationships with the game entity and book (or sportsbook) entity. The bet will *on* a game which is *held by* a book. However more information may be required to further describe the game to the end user. Attributes such as score, weather, and game type are needed. Any more details about the teams that are required will likely result in a team entity, which will be why we create one now.

A team will *play in* a game. The play relationship will have the boolean attribute “at home”. The team will have a team name, location, record, and current streak as attributes. Let us now go back and examine the relationship between the bet and the book.

A bet is held by a book. The book provides odds for the game, so they willl have the *odds on* relationship with the game. The odds on will also describe the spread of the game. The book will have the attribute of bank amount, as potential winnings can never exceed the amount in the book’s bank. The book will also have basic attributes such as name and address.

* 1. Data Views and Operations for User Groups

There are two user groups: the gamblers and the bookies. The gamblers will need to be able to log in and place bets with the book on a game. They will also need to be able to view the status of the bet after they have placed it in order to know if they have won, and how much they have won. The bookies will need to be able to set the odds on a game, and manage game data. The bookies will need to be able to view both betting summaries for a game as well as detailed gambler-level betting information.

1. **Conceptual Database Design**

### Entity Set Description

**User**

* This entity describes anybody who uses the system to place or manage bets. The intent of this database it to manage bets, so minimal contact as well as secure login information is stored.
* Candidate keys: userID, userName
* Primary key: userID
* Strong/Weak Entity: Strong
* Fields to be indexed: userID, userName

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | userID | userName | fullName | address | emailAddress | password |
| **Description** | An auto-incremented value. | A user-chosen identifier | User’s full Name | User’s address | User’s email | User’s password |
| **Domain/Type** | 32 bit Unsigned Integer | String | String | String | String | String |
| **Value Range** | 0 … 2^32 | Any Char array | Any Char array | Any Char array | Any string with ‘@’ and ‘.’ | Any Char array |
| **Default Value** | None | None | None | None | None | None |
| **Nullable?** | No | No | No | No | No | No |
| **Unique?** | Yes | Yes | No | Yes | Yes | No |
| **Single or Multiple Value** | Single | Single | Single | Single | Single | Single |
| **Simple or Composite** | Simple | Simple | Composite | Composite | Simple | Simple |

**Bet**

* This entity describes a bet placed by a gambler on a game with a book. It details amount, time, date, and winning information.
* Candidate keys: betID
* Primary key: betID
* Strong/Weak Entity: Weak
* Fields to be indexed: BetID, Date

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | betID | amount | time | date | type | win |
| **Description** | An auto-incremented value. | A positive number | Time of bet | Date of bet | Bet type | Did the bet win? |
| **Domain/Type** | 32 bit Unsigned Integer | 32 bit Unsigned Double | Time | Date | String | Boolean |
| **Value Range** | 0 … 2^32 | 0.00 … 10,000.00 | Current time only | Current Date only | Money line, spread, or totals | 0 or 1 |
| **Default Value** | None | None | Current time | Current Date | None | null |
| **Nullable?** | No | No | No | No | No | Yes |
| **Unique?** | Yes | No | No | No | No | No |
| **Single or Multiple Value** | Single | Multiple | Multiple | Multiple | Multiple | Multiple |
| **Simple or Composite** | Simple | Simple | Composite | Composite | Simple | Simple |

**Game**

* This entity describes a match between two teams.
* Candidate keys: gameID
* Primary key: gameID
* Strong/Weak Entity: Weak
* Fields to be indexed: gameID, Date

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | gameID | homeScore | awayScore | time | date | gameType | weather |
| **Description** | An auto-incremented value. | A positive number | A positive number | Time of bet | Date of bet | What type of game is this? | Weather forcast for the game |
| **Domain/Type** | 32 bit Unsigned Integer | 32 bit Unsigned Integer | 32 bit Unsigned Integer | Time | Date | String | String |
| **Value Range** | 0 … 2^32 | 0 … 999 | 0 … 999 | Current time only | Current Date only | Pre-season, regular season, playoff, super-bowl | Any char array |
| **Default Value** | None | null | null | Current time | Current Date | None | null |
| **Nullable?** | No | Yes | Yes | No | No | No | Yes |
| **Unique?** | Yes | No | No | No | No | No | No |
| **Single or Multiple Value** | Single | Multiple | Multiple | Multiple | Multiple | Multiple | Multiple |
| **Simple or Composite** | Simple | Simple | Simple | Composite | Composite | Simple | Composite |

**Team**

* This entity provides information on the teams that are bet upon.
* Candidate keys: teamID, teamName
* Primary key: teamID
* Strong/Weak Entity: Strong
* Fields to be indexed: teamID

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | teamID | teamName | city | State | record | streak |
| **Description** | An auto-incremented value. | The team’s name | The team’s city | The team’s state | The team’s record | A number of consecutive wins or losses |
| **Domain/Type** | 32 bit Unsigned Integer | String | String | String | String | 32 bit  int |
| **Value Range** | 0 … 2^32 | Any char array | Any char array | Any char array | 0-0-0 to 16-16-16 | -16 … 16 |
| **Default Value** | None | None | None | None | 0-0-0 | 0 |
| **Nullable?** | No | No | No | No | No | No |
| **Unique?** | Yes | Yes | No | No | No | No |
| **Single or Multiple Value** | Single | Single | Multiple | Multiple | Multiple | Multiple |
| **Simple or Composite** | Simple | Simple | Simple | Simple | Composite | Simple |

**Book**

* This entity describes an entity that accepts bets, sets odds, and pays winnings
* Candidate keys: bookID, bookName
* Primary key: bookID
* Strong/Weak Entity: Strong
* Fields to be indexed: bookID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | bookID | bookName | address | bank |
| **Description** | An auto-incremented value. | The book’s name (casino or otherwise) | User’s address | The book’s available assets |
| **Domain/Type** | 32 bit Unsigned Integer | String | String | 32 bit  Unsigned Int |
| **Value Range** | 0 … 2^32 | Any char array | Any Char array | 0 … 2^32 |
| **Default Value** | None | None | None | None |
| **Nullable?** | No | No | No | No |
| **Unique?** | Yes | Yes | Yes | No |
| **Single or Multiple Value** | Single | Single | Single | Multiple |
| **Simple or Composite** | Simple | Simple | Composite | Simple |

### Relationship Set Description

**Gambles:**

This ternary relationship is between the gambler, the bet, and the game. Every bet must have one gambler and one game. Whenever a gambler creates a bet on a game, this relationship is created. This links the userID with the betID and the gameID.

* Mapping cardinality: M..M..M
* Descriptive field: teamID
* Participation constraint: mandatory for Bet, optional for gambler and game

**Held\_By:**

This relationship is between the bet and the book. It describes the entity that manages the bet, and pays winnings to the gambler. The linking attributes are betID and bookID.

* Mapping cardinality: M..1
* Descriptive field: none
* Participation constraint: mandatory for Bet, optional for book

**Plays\_In:**

This relationship is between the game and a team. It describes the game a team will play in. It also describes if the team is playing at home and if they win. The linking attributes are gameID and teamID.

* Mapping cardinality: 2..M
* Descriptive field: atHome, win
* Participation constraint: mandatory for game, optional for team.

**Odds\_On:**

This relationship is between the book and the game entity. It describes which team is favorited in a game by the teamID, and by what point spread. This information is used when placing bets. The linking attributes are bookID and gameID.

* Mapping cardinality: M..M
* Descriptive field: favTeamID, pointSpread
* Participation constraint: optional for book and for game

**Works\_For:**

This relationship is between the bookie and the book entity. It describes which book a bookie works for and can access. This information is used when managing the system. The linking attributes are bookID and userID.

* Mapping cardinality: M..1
* Descriptive field: position
* Participation constraint: optional for book mandatory for bookie

### Related Entity Set

**Gambler**

* This entity is a subclass specialization of the User entity, and serves the purpose of storing a gambler’s payment information.
* Specialization/Generalization Relationship: disjoint, partial participation
* Aggregation: IS-A-PART-OF Users

|  |  |  |
| --- | --- | --- |
| **Name** | routingNo | bankAcctNo |
| **Description** | Bank routing number | Bank Account Number |
| **Domain/Type** | String | String |
| **Value Range** | Array of numerical chars | Array of numerical chars |
| **Default Value** | None | None |
| **Nullable?** | No | No |
| **Unique?** | No | Yes |
| **Single or Multiple Value** | Multiple | Single |
| **Simple or Composite** | Simple | Simple |

**Bookie**

* This entity is a subclass specialization of the User entity, and serves the purpose of storing a bookie’s affiliation information. This information will be used to filter betting reports and management access.
* Specialization/Generalization Relationship: disjoint, partial participation
* Aggregation: IS-A-PART-OF Users

|  |  |
| --- | --- |
| **Name** | position |
| **Description** | Bookie’s official title |
| **Domain/Type** | String |
| **Value Range** | Array of chars |
| **Default Value** | None |
| **Nullable?** | No |
| **Unique?** | No |
| **Single or Multiple Value** | Multiple |
| **Simple or Composite** | Simple |

* 1. E-R Diagram

**Users**

**userID (PK)**

**userName**

fullName

address

emailAddress

password

**Bookie**

position

**Gambler**

routingNo

bankAcctNo

n

n

**Bet**

**betID(PK)**

amount

time

date

type

win

**Game**

**gameID(PK)**

homeScore

awayScore

time

date

gameType

weather

n n n 2

Works For

Gambles

Plays\_In

**Team**

**teamID(PK)**

teamName

city

state

record

streak

teamID

n

atHome

win

n

Held\_By

**Book**

**bookID(PK)**

bookName

address

bank

n n

Odds\_On

1

1

favTeamID

pointSpread

# Phase II: Relational Database Model

1. **E-R Model and Relational Model**

### Description

The Relationship Data Model first caught the attention of the programming industry due to its simplicity and mathematical foundation in a paper written by Ted Codd of IBM in 1970. In the early 1980s, the model was applied for commercial use as the SQL/DS system. Since then the model has been widely used in major systems such as SQL Server and Oracle. This model uses a collection of relations to compose the database. Each relation contains instances (tuples) that are described by the data which they hold. This data is organized into attributes that is further specified by the domain. The result is a logically-simplified theoretical representation of a database which is therefore easier to convert into an actual database.

### Comparison

The Entity-Relationship model is a highly conceptual model that makes it ideal for the initial design of a database. This high-level conceptualization is most useful when represented visually through an ER diagram. This is perhaps one of the few mediums through which a database designer and a non-technical person of reference (or users) can clearly communicate their understandings of the database concept. It then follows that this would be an ideal starting model, as any input on the design from non-technical persons from here forward will typically be difficult to integrate.

