Video Game Developer Database

For CS 342, Fall 2010

Travis Ragle
11/24/2010

TABLE OF CONTENTS

PHASE I

FACT FINDING AND INFORMATION GATHERING

FACT FINDING TECHNIQUES

While there are numerous differences between the structures of various video game development companies, there are also several common qualities. The wealth of information available through the internet gave us a strong foundation. In particular, both Wikipedia and specific company websites provide general and specific information on many different video game development companies. These served as our primary source for data.

METHODS USED TO GATHER DATA

In order to keep the database flexible, common traits among many companies were singled out as the primary foundation for our model. However, some specific traits were included wherever such limitation was either irrelevant or not a concern. The primary goal was to include data that would be useful to a variety of video game development companies.

INTRODUCTION TO ENTERPRISE

Video game development companies are organizations that typically take a game concept and develop it into software. A typical development company will consist of various programmers and artists as the core development team. Additionally, there are many jobs which may be outsourced or performed in-house, depending on the size of the company and specifics of a given project. Examples are game design, story, musical composition and concept art.

STRUCTURE OF THE ENTERPRISE

Many companies take a team approach, where a set of individuals with certain skills are grouped together as a project team. This team will work on a single game until completion. Afterward, the team will take a short vacation before starting development on a new game.

One potential problem with this approach is over-utilization and under-utilization of staff members. During the initial weeks of a new project, many staff members may often create work that will not be part of the final product. During the final weeks of a project, just prior to release, many staff members are asked to work a large amount of overtime.

An alternative is to have a collective pool of employees that are assigned to projects based on the project needs, rather than preset teams. Additionally, employees can be assigned to multiple projects as needed. The database application explained in this document proposes an implementation of this alternate approach.

MAJOR OBJECTS

The employee entity is one or two primary entities in this database. Basic information for each employee is included directly in this entity type, while the department and position entities round out the description of the employee.

The video game entity is the second of two primary entities in this database. Basic information about each video game release is included in this entity, while many other entities support the full description of a given video game. The genre, console and region entities add further information to each video game release. The phase and milestone entities help give a historical record of the work accomplished during a project. Finally, the series entity helps to group together related video game releases.

The relationship between the employee and video game entities is supported through the position entity.

DATA VIEWS AND OPERATIONS FOR USER GROUPS

Numerous views will be created to support various parts of the project team. A view of video game releases will be created that contains general, up to date information for each video game release such as genre, current phase, latest milestone and release date for each console and region. Additionally, historical views will be created to show a timeline of phases and milestones reached for any particular video game. An employee view will be created that list all current positions filled for every video game.

A set of operations will be created to allow video game projects to be added to the video game entity and other supporting entities. Another set of operations will allow such things as milestones and phases to be updated, as needed.

CONCEPTUAL DATABASE DESIGN

ENTITY SET DESCRIPTION

EMPLOYEE

Description: Stores basic information for each employee, such as age and name. An entry is inserted only when a new employee is hired, and it is never deleted except in extreme circumstances. However, it may be updated when an employee's address or name changes. Additionally, each employee will be part of a department which will generalize his or her skill set, and each employee can hold one or more positions on a given video game at any time.

Candidate keys:

1. EmployeeID (primary key)
2. Name

This is a strong entity.

Fields to be indexed:

1. EmployeeID
2. Name

Attribute Name	EmployeeID	Name	Birthdate	Sex	Address
Description	Id column for internal use.	First name, Last name, Middle name(s).	Date of birth	Male or Female	Street address, City, State, Zip code.
Domain / Type	Integer	Full Name (3 strings)	Date	M or F	Address (5 strings)
Value-Range	0 to Max Integer	Any	Range of Date type	M or F	Any
Default Value	(Max ID) + 1	None	None	None	None
Nullable	No	Yes (Middle name only)	No	No	Yes (second street address string only)
Unique	Yes	Yes	No	No	No
Single / Multiple	Single	Single	Single	Single	Single
Simple / Composite	Simple	Composite	Composite	Simple	Composite

DEPARTMENT

Description: Stores basic information for each department, such as the name and hierarchical organization. An entry is inserted only when a new department is formed or subdivided into further departments. A department may have a name change or reorganization, but updates and deletions are infrequent. Additionally, each department will have one or more employees. The department entity is the primary means for grouping employees by skill sets.

Candidate keys:

1. DepartmentID (primary key)

This is a strong entity.

Fields to be indexed:

1. DepartmentID

Attribute Name	DepartmentID	Name	Description
Description	Id column for internal use.	Name of department.	Description of department
Domain / Type	Integer	String	String
Value-Range	0 to Max Integer	Any	Any
Default Value	(Max ID) + 1	None	None
Nullable	No	No	Yes
Unique	Yes	No	No
Single / Multiple	Single	Single	Single
Simple / Composite	Simple	Simple	Simple

POSITION

Description: Stores basic information for each position an employee holds on a video game project, such as title and hierarchical organization. Lead and subordinate positions are separate entities. An entry is inserted only when a new type of position is created or an existing position is further subdivided. Positions will only be updated or deleted during phases of reorganization, which are infrequent. An employee can hold one or more positions on any video game. Each position can manage or lead one or more other positions. The position entity is the primary means for a hierarchical representation of the staff for any given video game. Additionally, this entity forms a relationship with the employee and video game to store a work history.