The Relational model is the next step in transforming the conceptualization into an actual database. Each entity and relationship from the previous model becomes a relation. Each relation contains attributes that describe the relation. For each instance of the relation there is a tuple; each tuple contains values for the attributes which collectively describe the instance. While it may be harder to visualize like the E-R Model, the structure and detail of the Relational model is a more explicit iteration of the database concept. This structure is also closer to the structure of the implemented database.

### Conversion from E-R Model to Relational Model.

The conversion from an E-R Model is not as much a necessary process as it is a natural process. To skip this conversion, one could argue that a lot of time could be saved. The same could be said if we also skipped the E-R Model. However, through experience we have learned that this is a very bad idea. We first take the ideas behind a database, and make it a high-level E-R Model. Then we take the high-level E-R Model and convert it to a lower-level Relational Model. Finally, we convert the Relational Model to the actual database. These careful iterations allow us to properly think through the organization and structure of our database so that the final product is flawless in its implementation.

The conversion of strong entity types is relatively simple; each strong entity becomes a relation. This relation contains all the simple attributes of the entity. A composite attribute is broken into its simple components. One primary key is chosen, while making note of any other keys as candidates for indexing.

Weak entity types are similarly handled. Each weak entity becomes a relation with its attributes made up of the simple attributes of the entity. However, the weak entity also includes the primary key of the owner entity type as a foreign key. This, along with the partial key (if any) of the weak entity compose the primary key.

Binary 1:1 relationship types have three approaches that can be used to convert to the relatioal model depending on the situation. In real-world situations these relationship types are not very likely to occur. The three approaches are:

* Foreign key approach: add the primary key of the other relation to the one that has total participation as a foreign key. This avoids having a large number of null values.
* Merged relation approach: merge the two entity types and the relation into one relation that includes the attributes of all its constituents. This is only acceptable when both entities have full participation.
* Cross-reference or relationship relation approach: create a relationship relation that contains both participating entities’ keys as attributes. This is ideal of low participation relationships as it saves us from having a large number of null values in one of the relations.

The foreign key and relationship relation approach are used for binary 1:N relationships as well. The foreign key approach asks us to add the primary key of the 1-side as a foreign key of the N-side. The relationship relation works the same as before, for each relationship instance we have a tuple containing the primary keys of the two entities. Which approach to use depends on the participation of the N-side entity as well as the size of memory each approach uses per tuple. By multiplying the number of records by the memory size, we should be able to determine which approach is appropriate. Binary M:N relationship types must also use the relationship relation due to the cardinality constraints.

Multivalued attributes are handled by creating a new relation for the attribute, and assigning each part of the multi-valued attribute as its own attribute to the relation. The relation can then be referenced by a foreign key attribute by any relation that wishes to use it. For N-ary relationship types, we create a relationship relation that contains all participating relations’ primary keys as attributes along with any simple attributes of the relationship type.

For specialization and generalization, we also have multiple options for conversion. These options are:

* Create a relation for the superclass, and a relation for each subclass. Each subclass would have its attributes union with the superclass. Also, the primary key of the subclass would be the same as the superclass. This option is acceptable for any specialization.
* Create a relation for every subclass that has its own attributes as well as the superclass’ attributes and primary key. This only works when every superclass entity belongs to at least one of the subclasses.
* Create a single relation that contains all the subclass and superclass attributes, the superclass’ primary key, and a type attribute to specify which subclass to which a tuple belongs. This option could have many null values if there are numerous subclass attributes and only works if they are disjoint.
* Create a single relation as per the previous option; however, create Boolean type attributes for each subclass type. This option is appropriate for a specialization where subclasses overlap as well as disjoint.

When converting a category, you must add a surrogate key if the defining superclasses do not share a common key. The surrogate key becomes the primary key of the category’s relation, and a foreign key to the superclasses. If the superclasses share a primary key, then we merely use this as the primary key of the new relation. Now that we have covered various instances in the conversion process, we need to consider constraints.

### Constraints

Constrains are limitations we enforce upon a database to ensure that order persists in our operations and that no unexpected value occurs within the data. Entity constraints maintain that no two tuples are duplicated. This is usually achieved by including a unique primary key to each tuple in a relation; a primary key can be unique but must not be null. Having a unique identifier provides us with a means to select and compare specific tuples within a relation. Similar is the constraint that a reference to a tuple must refer to a tuple and not null. This referential constraint can be taken further as a foreign key. A foreign key must have the same domain as the primary key of which it refers. A foreign key must also exist as a primary key in the reference relation, or be a null value. Check constrains and business rules allow us to customize a database to the specific application. Values must not exist outside the domain of the business. These constraints keep the data relevant and concise.

1. **E-R Database to Relational Database Conversion**

**Users**

*Attributes*

userID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

userName

Domain: string. Must be between 6 and 12 alpha-numeric characters long. Cannot be NULL.

fullName

Domain: string. Must be between 4 and 50 characters long. Composite attribute in the format of: lastName, firstName. A comma and space separate the constituent attributes. Cannot be NULL.

Address

Domain: string. Must be less than or equal to 75 characters. Composite attribute in the format of: Street 1, Street2, City, State, Zip. Composite is comma delimited. Cannot be NULL.

emailAddress

Domain: string. Must be a valid email address containing one ‘@’ character and at least one ‘.’ character. Cannot be NULL.

Password

Domain: string. Must be between 6 and 12 characters long. Cannot be NULL.

*Constraints*

Primary key: userID, must be unique and not NULL.

Business Rule: none of the attributes can be null for payment purposes. Every userName, address, and emailAddress must be unique. Every user must be a gambler or bookie.

*Candidate Keys*

userID, userName

**Gambler**

*Attributes*

userID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

routingNo

Domain: string. Must be 9 numeric characters long. Cannot be NULL.

bankAcctNo

Domain: string. Must be less than 20 numeric characters long. Cannot be NULL.

*Constraints*

Primary Foreign key: userID, must be unique and not NULL. Must exist in the Users relation.

Business Rule: the bank information must be present in order to charge for bets. The bankAcctNo must be unique for the given routingNo, thus giving a unique bank account.

*Candidate Keys*

userID

**Bookie**

*Attributes*

userID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

position

Domain: string: Must be less than 20 numeric characters long. Cannot be NULL.

bookID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

*Constraints*

Foreign key: userID, bookID. Must exist in their respective relations and be NOT NULL.

Business Rule: each employee must have a position and bookID to determine access rights.

*Candidate Keys*

userID

**Bet**

*Attributes*

betID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

bookID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

userID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

gameID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

teamID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

amount

Domain: unsigned double. Must be between 0.00 and 10,000.00 numeric characters long. Precision of 2. Cannot be NULL.

bDateTime

Domain: date. Composite attribute in the format of: DD-MON-YY HH:MI:SS. Default value of current datetime. Cannot be NULL.

bType

Domain: string. Must be either “money line”, “spread”, or “totals”. Cannot be NULL.

win

Domain: boolean. Must be 0 or 1; false or true.

*Constraints*

Primary key: betID, must be unique and not NULL.

Foreign key: bookID, userID, gameID, teamID, must be unique and not NULL. Must contain a value that exists in their respective relations.

Business Rule: none of the attributes can be null but the win attribute which will not be set until the conclusion of the bet. Every bet has an associated gambler, game, and book.

*Candidate Keys*

userID

**Game**

*Attributes*

gameID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

hTeam

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

aTeam

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

homeScore

Domain: unsigned integer. Must be between 0 and 999.

awayScore

Domain: unsigned integer. Must be between 0 and 999.

gDateTime

Domain: date. Composite attribute in the format of: DD-MON-YY HH:MI:SS, value of current date. Cannot be NULL.

gameType

Domain: string. Must be either “pre-season”, “regular season”, “playoff”, or “super-bowl”. Cannot be NULL.

weather

Domain: string. Composite value in the format of: temperature/weatherType. Delimited using ‘/’ character. Cannot be NULL.

*Constraints*

Primary key: gameID, must be unique and not NULL.

Foreign key: hTeam, aTeam must be unique and not NULL. Must contain teamIDs that exist in the Team relation and must not be equal.

Business Rule: the pk, gTime, gDate, and gameType are set upon creation. The other attributes accept null so that they may be entered when available. Every game has two teams associated with it.

*Candidate Keys*

gameID

**Team**

*Attributes*

teamID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

teamName

Domain: string. Must be less than 25 characters long. Cannot be NULL.

city

Domain: string. Must be less than 25 characters long. Cannot be NULL.

state

Domain: string. Must be less than 25 characters long. Cannot be NULL.

record

Domain: string. Composite attribute in the format of: W-L-T. Hash mark, ‘-‘, delimited. Default value of “0-0-0”. Cannot be NULL.

streak

Domain: signed integer. Must be between -16 and 16. Default value of 0. Cannot be NULL.

*Constraints*

Primary key: teamID, must be unique and not NULL.

Business Rule: the teamName must be unique. The record will be the team’s record for the season, and streak their current winning/losing streak.

*Candidate Keys*

teamID, teamName

**Book**

*Attributes*

bookID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

bookName

Domain: string. Must be less than 50 characters long. Cannot be NULL.

address

Domain: string. Must be less than or equal to 75 characters. Composite attribute in the format of: Street 1, Street2, City, State, Zip. Composite is comma delimited. Cannot be NULL.

bank

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

*Constraints*

Primary key: bookID, must be unique and not NULL.

Business Rule: the bookName must be unique. The bank must be greater than 0 to take bets. The book must be able to cover the open bets with the amount in its bank.

*Candidate Keys*

teamID, bookName

**Odds\_On**

*Attributes*

bookID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

gameID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

teamID

Domain: unsigned integer: 0 to 2^32-1. Cannot be NULL.

pointSpread

Domain: signed integer: -999 to 999. Cannot be NULL.

*Constraints*

Foreign key: bookID, gameID, and teamID collectively make a unique key. They must all exist in their respective reference relations.

Business Rule: if a bet type is “point spread” this record will be referenced to determine if they have won or not. This information will also be displayed to the users.

*Candidate Keys*

bookID, gameID, and teamID

1. **E-R Database to Relational Database Conversion**

**Users(userID,** userName, fullName, Address, emailAddress, password)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **userID** | **userName** | **fullName** | **Address** | **emailAddress** | **password** |
| 1 | Nkott0 | Kott, Nicholas | 1920 Hugo St., Bakersfield, CA, 93308 | Nkott0@gmail.com | Alfred0 |
| 2 | SawCat | Cat, Sawyer | 1922 Hugo St., Bakersfield, CA, 93308 | SawCat@gmail.com | 1e8gn54 |
| 3 | Mialicious | Cat, Mia | 2000 Hugo St., Bakersfield, CA, 93308 | Mialicious@gmail.com | Randumb1 |
| 4 | FrankC | Caliendo, Frank | 2100 Truxtun Ave., Saint Clair, MI, 48079 | FrankC@yahoo.net | 123456 |
| 5 | DrWang | Wang, Huaqing | 4200 Camino Media, Bakersfield, CA, 12345 | hwang@cs.csubak.edu | fluffydog |
| 6 | JCardenas | Cardenas, Jorge | 999 Olive Dr., San Deigo, CA, 10000 | JCardenas@kern.co.ca.us | 3l337one |
| 7 | TRutledge | Rutledge, Thomas | 777 Luck St., Las Vegas, NV, 77777 | TRutledge@kern.co.ca.us | 8008ies |
| 8 | SySamat | Sy, Samat | 123 Fake St., Austin, TX, 56560 | SySamat@kern.co.ca.us | 1ang2dt |
| 9 | TVanMetre | Van Metre, Tom | 688 Green Ave., ID, 88809 | TVanMetre@kern.co.ca.us | 45f98fdhg9 |
| 10 | PiersonJiu | Jiu, Pierson | 45247 Golden Gate Pkwy., San Francisco, CA, 91240 | PiersonJiu@kern.co.ca.us | Pazz7432 |
| 11 | Mialing | Cart, Mart | 4000 Hugo St., Bakersfield, CA, 93309 | cart@gmail.com | Randsfgh |
| 12 | UglyOrg | Brown, Orge | 5500 Truxtun Ave., Saint Clair, MI, 48079 | haaaar@yahoo.net | 1j4788g |
| 13 | OtherPer | Van, Big | 200 Camino Media, Bakersfield, CA, 12345 | hoolahut@austin.edu | blueish |
| 14 | Obamakin | Obama, Barak | 999 Olive Dr., SomeCity, D.C., 10000 | Theprez@usa.gov | Freeh3alth |
| 15 | drooling | Robers, Trey | 777 Luck St., Las Vegas, FL, 86095 | TREY@treyz.us | ohmygosh |
| 16 | toomany | Que, Por | 123 Fake St., Austin, Az, 55670 | whyis@thishappening.org | 49857v4 |
| 17 | NOOOO | Black, Jack | 688 Verners Ave., ID, 88809 | blackJ@aol.com | H000ti3 |
| 18 | tokcin | Mama, Joe | 45247 Golden Gate Pkwy., Toledo, OH, 46709 | tokcin@ohio.oh | Joebos87 |

**Gambler(userID,** routingNumber, bankAcctNo)

|  |  |  |
| --- | --- | --- |
| **userID** | **routingNumber** | **bankAcctNo** |
| 1 | 332484215 | 704654654 |
| 2 | 194650252 | 605465 |
| 3 | 956831478 | 00054480 |
| 4 | 021859623 | 468405210 |
| 5 | 540405465 | 000584584 |
| 6 | 465406545 | 5404654 |
| 7 | 332484215 | 455404654 |
| 8 | 970321454 | 980051 |
| 9 | 002415547 | 04654 |
| 10 | 224086752 | 06540644 |

**Bookie(userID,** position, bookID)

|  |  |  |
| --- | --- | --- |
| **userID** | **position** | **bookID** |
| 11 | Sys Admin | 1 |
| 12 | Sys Admin | 2 |
| 13 | Sys Admin | 3 |
| 14 | Sys Admin | 4 |
| 15 | Sys Admin | 5 |
| 16 | Sys Admin | 6 |
| 17 | Sys Admin | 7 |
| 18 | Data Clerk | 7 |

**Bet(betID,** *bookID*, *userID****,*** *gameID, teamID,* amount, time, date, type, win)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **betID** | **bookID** | **userID** | **gameID** | **teamID** | **amount** | **time** | **date** | **type** | **win** |
| 1 | 1 | 1 | 1 | 6 | 20.00 | 13:18 | 04/09/2010 | money line | 1 |
| 2 | 2 | 2 | 2 | 7 | 100.00 | 12:24 | 05/24/2010 | totals | 0 |
| 3 | 3 | 3 | 3 | 3 | 1.00 | 08:32 | 01/30/2010 | spread | 1 |
| 4 | 4 | 4 | 3 | 3 | 25.00 | 06:41 | 02/14/2010 | money line | 1 |
| 5 | 2 | 5 | 4 | 5 | 88.00 | 15:01 | 03/29/2010 | spread | 0 |
| 6 | 3 | 6 | 5 | 7 | 2,000.00 | 10:08 | 09/19/2010 | money line | 0 |
| 7 | 1 | 7 | 6 | 1 | 200.00 | 22:56 | 10/14/2010 | totals | 0 |
| 8 | 5 | 8 | 7 | 8 | 350.00 | 00:14 | 10/10/2010 | money line | 1 |
| 9 | 6 | 9 | 8 | 3 | 75.00 | 02:21 | 10/14/2010 | spread | 1 |
| 10 | 7 | 10 | 9 | 4 | 20.00 | 03:51 | 10/14/2010 | totals | 0 |

**Game(gameID,** *hTeam, aTeam*, homeScore, awayScore, time, date, gameType, weather)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **gameID** | **hTeam** | **aTeam** | **homeScore** | **awayScore** | **time** | **date** | **gameType** | **weather** |
| 1 | 1 | 6 | 10 | 32 | 10:00 | 08/15/2010 | pre-season | 95F/Sunny |
| 2 | 2 | 7 | 24 | 23 | 13:00 | 08/15/2010 | pre-season | 75F/Indoors |
| 3 | 3 | 8 | 13 | 7 | 18:00 | 01/30/2010 | super-bowl | 19F/Snow |
| 4 | 5 | 10 | 3 | 21 | 13:00 | 09/21/2010 | regular season | 95F/Sunny |
| 5 | 3 | 7 | 32 | 10 | 10:00 | 10/08/2010 | regular season | 85F/Sunny |
| 6 | 1 | 2 | 23 | 24 | 13:00 | 10/15/2010 | regular season | 55F/Windy |
| 7 | 5 | 8 | 12 | 56 | 10:00 | 10/15/2010 | regular season | 65F/Rain |
| 8 | 6 | 3 | 7 | 13 | 13:00 | 10/15/2010 | regular season | 80F/Cloudy |
| 9 | 7 | 4 | 21 | 3 | 10:00 | 10/15/2010 | regular season | 105F/Sunny |

**Team(teamID,** teamName, city, state, record, streak)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **teamID** | **teamName** | **city** | **state** | **record** | **streak** |
| 1 | Lions | Detroit | MI | 1-4-0 | -1 |
| 2 | Chargers | San Diego | CA | 1-4-0 | 1 |
| 3 | Bills | Buffalo | NY | 0-5-0 | -5 |
| 4 | Texans | Houston | TX | 4-1-0 | 2 |
| 5 | Dolphins | Miami | FL | 3-2-0 | 2 |
| 6 | Colts | Indianapolis | IN | 4-1-0 | 2 |
| 7 | Bengals | Cincinnati | OH | 2-2-0 | -1 |
| 8 | Giants | New York | NY | 3-2-0 | 2 |
| 9 | Patroits | Foxboro | MA | 3-2-0 | 1 |
| 10 | Packers | Green Bay | WI | 2-3-0 | -3 |

**Book(bookID,** bookName, address, bank)

|  |  |  |  |
| --- | --- | --- | --- |
| **bookID** | **bookName** | **address** | **bank** |
| 1 | Mirage | 3400 Las Vegas Boulevard South, Las Vegas, NV | 24,000,000.00 |
| 2 | Bellagio | 3600 Las Vegas Blvd South, Las Vegas, NV | 38,000,000.00 |
| 3 | MGM Grand | 3799 S. Las Vegas Blvd., Las Vegas, NV | 300,000,000.00 |
| 4 | Venetian | Venetian, 3355 Las Vegas Blvd S, Uninc, NV | 220,000,000.00 |
| 5 | Caesars Palace | 3570 Las Vegas Blvd South, Las Vegas, NV | 700,000,000.00 |
| 6 | Wynn | 3131 Las Vegas Blvd. South, Las Vegas, NV | 1,100,000,000.00 |
| 7 | Luxor | 3900 Las Vegas Blvd, S, Las Vegas, NV | 550,000,000.00 |

**Odds\_On(***bookID***,** *gameID*, *teamID*, pointSpread)

|  |  |  |  |
| --- | --- | --- | --- |
| **bookID** | **gameID** | **teamID** | **pointSpread** |
| 1 | 1 | 6 | 1 |
| 2 | 1 | 1 | 3 |
| 3 | 1 | 1 | 3 |
| 4 | 1 | 1 | 3 |
| 5 | 1 | 1 | 3 |
| 6 | 1 | 1 | 7 |
| 7 | 1 | 1 | 4 |
| 1 | 2 | 7 | 7 |
| 2 | 2 | 7 | 7 |
| 3 | 2 | 7 | 7 |
| 4 | 2 | 7 | 6 |
| 5 | 2 | 7 | 7 |
| 6 | 2 | 7 | 7 |
| 7 | 2 | 7 | 7 |
| 1 | 3 | 3 | 2 |
| 2 | 3 | 3 | 0 |
| 3 | 3 | 3 | 2 |
| 4 | 3 | 3 | 3 |
| 5 | 3 | 3 | 3 |
| 6 | 3 | 3 | 4 |
| 7 | 3 | 3 | 2 |
| 1 | 4 | 10 | 1 |
| 2 | 4 | 10 | 1 |
| 3 | 4 | 10 | 2 |
| 4 | 4 | 10 | 4 |
| 5 | 4 | 10 | 2 |
| 6 | 4 | 10 | 1 |
| 7 | 4 | 5 | 1 |
| 1 | 5 | 3 | 3 |
| 2 | 5 | 3 | 3 |
| 3 | 5 | 3 | 3 |
| 4 | 5 | 3 | 3 |
| 5 | 5 | 3 | 4 |
| 6 | 5 | 3 | 2 |
| 7 | 5 | 3 | 7 |
| 1 | 6 | 1 | 0 |
| 2 | 6 | 1 | 0 |
| 3 | 6 | 2 | 0 |
| 4 | 6 | 2 | 0 |
| 5 | 6 | 1 | 0 |
| 6 | 6 | 2 | 0 |
| 7 | 6 | 1 | 1 |
| 1 | 7 | 8 | 10 |
| 2 | 7 | 8 | 10 |
| 3 | 7 | 8 | 10 |
| 4 | 7 | 8 | 10 |
| 5 | 7 | 8 | 9 |
| 6 | 7 | 8 | 10 |
| 7 | 7 | 8 | 10 |
| 1 | 8 | 6 | 3 |
| 2 | 8 | 6 | 3 |
| 3 | 8 | 6 | 3 |
| 4 | 8 | 6 | 3 |
| 5 | 8 | 6 | 3 |
| 6 | 8 | 6 | 3 |
| 7 | 8 | 6 | 3 |
| 1 | 9 | 7 | 2 |
| 2 | 9 | 7 | 4 |
| 3 | 9 | 7 | 2 |
| 4 | 9 | 7 | 3 |
| 5 | 9 | 7 | 3 |
| 6 | 9 | 7 | 7 |
| 7 | 9 | 7 | 3 |

1. **Queries**

* Select teams that have won more than 1 game as the away team.
* Select gamblers that have bet on more than one game.
* Select the largest winnings for games on 10/24/2010.
* Select gamblers who have bets with all books.
* Select gamblers who have open bets.
* Select books that have never had a bet.
* Select gamblers that have never had a winning bet
* Select gamblers that have only placed bets over $1,000.00
* Select gamblers that have won more than once.
* Select the game that has the largest bet on it.