Candidate keys:

1. PositionID (primary key)

This is a strong entity.

Fields to be indexed:

1. PositionID

Attribute Name	PositionID	Title	Description
Description	Id column for internal use.	Position title.	Description of position.
Domain / Type	Integer	String	String
Value-Range	0 to Max Integer	Any	Any
Default Value	(Max ID) + 1	None	None
Nullable	No	No	Yes
Unique	Yes	No	No
Single / Multiple	Single	Single	Single
Simple / Composite	Simple	Simple	Simple

VIDEO GAME

Description: A video game is the sole type of project for any employee in the database. This entity stores information for a specific release of a given video game. However, most of the information describing both a video game and the project employees are available through relationships to other entities. This entity will contain such information as the title, release hierarchy and website. Entries will be inserted several times a year, as new projects are created. Although this entity itself will be infrequently updated, the relationships to other entities will be updated several times a year until the specific video game is released. Entries will not be deleted, except in extreme cases.

Each video game will have multiple employees filling one or more positions during various phases of development. During the course of development, various milestones will be recorded as an indication of progress. Each video game belongs to one genre, and is released on one or more consoles in one or more regions. Each video game can have one or more re-releases. The video game entity is related to almost every other entity in the database.

Candidate keys:

1. VideogameID (primary key)

This is a strong entity.

Fields to be indexed:

1. VideogameID
2. Title

Attribute Name	VideogameID	Title	Description	ReleaseDate	Website
Description	Id column for internal use.	Title of video game.	Description of video game	Date of release	Website
Domain / Type	Integer	String	String	Date	String
Value-Range	0 to Max Integer	Any	Any	Any	Any
Default Value	(Max ID) + 1	None	None	None	None
Nullable	No	No	Yes	Yes	Yes
Unique	Yes	No	No	No	No
Single / Multiple	Single	Single	Single	Single	Single
Simple / Composite	Simple	Simple	Simple	Composite	Simple

PHASE

Description: Each video game will be in exactly one phase during any given point of its life cycle, which helps indicate development progress. This entity will store basic information such as the name of the phase and any hierarchical subdivisions. Changes to this entity set will only occur during company reorganization, which is infrequent.

Candidate keys:

1. PhaseID (primary key)
2. Name

This is a strong entity.

Fields to be indexed:

1. PhaseID

Attribute Name	PhaseID	Name	Description
Description	Id column for internal use.	Name of phase.	Description of the phase.
Domain / Type	Integer	String	String
Value-Range	0 to Max Integer	Any	Any
Default Value	(Max ID) + 1	None	None
Nullable	No	No	Yes
Unique	Yes	Yes	No
Single / Multiple	Single	Single	Single
Simple / Composite	Simple	Simple	Simple

MILESTONE

Description: Each video game will reach one or more milestones during the course of development. Each milestone is reached at most once, at which point it is given a date stamp. This entity will store basic information about the milestone, such as its name. Changes to this entity set will only occur during company reorganization, which is infrequent.

Candidate keys:

1. MilestoneID (primary key)
2. Name

This is a strong entity.

Fields to be indexed:

1. MilestoneID

Attribute Name	MilestoneID	Name	Description
Description	Id column for internal use.	Name of milestone.	Description of milestone
Domain / Type	Integer	String	String
Value-Range	0 to Max Integer	Any	Any
Default Value	(Max ID) + 1	None	None
Nullable	No	No	Yes
Unique	Yes	Yes	No
Single / Multiple	Single	Single	Single
Simple / Composite	Simple	Simple	Simple

GENRE

Description: Each video game will belong to exactly one genre, which classifies the video game type. This entity will hold genre names and subdivisions. Genres typically will not be updated or deleted. However, new subdivisions of existing genres may be added infrequently.

Candidate keys:

1. GenreID (primary key)
2. Name

This is a strong entity.

Fields to be indexed:

1. GenreID

Attribute Name	GenreID	Name	Description
Description	Id column for internal use.	Name of genre.	Description of genre.
Domain / Type	Integer	String	String
Value-Range	0 to Max Integer	Any	Any
Default Value	(Max ID) + 1	None	None
Nullable	No	No	Yes
Unique	Yes	Yes	No
Single / Multiple	Single	Single	Single
Simple / Composite	Simple	Simple	Simple

SERIES

Description: A collection of video games will belong to a single series, although each series will be divided into entries through its relationship to the video game entity. The series entity will hold the common, single name that describes all video games belonging to the series. Typically, entries will neither be updated nor deleted. However, new entries will be added any time a video game project is created that will not belong to an existing series, which will happen a few times a year.