1. **Query Representation**

**Select teams that have won more than 1 game as the away team.**

*Relational Algebra:*

OneWin

*Tuple Relational Calculus:*

}

*Domain Relational Calculus:*

{<t,n> | Team(t,n,\_,\_,\_,\_) ^ (g1)(g2)(Game(g1,\_,t,hs1,>hs1,\_,\_,\_,\_) ^ Game(g1,\_,t,hs1,>hs1,\_,\_,\_,\_) ^ g1 != g2) }

**Select the gamblers that have bet on more than one game.**

*Relational Algebra:*

OneBetBet \* Users \* Gambler

((OneBet g1 OneBet g2))

*Tuple Relational Calculus:*

{ u.fullName | Users(u) ^ (b1)(b2)(Bet(b1) ^ Bet(b2) ^ b1.userID = u.userID ^ b2.userID = u.userID ^ b1.betID != b2.betID) }

*Domain Relational Calculus:*

{<u,n> | Users(u,\_,n,\_,\_,\_) ^ (b1)(b2)(Bet(b1,\_,u,\_,\_,\_,\_,\_,\_) ^ Game(b2,\_,u,\_,\_,\_,\_,\_,\_) ^ b1 != b2) }

**Select the largest winnings for games on 10/24/2010.**

*Relational Algebra:*

bg (Bet b Game g)

*Tuple Relational Calculus:*

{ b.amount | Bet(b) ^ (g)(Game(g) ^ g.gameID = b.gameID ^ g.date = “10/24/2010” ^ b.win = 1^ (b2)(Bet(b2) ^ g.gameID = b2.gameID ^ g.date = “10/24/2010” ^ b2.win = 1 ^ b2.amount > b.amount) }

*Domain Relational Calculus:*

{ a | (g) (b)(Bet(b,\_,\_,g,\_,a,\_,\_,\_,1) ^ Game(g,\_,\_,\_,\_,\_,”10/24/2010”,\_,\_) ^ (b2) (g2)( Bet(b2,\_,\_,g2,\_,>a,\_,\_,\_,1) ^ Game(g2,\_,\_,\_,\_,\_,”10/24/2010”,\_,\_)) }

**Select gamblers who have bets with all books.**

*Relational Algebra:*

Users \* ( (Book b))

*Tuple Relational Calculus:*

{ u | Users(u) ^ (b)(Book(b) ^ ()(Bet(e) u.userID = e.userID ^ b.bookID = e.bookID)) }

*Domain Relational Calculus:*

{<u,n> | User(u,\_,n,\_,\_,\_) ^(b)( Book(b,\_,\_,\_)Bet(\_,b,u,\_,\_,\_,\_,\_,\_,\_) ) }

**Select gamblers who have open bets.**

*Relational Algebra:*

*Tuple Relational Calculus:*

{ u.fullName | Users(u) ^ (b)(bet(b) ^b.userID = u.userID ^ b.win = null)) }

*Domain Relational Calculus:*

{<u,n> | User(u,\_,n,\_,\_,\_) ^(b)( Bet(b,\_,u,\_,\_,\_,\_,\_,\_,null)) }

**Select books that have never had a bet.**

*Relational Algebra:*

*Tuple Relational Calculus:*

{ b.bookName | Book(b) ^ (e)(bet(e) ^b.bookID = e.bookID)) }

*Domain Relational Calculus:*

{<b,n> | Book(b,n,\_,\_,) ^ (e)( Bet(e,b,\_,\_,\_,\_,\_,\_,\_,\_)) }

**Select gamblers that have never had a winning bet.**

*Relational Algebra:*

*Tuple Relational Calculus:*

{ u | Users(u) ^ (e)(bet(e) ^ u.userID = e.userID ^ (e2)(bet(e2) ^ u.userID = e2.userID ^ e2.win=1) )}

*Domain Relational Calculus:*

{<u,n> | Users(u,n,\_,\_,\_,\_) ^(e)( Bet(e,\_,u,\_,\_,\_,\_,\_,\_,\_)^ (e2)( Bet(e2,\_,u,\_,\_,\_,\_,\_,\_,1)) }

**Select gamblers that have only placed bets over $1,000.00**

*Relational Algebra:*

*Tuple Relational Calculus:*

{ u | Users(u) ^ (e)(bet(e) ^ u.userID = e.userID e.amount > 1000)}

*Domain Relational Calculus:*

{<u,n> | Users(u,n,\_,\_,\_,\_) ^(e)( Bet(e,\_,u,\_,\_,\_ ,\_,\_,\_,\_) Bet(e,\_,u,\_,\_,>1000 ,\_,\_,\_,\_))}

**Select gamblers that have won more than once.**

*Relational Algebra:*

*Tuple Relational Calculus:*

{ u | Users(u) ^ (e)(bet(e) ^ u.userID = e.userID ^ (e2)(bet(e2) ^ u.userID = e2.userID ^ e.betId != e2.betID))}

*Domain Relational Calculus:*

{<u,n> | Users(u,n,\_,\_,\_,\_) ^(e)( Bet(e,\_,u,\_,\_,\_ ,\_,\_,\_,\_)^ e != e2)}

**Select the game that has the largest bet on it.**

*Relational Algebra:*

*Tuple Relational Calculus:*

{ g | Game(g) ^ (e)(bet(e) ^ g.gameID = e.gameID ^ (e2) (g2)(bet(e2) ^ game(g2) ^ g2.gameID = e2.gameID ^ e2.amount > e.amount))}

*Domain Relational Calculus:*

{g | Game(g,\_,\_,\_,\_,\_,\_,\_,\_) ^(a)( Bet(\_,\_,\_,g,\_,a ,\_,\_,\_,\_))}

# Phase III: Implementation of Relational Database

1. **SQL\*PLUS**

The Structured Query Language (SQL) is the standard language for interacting with a DBMS. SQL allows for a standard, efficient way of using a database management system regardless of the specific database type. Several variations of the SQL language are T-SQL, MySQL, and SQL\*PLUS. While these may have slight variations between them, they are all the same basic SQL language. SQL\*PLUS allows users to execute SQL scripts that run queries. This allows users the ability to destroy and recreate a database in a few seconds.

1. **Oracle Schema Objects**

A set of logical data structures or schema objects comprises a schema. Schema object are not physically related in a one-to-one relationship to their physical files. Rather, schema objects are logically stored within a tablespace of the database and the phisical files are contained in one or more of the tablespace’s datafiles. Oracle has a number of Schema Objects, the most widely used being the Table.

* Tables

Tables are the most basic unit of storage in the Oracle database. Tuples and attributes take the form of rows and columns. Columns are assigned unique names, a datatype and a width. The width can be predetermined for some datatypes, but needs to be specified for others. Rules can be set for tables, called constraints, that limit the acceptable values that go into a column for a given row.

* Views

A view is a customizable presentation of the data contained in a table, or presented in separate views. A view could be considered a virtual table in that it takes the result set of a query and presents it as a table. For the most part, you can operate a view like you would a table, with some restrictions on the update, insert, and delete operations. A view is stored as only its definition (the query) and so takes very little space in the database.

* Dimensions

A dimension declares the hierarchical relationship between columns. It is a container of logical relationships between columns, and does not have any data storage assigned to it. If a dimension is denormalized, the columns will come from the same table. Conversely, if they are from multiple tables the dimension is considered to be fully or partially normalized.

* Sequence Generator

A sequence generator allows for faster throughput in a multiuser environment. The sequence generator avoids the serialization necessary when two users are inserting and waiting for sequential numbers at the same time. Thus the user’s wait time is reduced. Oracle stores the definitions for all of a database’s sequences in the SYSTEM tablespace as a single dictionary table. The sequence numbers are generated independently of tables, and therefore can be reused within a database.

* Synonyms

A synonym is an alias for any other Schema object, even a synonym. They require no more storage than their definition in the data dictionary. There are both public and private synonyms, depending on who has access to it. The can be used to shorten the string used to access a schema object, or to hide its identity or location in a schema.

* Indexes

Indexes are optional structures used to enhance the access time associated with a table. Indexes store associations between columns based on a specific logical indexing scheme. They are best used only on the columns that are frequently used to identify and return rows, and do little good on columns that contain frequently repeated data. While indexes help with information retrieval, they can cause increased latency in insertion.

* Database Links

A database link can be thought of as a pointer to a constant database server. They are a read-only link that allows users to access the information on another server, however they cannot manipulate the data on that server. They are useful for access information without being an actual user of the remote database.

* Stored Procedures and Functions

Stored procedures in Oracle are PL/SQL procedures and operate much like a cross between a function and a query. Functions accept parameters and return a scalar value. Stored procedures also accept parameters, however they return a result set, much like a table or view.

* Packages

Packages are a construct of PL/SQL objects (such as procedures, variables, cursors, or functions). A package has two parts: the specification and the body. The specification defines what objects comprise the package, while the body implements the objects in the code. Their purpose is to execute as a single instruction.

1. **Relation Schemas and Instances**

Tables in the project were created in a similar format to the relation example below:

CREATE TABLE NK\_Team

( teamID number(10) PRIMARY KEY,

teamName varchar2(25) unique not null,

city varchar2(25) unique not null,

state varchar2(25) unique not null,

record varchar2(25) not null,

streak number(10) DEFAULT 0)

ENABLE PRIMARY KEY USING INDEX TABLESPACE cs342index;

Relation names are directly related to table names in that: NK\_RelationName. The following are the schemas and instances of the relational database implemented:

**NK\_Users**

SQL query executed:

desc NK\_Users

Name Null? Type

----------------------------------------- -------- -------------------------

USERID NOT NULL NUMBER(10)

USERNAME NOT NULL VARCHAR2(12)

FULLNAME NOT NULL VARCHAR2(50)

ADDRESS NOT NULL VARCHAR2(75)

EMAILADDRESS NOT NULL VARCHAR2(50)

PASSWORD NOT NULL VARCHAR2(12)

SQL query executed:

select \* from NK\_Users

USERID USERNAME FULLNAME ADDRESS EMAILADDRESS PASSWORD

---------- ------------ -------------------------------------------------- --------------------------------------------------------------------------- -------------------------------------------------- ------------

2 SawCat Cat, Sawyer 1922 Hugo St., Bakersfield, CA, 93308 SawCat@gmail.com 1e8gn54

3 Mialicious Cat, Mia 2000 Hugo St., Bakersfield, CA, 93308 Mialicious@gmail.com Randumb1

4 FrankC Caliendo, Frank 2100 Truxtun Ave., Saint Clair, MI, 48079 FrankC@yahoo.net 123456

5 DrWang Wang, Huaqing 4200 Camino Media, Bakersfield, CA, 12345 hwang@cs.csubak.edu fluffydog

6 JCardenas Cardenas, Jorge 999 Olive Dr., San Deigo, CA, 10000 JCardenas@kern.co.ca.us 3l337one

7 TRutledge Rutledge, Thomas 777 Luck St., Las Vegas, NV, 77777 TRutledge@kern.co.ca.us 8008ies

8 SySamat Sy, Samat 123 Fake St., Austin, TX, 56560 SySamat@kern.co.ca.us 1ang2dt

9 TVanMetre Van Metre, Tom 688 Green Ave., ID, 88809 TVanMetre@kern.co.ca.us 45f98fdhg9

10 PiersonJiu Jiu, Pierson 45247 Golden Gate Pkwy., San Francisco, CA, 91240 PiersonJiu@kern.co.ca.us Pazz7432

11 Mialing Cart, Mart 4000 Hugo St., Bakersfield, CA, 93309 cart@gmail.com Randsfgh

12 UglyOrg Brown, Orge 5500 Truxtun Ave., Saint Clair, MI, 48079 haaaar@yahoo.net 1j4788g

13 OtherPer Van, Big 200 Camino Media, Bakersfield, CA, 12345 hoolahut@austin.edu blueish

14 Obamakin Obama, Barak 999 Olive Dr., SomeCity, D.C., 10000 Theprez@usa.gov Freeh3alth

15 drooling Robers, Trey 777 Luck St., Las Vegas, FL, 86095 TREY@treyz.us ohmygosh

16 toomany Que, Por 123 Fake St., Austin, Az, 55670 whyis@thishappening.org 49857v4

17 NOOOO Black, Jack 688 Verners Ave., ID, 88809 blackJ@aol.com H000ti3

18 tokcin Mama, Joe 45247 Golden Gate Pkwy., Toledo, OH, 46709 tokcin@ohio.oh Joebos87

1 Nkott0 Kott, Nicholas 1920 Hugo St., Bakersfield, CA, 93308 Nkott0@gmail.com Alfred0

Total number of rows retrieved: 18

**Gambler**

SQL query executed:

desc NK\_Gambler

Name Null? Type

----------------------------------------- -------- -------------------------

USERID NOT NULL NUMBER(10)

ROUTINGNO NOT NULL VARCHAR2(9)

BANKACCTNO NOT NULL VARCHAR2(20)

SQL query executed:

select \* from NK\_Gambler

USERID ROUTINGNO BANKACCTNO

---------- --------- --------------------

1 332484215 704654654

2 194650252 605465

3 956831478 54480

4 21859623 468405210

5 540405465 584584

6 465406545 5404654

7 332484215 455404654

8 970321454 980051

9 2415547 4654

10 224086752 6540644

Total number of rows retrieved: 10

**Bookie**

SQL query executed:

desc NK\_Bookie

Name Null? Type

----------------------------------------- -------- -------------------------

USERID NOT NULL NUMBER(10)

POSITION NOT NULL VARCHAR2(20)

BOOKID NOT NULL NUMBER(10)

SQL query executed:

select \* from NK\_Bookie

USERID POSITION BOOKID

---------- -------------------- ----------

11 Sys Admin 1

12 Sys Admin 2

13 Sys Admin 3

14 Sys Admin 4

15 Sys Admin 5

16 Sys Admin 6

17 Sys Admin 7

Total number of rows retrieved: 7

**Book**

SQL query executed:

desc NK\_Book

Name Null? Type

----------------------------------------- -------- -------------------------

BOOKID NOT NULL NUMBER(10)

BOOKNAME NOT NULL VARCHAR2(50)

ADDRESS NOT NULL VARCHAR2(75)

BANK NOT NULL NUMBER(16,2)

SQL query executed:

select \* from NK\_Book

BOOKID BOOKNAME ADDRESS BANK

---------- -------------------------------------------------- --------------------------------------------------------------------------- ----------

1 Mirage 3400 Las Vegas Boulevard South, Las Vegas, NV 24000000

2 Bellagio 3600 Las Vegas Blvd South, Las Vegas, NV 38000000

3 MGM Grand 3799 S. Las Vegas Blvd., Las Vegas, NV 300000000

4 Venetian Venetian, 3355 Las Vegas Blvd S, Uninc, NV 220000000

5 Caesars Palace 3570 Las Vegas Blvd South, Las Vegas, NV 700000000

6 Wynn 3131 Las Vegas Blvd. South, Las Vegas, NV 1100000000

7 Luxor 3900 Las Vegas Blvd, S, Las Vegas, NV 550000000

Total number of rows retrieved: 7

**Team**

SQL query executed:

desc NK\_Team

Name Null? Type

----------------------------------------- -------- -------------------------

TEAMID NOT NULL NUMBER(10)

TEAMNAME NOT NULL VARCHAR2(25)

CITY NOT NULL VARCHAR2(25)

STATE NOT NULL VARCHAR2(25)

RECORD NOT NULL VARCHAR2(25)

STREAK NUMBER(10)

SQL query executed:

select \* from nk\_team

TEAMID TEAMNAME CITY STATE RECORD STREAK

---------- ------------------------- ------------------------- ------------------------- ------------------------- ----------

1 Lions Detroit MI 1-4-0 -1

2 Chargers San Diego CA 1-4-0 1

3 Bills Buffalo NY 0-5-0 -5

4 Texans Houston TX 4-1-0 2

5 Dolphins Miami FL 3-2-0 2

6 Colts Indianapolis IN 4-1-0 2

7 Bengals Cincinnati OH 2-2-0 -1

9 Patroits Foxboro MA 3-2-0 1

10 Packers Green Bay WI 2-3-0 -3

8 Giants New York NY 3-2-0 2

Total number of rows retrieved: 10

**Game**

SQL query executed:

desc NK\_Game

Name Null? Type

----------------------------------------- -------- ----------------------------

GAMEID NOT NULL NUMBER(10)

HTEAM NOT NULL NUMBER(10)

ATEAM NOT NULL NUMBER(10)

HOMESCORE NOT NULL NUMBER(3)

AWAYSCORE NOT NULL NUMBER(3)

GDATETIME DATE

GAMETYPE NOT NULL VARCHAR2(15)

WEATHER NOT NULL VARCHAR2(50)

SQL query executed:

select \* from NK\_Game

GAMEID HTEAM ATEAM HOMESCORE AWAYSCORE GDATETIME GAMETYPE WEATHER

---------- ---------- ---------- ---------- ---------- --------------- --------------- --------------------------------------------------

1 1 6 10 32 15-AUG-10 pre-season 95F/Sunny

2 2 7 24 23 15-AUG-10 pre-season 75F/Indoors

4 5 10 3 21 21-SEP-10 regular season 95F/Sunny

5 3 7 32 10 08-OCT-10 regular season 85F/Sunny

6 1 2 23 24 15-OCT-10 regular season 55F/Windy

8 6 3 7 13 15-OCT-10 regular season 80F/Cloudy

9 7 4 21 3 15-OCT-10 regular season 105F/Sunny

3 3 1 13 7 30-JAN-10 super-bowl 19F/Snow

7 5 8 12 56 15-OCT-10 regular season 65F/Rain

Total number of rows retrieved: 9

**Bet**

SQL query executed:

desc NK\_Bet

Name Null? Type

----------------------------------------- -------- -------------------------

BETID NOT NULL NUMBER(10)

BOOKID NOT NULL NUMBER(10)

USERID NOT NULL NUMBER(10)

GAMEID NOT NULL NUMBER(10)

TEAMID NOT NULL NUMBER(10)

AMOUNT NOT NULL NUMBER(9,2)

BDATETIME DATE

BTYPE NOT NULL VARCHAR2(11)

WIN NUMBER(1)

SQL query executed:

select \* from nk\_bet

BETID BOOKID USERID GAMEID TEAMID AMOUNT BDATETIME BTYPE WIN

---------- ---------- ---------- ---------- ---------- ---------- --------------- ----------- ----------

1 1 1 1 6 20 04-APR-10 money line 1

2 2 2 2 7 100 24-MAY-10 totals 0

3 3 3 3 3 1 30-JAN-10 spread 1

4 4 4 3 3 25 14-FEB-10 money line 1

5 2 5 4 5 88 29-MAR-10 spread 0

6 3 6 5 7 2000 19-SEP-10 money line 0

7 1 7 6 1 200 14-OCT-10 totals 0

8 5 8 7 8 350 10-OCT-10 money line 1

9 6 9 8 3 75 14-OCT-10 spread 1

10 7 10 9 4 20 14-OCT-10 totals 0

Total number of rows retrieved: 10

**Odds\_On**

SQL query executed:

desc nk\_odds\_on

Name Null? Type

----------------------------------------- -------- ----------------------------

BOOKID NOT NULL NUMBER(10)

GAMEID NOT NULL NUMBER(10)

TEAMID NOT NULL NUMBER(10)

POINTSPREAD NOT NULL NUMBER(3)

SQL query executed:

select \* from nk\_odds\_on

BOOKID GAMEID TEAMID POINTSPREAD

---------- ---------- ---------- -----------

1 1 6 1

2 1 1 3

3 1 1 3

4 1 1 3

5 1 1 3

6 1 1 7

7 1 1 4

1 2 7 7

2 2 7 7

3 2 7 7

4 2 7 7

5 2 7 7

6 2 7 7

7 2 7 7

1 3 3 2

2 3 3 0

3 3 3 2

4 3 3 3

5 3 3 3

6 3 3 4

7 3 3 2

1 4 10 1

2 4 10 1

3 4 10 2

4 4 10 4

5 4 10 2

6 4 10 1

7 4 5 3

1 5 3 3

2 5 3 3

3 5 3 3

4 5 3 3

5 5 3 4

6 5 3 2

7 5 3 7

1 6 1 0

2 6 1 0

3 6 2 0

4 6 2 0

5 6 1 0

6 6 2 0

7 6 1 1

1 7 8 10

2 7 8 10

3 7 8 10

4 7 8 10

5 7 8 9

6 7 8 10

7 7 8 10

1 8 6 3

2 8 6 3

3 8 6 3

4 8 6 3

5 8 6 3

6 8 6 3

7 8 6 3

1 9 7 2

2 9 7 4

3 9 7 2

4 9 7 3

5 9 7 3

6 9 7 7

7 9 7 3

Total number of rows retrieved: 63

1. **SQL Queries**

**Select teams that have won more than 1 game as the away team.**

SQL query executed:

select \*

from nk\_game g, nk\_team t

where g.ateam = t.teamid and g.awayscore > g.homescore and

exists( select \*

from nk\_game g2, nk\_team t2

where g2.ateam = t2.teamid and g2.awayscore > g2.homescore and g2.gameid <> g.gameid and t.teamid = t2.teamid

)

Total number of rows retrieved: 0

0 row(s) affected.

**Select the gamblers that have bet on more than one game.**

SQL query executed:

select u.userid,u.fullname, count(\*)

from nk\_users u inner join nk\_gambler g on u.userid = g.userid

inner join nk\_bet b on u.userid = b.userid

group by u.userid,u.fullname

having count(\*) > 1

USERID FULLNAME COUNT(\*)

---------- -------------------------------------------------- ----------

1 Kott, Nicholas 2

Total number of rows retrieved: 1

**Select the largest winnings for games on 10/15/2010.**

SQL query executed:

select max(b.amount)

from nk\_users u inner join nk\_gambler g on u.userid = g.userid

inner join nk\_bet b on u.userid = b.userid

inner join nk\_game game on b.gameid = game.gameid

where to\_char(game.gdatetime, 'DD-Mon-YY') = '15-Oct-10' and b.win = 1

group by to\_char(game.gdatetime, 'DD-Mon-YY')

MAX(B.AMOUNT)

-------------

350

Total number of rows retrieved: 1

**Select gamblers who have open bets.**

SQL query executed:

select u.\*

from nk\_users u inner join nk\_gambler g on u.userid = g.userid

where exists ( select \* from nk\_bet b where b.userid = u.userid and b.win is null )

Total number of rows retrieved: 0

0 row(s) affected.

**Select books that have never had a bet.**

SQL query executed:

select b.bookname

from nk\_book b

where not exists ( select \* from nk\_bet bet where bet.bookid = b.bookid )

BOOKNAME

--------------------------------------------------

Crooks

Total number of rows retrieved: 1

**Select gamblers that have never had a winning bet.**

SQL query executed:

select u.userid,u.fullname

from nk\_users u inner join nk\_gambler g on u.userid = g.userid

where not exists ( select \* from nk\_bet bet where bet.userid = u.userid and bet.win = 1 )

USERID FULLNAME

---------- --------------------------------------------------

5 Wang, Huaqing

6 Cardenas, Jorge

10 Jiu, Pierson

2 Cat, Sawyer

7 Rutledge, Thomas

Total number of rows retrieved: 5

**Select gamblers that have only placed bets over $1,000.00.**

SQL query executed:

select u.userid,u.fullname

from nk\_users u inner join nk\_gambler g on u.userid = g.userid

where not exists ( select \* from nk\_bet bet where bet.userid = u.userid and bet.amount <= 1000 )

USERID FULLNAME

---------- --------------------------------------------------

6 Cardenas, Jorge

Total number of rows retrieved: 1

**Select gamblers who have won more than once.**

SQL query executed:

select u.userid,u.fullname, count(\*)

from nk\_users u inner join nk\_gambler g on u.userid = g.userid

inner join nk\_bet b on u.userid = b.userid

where b.win = 1

group by u.userid,u.fullname

having count(\*) > 1

USERID FULLNAME COUNT(\*)

---------- -------------------------------------------------- ----------

1 Kott, Nicholas 2

Total number of rows retrieved: 1

**Select the game that has the largest bet on it.**

SQL query executed:

select g.gameid,g.gametype

from nk\_game g inner join nk\_bet b on g.gameid = b.gameid

where b.amount = ( select max(b2.amount) from nk\_bet b2 )

GAMEID GAMETYPE

---------- ---------------

5 regular season

Total number of rows retrieved: 1

# Phase IV: Implementation of Relational Database

## Common Features in Oracle PL/SQL and MS Trans-SQL

PL/SQL and T-SQL are very similar in operability, however have a few key differences that are mostly syntactical. Both SQL languages call their database objects the same: tables, views, procedures, functions, triggers. The operations you can execute are the same for both, and the differences are actually quite small.

One of the more important differences is that the T-SQL’s RAISEERROR command does not break the flow of the procedure. It merely returns an error string or message but still returns normally. The PL/SQL’s raise\_application\_error throws an exception, exits the stored procedure, and rolls back to an implicit savepoint at the beginning of the stored procedure. The stored procedure, in general, remains the same in both languages because the user requirements of the stored procedure are vastly similar.

The purpose of a stored subprogram is to execute a query efficiently and without having to construct the entire query for each execution. A stored subprogram accepts parameters, which it then uses to perform an operation such as inserting, updating, and deleting. Also, it is similar to object oriented programming in that it can allow users functionality without giving them the code implemented in the function. One of the advantages of this over dynamic SQL are that it allows the SQL query to be built on the back end, as opposed to burdening the front end with tedious string allocations. Also, it makes the database more secure from SQL injections. However, with good programming those can be avoided in dynamic SQL as well.

## Oracle PL/SQL

The basic PL/SQL program structure breaks down into three distinct parts:

* Declaration (keyword: “IS”) – this section is used to declare any local variables, cursors, or user-defined exceptions.
* Execution (keywords: “BEGIN” and “END”) – this section contains the code that operates on/using the data in the declaration section.
* Exception (keyword “EXCEPTION”) – this section contains error handling procedures.

**Format**:

<TYPE> <Name> IS

BEGIN -- executable part starts here

[EXCEPTION]

END;

**Control Statements**:

The control statements give instructions as to which code should be executed at runtime. They can be like the ‘if’ statement and for statement in traditional languages such as C++ and Java.

IF --true/false condition

THEN -- statement

ELSEIF -- true/false condition

THEN -- statement

ENDIF;

[<<label\_name>>]

CASE selector

WHEN expression1 THEN sequence\_of\_statements1;

WHEN expression2 THEN sequence\_of\_statements2;

...

WHEN expressionN THEN sequence\_of\_statementsN;

[ELSE sequence\_of\_statementsN+1;]

END CASE [label\_name];

LOOP

EXIT WHEN -- can be used similar to a “break” command

END LOOP;

WHILE condition LOOP

sequence\_of\_statements

END LOOP;

FOR counter IN [REVERSE] lower\_bound..higher\_bound LOOP

sequence\_of\_statements

END LOOP;

**Cursors**:

Cursors could be described as pointers that are assigned to rows from a select statement. They look like the following:

CURSOR cursor\_name (parameter\_list)

IS

SELECT\_statement;

**Stored Procedure:**

Stored procedures execute a set of commands using optional variables in a create/insert/select/update/delete statement. The format of a stored procedure is:

CREATE [OR REPLACE] PROCEDURE <NAME>

[parametername] [datatype]

IS

-- Declare constants and variables in this section.

-- Example: <Variable Identifier> <DATATYPE>

-- <Variable Identifier> CONSTANT <DATATYPE>

-- varEname VARCHAR2(40);

-- varComm REAL;

-- varSalary CONSTANT NUMBER:=1000;

-- comm\_missing EXCEPTION;

BEGIN -- executable part starts here

-- Write PL/SQL and SQL statements to implement the processing logic

-- of subprogram. Example:

-- SELECT ENAME,

-- COMM

-- INTO varEname,

-- varComm

-- FROM EMP

-- WHERE EMPNO = 7369;

--

-- IF varComm IS NULL THEN

-- RAISE comm\_missing;

-- END IF;

[EXCEPTION] -- exception-handling part starts here

-- WHEN comm\_missing THEN

-- dbms\_output.put\_line('Commision is NULL');

END;

**Stored Function**:

A stored function operates much like a stored procedure would, however it always returns a declared variable upon completion. The stored function format:

CREATE [OR REPLACE] FUNCTION <NAME> (

[parametername] IN [datatype]) RETURN [datatype] IS

-- Declare constants and variables in this section.

-- Example: <Variable Identifier> <DATATYPE>

-- <Variable Identifier> CONSTANT <DATATYPE>

-- varEname VARCHAR2(40);

-- varComm REAL;

-- varSalary CONSTANT NUMBER:=1000;

-- comm\_missing EXCEPTION;

BEGIN -- executable part starts here

-- Write PL/SQL and SQL statements to implement the processing logic

-- of subprogram. Example:

-- SELECT ENAME,

-- COMM

-- INTO varEname,

-- varComm

-- FROM EMP

-- WHERE EMPNO = 7369;

--

-- IF varComm IS NULL THEN

-- RAISE comm\_missing;

-- END IF;

RETURN <returnvalue>;

-- EXCEPTION -- exception-handling part starts here

-- WHEN comm\_missing THEN

-- dbms\_output.put\_line('Commision is NULL');

END;

**Packages**:

Packages can execute a number of procedures and functions together. Prototypes of each function and/or procedure are required.

CREATE [OR REPLACE] PACKAGE package\_name

[AUTHID {CURRENT\_USER | DEFINER}]

{IS | AS}

[PRAGMA SERIALLY\_REUSABLE;]

[collection\_type\_definition ...]

[record\_type\_definition ...]

[subtype\_definition ...]

[collection\_declaration ...]

[constant\_declaration ...]

[exception\_declaration ...]

[object\_declaration ...]

[record\_declaration ...]

[variable\_declaration ...]

[cursor\_spec ...]

[function\_spec ...]

[procedure\_spec ...]

[call\_spec ...]

[PRAGMA RESTRICT\_REFERENCES(assertions) ...]

END [package\_name];

[CREATE [OR REPLACE] PACKAGE BODY package\_name {IS | AS}

[PRAGMA SERIALLY\_REUSABLE;]

[collection\_type\_definition ...]

[record\_type\_definition ...]

[subtype\_definition ...]

[collection\_declaration ...]

[constant\_declaration ...]

[exception\_declaration ...]

[object\_declaration ...]

[record\_declaration ...]

[variable\_declaration ...]

[cursor\_body ...]

[function\_spec ...]

[procedure\_spec ...]

[call\_spec ...]

[BEGIN

sequence\_of\_statements]

END [package\_name];]

**Triggers**:

Triggers are useful operations that execute upon a specified database event. This means we can have a trigger that changes values in related tables when a value in the event table is manipulated.

CREATE [OR REPLACE] TRIGGER <Name>

<BEFORE,AFTER> <INSERT,[OR]UPDATE,[OR]DELETE> ON <tablename>

FOR EACH ROW

[WHEN] [condition]

DECLARE

[variable(s)]

BEGIN

<code>

END;

## Oracle PL/SQL Subprogram

**Stored Procedures**

NK\_INSERTGAMBLER

This stored procedure takes accepts all the necessary information required to create a gambler record in this database. Of note, a userID is not required because I created a sequence to handle that. Also, notice that a record is inserted into the NK\_USERS table first so as to avoid conflict with the NK\_GAMBLER table’s foreign key constraint.

CREATE PROCEDURE "NK\_INSERTGAMBLER" (

"UNAME" IN VARCHAR2,

"FNAME" IN VARCHAR2,

"ADDY" IN VARCHAR2,

"EADD" IN VARCHAR2,

"PWORD" IN VARCHAR2,

"ROUTNO" IN VARCHAR2,

"BACCTNO" IN VARCHAR2) IS

nuserid number;

BEGIN

select nk\_seqUsers.nextval into nuserid from dual;

INSERT INTO NK\_USERS(userid,username,fullname,address,emailaddress,password)

VALUES(nuserid,UNAME,FNAME,ADDY,EADD,PWORD);

INSERT INTO NK\_GAMBLER(userid,routingno,bankacctno)

VALUES(nuserid,routno,bacctno);

commit;

END;

NK\_DELETEGAMBLER

This stored procedure accepts a userId as a parameter and removes the corresponding records from its table. Note, again, that the record is first removed from the NK\_GAMBLER table to remain in compliance with the foreign key constraint. Also note that the user’s bets are not removed from the system, so that they can remain for historical reporting.

CREATE PROCEDURE "NK\_DELETEGAMBLER" (

"DUSERID" IN NUMBER) IS

BEGIN -- executable part starts here

DELETE FROM nk\_gambler

WHERE userID = DUSERID;

DELETE FROM nk\_users

WHERE userID = DUSERID;

commit;

END;

**Stored Function**

NK\_AVGGAMBLERBET

This function accepts a userID as a parameter and returns a scalar value of the average bet made by the user. This task was implemented using an aggregate function so as to reduce the number of local variables declared in the function and keep the code as simple and efficient as possible.

CREATE FUNCTION "NK\_AVGGAMBLERBET" (

"GUSERID" IN NUMBER) RETURN NUMBER IS

avgamount number(9,2);

BEGIN -- executable part starts here

SELECT AVG(amount)

INTO avgamount

FROM NK\_BET

WHERE userID = GUSERID;

RETURN avgamount;

END;

**Trigger**

NK\_USERS\_UPDELTRIGGER

This trigger executes whenever there is an update or delete on a record in the NK\_USERS table. After this event, the trigger enters the old and new values of the userID and userName into NK\_LOGTABLE where the UPDATED field is defaulted to SYSDATE.

CREATE TRIGGER "NK\_USERS\_UPDELTRIGGER"

AFTER

UPDATE OF "USERID", "USERNAME" OR DELETE

ON "NK\_USERS"

FOR EACH ROW

DECLARE

BEGIN

INSERT INTO NK\_LOGTABLE(OLDVAL,NEWVAL)

VALUES(:old.USERID + ',' + :old.USERNAME, :new.USERID + ',' + :new.USERNAME);

END;

# Phase V: GUI Design and Implementation

## Daily User Activities

There are two basic user groups who will utilize the proposed database. These two groups fall into the generalized relation NK\_USERS, and are further described as gamblers and bookies. The gamblers will use the database from a client perspective, while the bookies will maintain a role that is more akin to an administrator.

### Gamblers

The gambler’s most important activity is being able to place a bet. Without this activity there can be no effective business. They will need a secure way of placing a bet in order to prevent fraudulent bets from being inserted into the database. They also need to be able to view their current and past bets so that they can manage their betting. Also, while placing a bet, they need the ability to choose from a list games and of various casinos that are offering to accept bets on a specific game.

### Bookies

The bookies, like the gamblers, need a secure way of logging in. This is paramount because each casino will have to have one or more people with access to manage their information in the database without allowing them to access other casino’s information and vice versa. The bookies will need to be able to see the number of bets they are currently holding, and the potential loss of capitol if all of those bets were to go to their respective gamblers. They also need a list of gamblers who are due payments and monthly expenditure and revenue totals. Bookies also need the ability to set the favorite and odds on any pending games.

## Relations, Views, and Subprograms

Most of the relations will be involved in the daily activities of the database, which is typical of a database that does not keep historical records of every relation. NK\_BOOK, NK\_GAMBLER, and NK\_TEAM have a small likelihood of being accessed and operated on in daily use. NK\_BOOKIE, unlike NK\_GAMBLER, will be access as it will have an impact on the bookie user group’s privileges. The following are the relations involved in daily activities:

* NK\_BET
* NK\_BOOKIE
* NK\_GAME
* NK\_ODDS\_ON
* NK\_USERS

Views provide a useful resource for storing a frequently used query in the database. Databases that are heavily normalized, such as this one, can benefit from these as they can de-normalize the database’s information into a comprehensive state. All of the views created in this database de-normalize data from several relations to create specific interpretations of the information. Some also modify field values, or make use of aggregate functions to calculate new fields. The following is the list of views used in the database implementation:

* NK\_VW\_AVAILABLEGAMES
* NK\_VW\_CURRENTBETS
* NK\_VW\_HOTGAMES
* NK\_VW\_OLDBETS
* NK\_VW\_POINTSPREAD

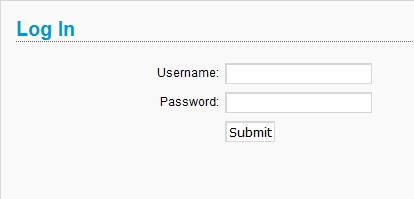
Subprograms enabled the implementation to avoid using dynamic SQL, which is inherently vulnerable to SQL injections. Also, in the case of NK\_INSERTGAMBLER and NK\_INSERTBOOKIE, these procedures execute two SQL statements. By only passing the procedure a set of parameters, we avoid unnecessary traffic between the web server and database. For this project is made sense to use procedure to carry out insertion, updating, and deletion. The list of procedures are as follows:

* NK\_DELETEBOOKIE
* NK\_DELETEGAMBLER
* NK\_INSERTBET
* NK\_INSERTBOOKIE
* NK\_INSERTGAMBLER

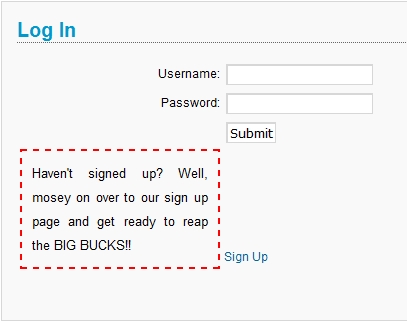
## Application Screen Shots

For this project, along with any other commercial website, a very important part of development is the graphic design. Unfortunately, my talents in this field are questionable at best. Therefore, the decision was made to take a css design from the website themebot.com. The css template was implemented, along with a few changes as follows:

**Login**

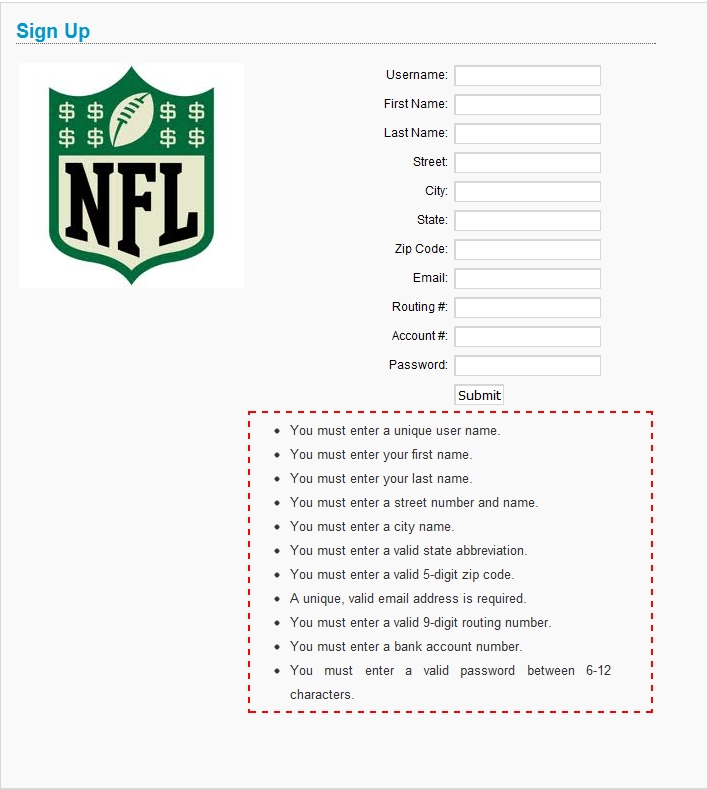


The login page is what any user is directed to if they have not already logged in. It is meant to be simple, clean, and concise. The username and password are checked against existing records in the NK\_USERS relation. If they are not found they receive notification:



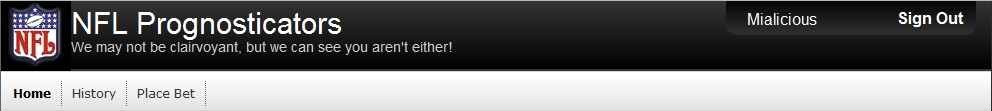
As you can see, the error message is surrounded by a red dashed line in order to grab the user’s attention. The message suggests that if they have not signed up, they may do so by clicking the link to the right (the link is also present in the navigation not pictured).

**Sign Up**



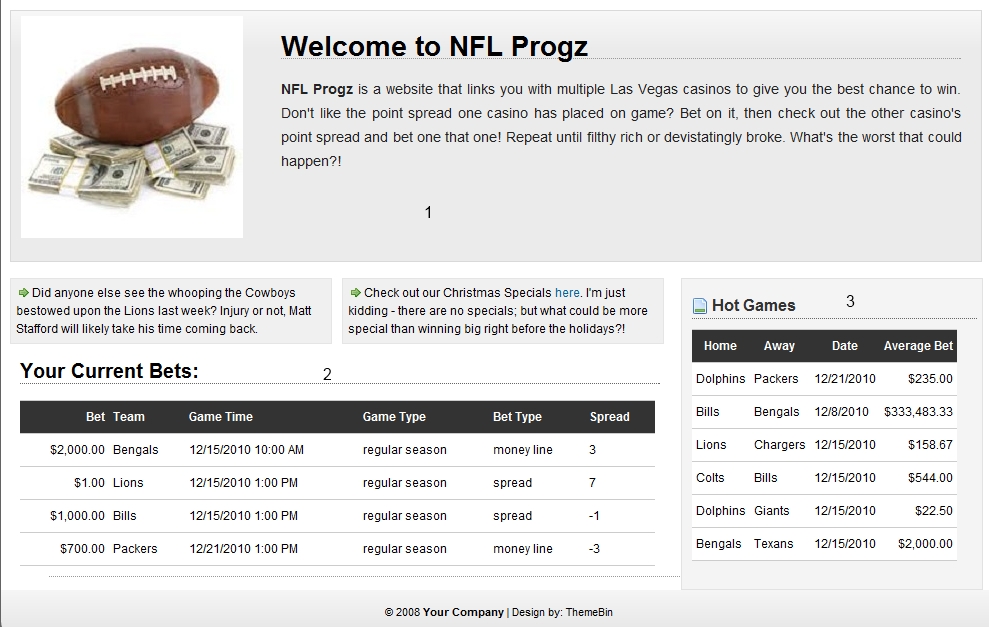
The signup page requires all the information necessary to insert a gambler record into the database. Sign up for a bookie requires offline contact due to security concerns. Invalid entries prompt the specific error to be shown in the list below the submit button. If the data passes the validation, then the record is inserted and the user is redirected to the login page.

**Menu**



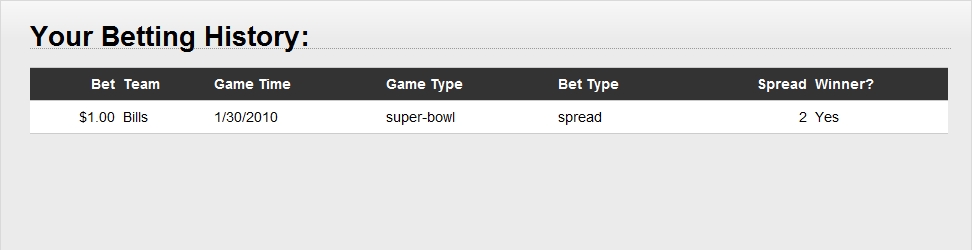
This menu is displayed once a user has logged into the website. In the top right corner, the username (“Mialicious”) is displayed along with a sign out link. In the bottom left corner, links to “home”, “history”, and “place bet” are listed for the user to navigate.

**Index**



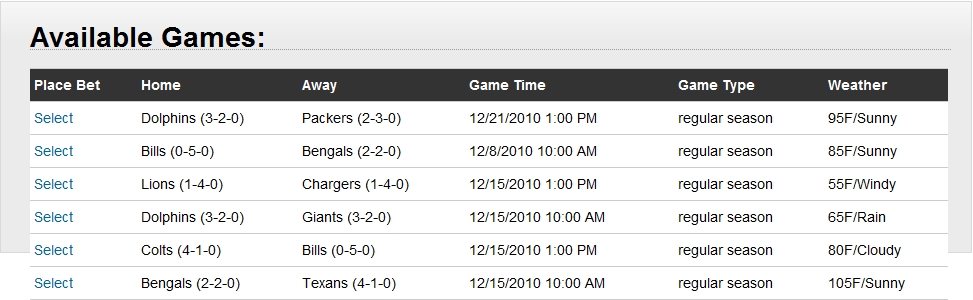
The index, or “Home”, page quickly displays useful information to the gambler. In section 1, we have a description of the website, along with two areas for admin to post messages or commentary to the user. While this area is not included in the database as a relation, it would prove to be a valuable addition. In section 2, the view NK\_VW\_CURRENTBETS is displayed using a Gridview object. Formatting is done to the bet and game time columns for readability. In section 3, another view (NK\_VW\_HOTGAMES) is displayed with another Gridview object. Formatting is done on the date and average bet columns for readability.

**History**

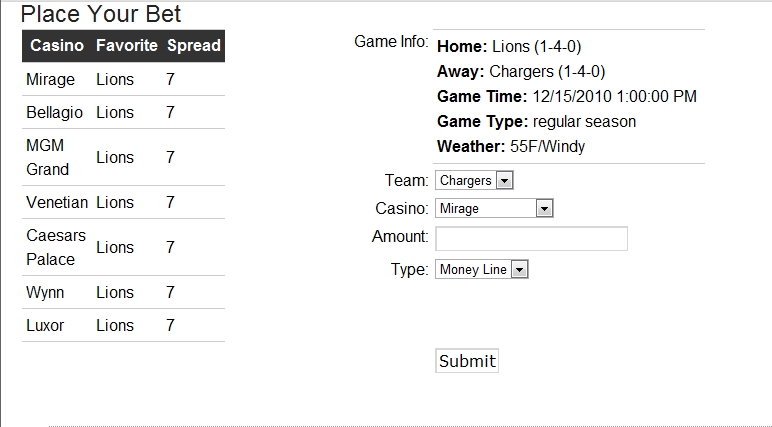


The history link brings the gambler to a Gridview of their bets which have been resolved. The view is NK\_VW\_OLDBETS, and formatting has been done on the bet and game time columns.

**Place Bet**



The place bet link takes the gambler to this page. Using a gridview, the view NK\_VW\_AVAILABLEGAMES is displayed along with a template item in the “Place Bet” column. Home and Away are also template items, which display the team name with their record in parenthesis. A tooltip attribute is also placed on these columns displaying the team’s current streak. If they user clicks the link, it redirects them to the following page with the gameID in the query string.



On the left side of the screen is the NK\_VW\_POINTSPREAD view displayed through another Gridview. On the top left, we have the game information from the NK\_VW\_AVAILABLEGAMES view displayed in a Formview object. The user make their choices in the dropdown boxes, and text box before hitting submit in order to place a bet.

## Code Description

The GUI design and implementation for this project was done using Microsoft Visual Studio 2010, and it is an ASP.NET 4.0 website. I used this development environment and language because it is what I use at my current place of employment, and they requested I do so in order to learn more about these tools. The code can be broken into three sections:

* Database objects
* HTML/XML
* ASP.NET

**Database Objects**

When implementing the design of the website, it became apparent that modularizing the interaction with the database would greatly improve the code. Therefore, I created two namespaces: DBObj and DBObjManager. DBObj classes are meant to make the processing of form information easier. Instead of passing all of the variables to the database procedure in the C# code of page, we create an object with the information. The object is then operated on with the corresponding DBObjManager class. This prevents changes from the database causing havoc on the C# code, and vice versa.

* DBUser

This class is a representation of the database NK\_USERS relation. It has all the relation’s attributes as member variables, and get() and set() methods to access those variables. This class is in the DBObj namespace.

* Gambler

This class is a representation of the databse NK\_GAMBLER relation. Like DBUser, it shares it’s corresponding relation’s attributes as member variables along with get() and set() methods. This class inherits DBUser, and is in the DBObj namespace.

* Bookie

Like the two before, this class is a representation of its corresponding database relation. It also inherits the DBUser class, and is in the DBObj namespace.

* UsersManager

This class contains some specific validation functions that deal with NK\_USERS. The methods can check if a username exists in the database, if an email exists in the database, or if a username and password combination are valid. This class is in the DBObjManager namespace.

* GamblerManager

This class handles the Gambler class object’s interaction with the database. It has methods to insert, retrieve, and delete Gamblers within the database. It also performs operations that a gambler needs to interface with the GUI such as inserting a bet, or getting casino information. This is a static class and is in the DBObjManager namespace.

* BookieManager

This class inserts, retrieves and deletes records from the NK\_BOOKIE relation. It handles all operation done upon the Bookie class objects, however this side of the project was not developed. This is a static class in the DBObjManager namespace.

**HTML/XML**

The html/xml programming consisted of what you would see in the web page. As I mentioned before, I utilized a free css template that was then modified and utilized to fit this project. Therefore, the skeleton of what I used was already coded, however the content of each page was coded by myself. This was perhaps the most difficult and important part of the project. I had to make sure each page was clearly and intuitively laid-out for the user. If a page were to appear convoluted to a user, they could potentially stop using the web site and leave for the competition.

**ASP.NET**

This section of the code handled all of the form validation, and display objects in the website. The validation was constructed using the business rules formed in the planning of the database. Most of the display object were Gridview objects that display the information received from their data source. Most of these data sources were ObjectDataSources, which referred to methods developed in the GamblerManager class. These ASP object would pass an argument to the class method and would have a datatable object returned. Formatting was done of the Gridviews to make the display more visually appealing to the user.

**Major Features**

This website has Forms authentication implemented as a security feature. This was written into the web.config file and also used to create a cookie in the login page. This provides users with a level of security to prevent fraudulent betting from occurring.

Another feature of this project is the ability for the user to get information presented to them in real-time in the form of tables. The user is able to view information on their current and previous bets, as well as information on available games.

Also, users are able to place new bets based on the previously mentioned information. They are able to select an available game, a casino, a team, and a bet type. This function is paramount to this website’s purpose.

## Development Process

The first and most difficult part of the developing an ASP.NET website with an Oracle database is the actual database connection. Surprisingly, there is not a good source of information on how to get a reliable connection between the two tools. Having completed the project, it now seems a trivial task, however at the time this was a very difficult step. Once I had connected to the database, the next step was to create classes to modularize the interaction with the database.

I created the database classes much in the style that we use at my place of work: we create classes to represent the relations in the database, and a static class to handle the transferring of these classes to and from the database. The result is a very clean way of grabbing form data and inserting it into the database. The next task was to get a template for the website.

As I have mentioned before, I used a free css template from thembot.com in order to make the website more visually appealing. In real-world applications, there are graphic designer that usually handle the aesthetics of the website while the code and functionality is left for the programmer. Nevertheless, I still needed to tweak some of the existing css and add some of my own to get the exact look that I wanted from this project. Once this was completed, I moved onto the login page.

For a site that handles a user’s financials as this one does, security is absolutely necessary. For this project I chose Forms authentication, because it seemed a simple and safe way of implementing security to the site. With a few entries into the web.config file, the security was in place. I was also able to create a cookie upon login to store the userID, which proved useful for presenting information directed at the particular user that has logged in. With this finished, I could then move on to page content.

The ASP.NET object used to display information were very easy to implement because I had created the class function to retrieve the information needed to display to the user. A few formatting modifications were needed to make the data clearer, as well as some template items. The template items combined some of the returned columns into one “item” to be displayed. The validation done was also easy using the CustomValidator and ValidationSummary objects. I was able to implement the business rules specified in planning the database, and convey error messages in a concise manner to the user.

## Conclusion

The development of this project benefitted greatly from my experiences working at the Kern County Auditor-Controller’s office. Implementation was carried out with ease using the methods of development I have learned there, as well as the database planning and development skills from this class. It is unfortunate that I was not able to fully complete the project, but for the time available there is a lot that was completed. The reason I took this class was for the unique understanding of query language that is achieved, but I certainly feel as though I’ve learned more than just that; the database planning skills used will undoubtedly prove useful in the workplace. Possibly the best reward out of this was finally figuring out how to set up IIS 7.5 to host this website from my home computer, so that I could send the link to my family in Michigan and have them see what I created.