Candidate keys:

1. SeriesID (primary key)
2. Name

This is a strong entity.

Fields to be indexed:

1. SeriesID

Attribute Name	SeriesID	Name
Description	Id column for internal use.	Name of series.
Domain / Type	Integer	String
Value-Range	0 to Max Integer	Any
Default Value	(Max ID) + 1	None
Nullable	No	No
Unique	Yes	Yes
Single / Multiple	Single	Single
Simple / Composite	Simple	Simple

CONSOLE

Description: Each video game will appear on one or more consoles. This entity will hold basic information for a given console, such as its name. Entries are added on an average of one or two a year. A console entity is never deleted, and will typically not be updated except for changes occurring around the initial release of the console. Each console will be released in multiple regions.

Candidate keys:

1. ConsoleID (primary key)
2. Name

This is a strong entity.

Fields to be indexed:

1. ConsoleID

Attribute Name	ConsoleID	Name
Description	Id column for internal use.	Name of the console.
Domain / Type	Integer	String
Value-Range	0 to Max Integer	Any
Default Value	(Max ID) + 1	None
Nullable	No	No
Unique	Yes	Yes
Single / Multiple	Single	Single
Simple / Composite	Simple	Simple

REGION

Description: Each video game will appear in one or more regions. This entity will hold basic information for a given region, such as its location. Entries are rarely added or updated, and never deleted. Each region will contain multiple consoles, and each console will have multiple video game releases.

Candidate keys:

1. RegionID (primary key)
2. Location

This is a strong entity.

Fields to be indexed:

1. RegionID

Attribute Name	RegionID	Location
Description	Id column for internal use.	Geographical location of the region
Domain / Type	Integer	String
Value-Range	0 to Max Integer	Any
Default Value	(Max ID) + 1	None
Nullable	No	No
Unique	Yes	Yes
Single / Multiple	Single	Single
Simple / Composite	Simple	Simple

RELATIONSHIP SET DESCRIPTION

WORKS ON

Description: An employee will work on many video games during his or her time with the company, filling one or more positions on each. Each video game will be the product of the work of many employees. This ternary relationship stores the work history for a given employee or video game. Both start date and end date attributes will be recorded in this relationship set, to keep a history of each time an employee joined and left a video game project, for a given position.

Entity sets involved: Employee, Video Game and Position

Mapping Cardinality: M..N..P

Descriptive Fields: Start Date, End Date.

Participation Constraint: Partial for employee, position and video game entity sets. This will allow a video game, position or an employee to be created before an assignment is decided.

POSITION (LEAD AND SUBORDINATE)

Description: This recursive relationship will define a hierarchy for the position entity set. Each position can have any number of subordinate positions.

Entity sets involved: Position

Mapping Cardinality: 1..N

Description Fields: None.

Participation Constraint: Partial for both lead and subordinate. The top level positions will not have any leads, while the bottom level positions will not have any subordinates.

EMPLOYEE BELONGS TO A DEPARTMENT

Description: Each employee can belong to exactly one department at any given time, but may belong to different departments during the course of an employment history. Both start date and end date attributes will be recorded in this binary relationship set, to keep a historical record of employment.

Entity sets involved: Employee, Department

Mapping Cardinality: M..N

Descriptive Fields: Start Date, End Date.

Participation Constraint: Both the employee and department entity sets have total participation. A department must have at least one employee to exist, while an employee must belong to a single department at any given time.

DEPARTMENT (GENERAL AND SPECIALIZED)

Description: This recursive relationship will define a hierarchy of departments and sub-departments. Each department can have any number of specialized sub-departments.

Entity sets involved: Department

Mapping Cardinality: 1..N

Descriptive Fields: None.

Participation Constraint: Partial for both general and specialized. The top level departments will not be part of any higher level general department, while the bottom level departments will not be divided into further sub-departments.

SERIES CONTAINS VIDEO GAMES

Description: Each series will contain one or more video game releases, and each video game will be part of exactly one series. This binary relationship set will contain an entry attribute that will aid in categorizing specific video games within a series.

Entity sets involved: Series, Video Game

Mapping Cardinality: 1..N

Descriptive Fields: Entry.

Participation Constraint: Both series and video game entity sets have total participation. For a series to exist, it must contain at least one video game. Each video game must belong to exactly one series.

VIDEO GAME (ORIGINAL AND RE-RELEASE)

Description: This recursive relationship set will define a hierarchy of additional releases for any video game release. After a video game is created, it may be released in additional regions or on additional consoles. Additionally, enhanced versions of a video game may be released.

Entity sets involved: Video Game

Mapping Cardinality: 1..N

Descriptive Fields: None

Participation Constraint: Each re-release must have total participation by definition. However, the original release will have partial participation, because re-releases are not mandatory.

VIDEO GAME IS IN A GENRE

Description: In this binary relationship set, each video game will have exactly one genre which will describe the video game type.

Entity sets involved: Video Game, Genre

Mapping Cardinality: N..1

Descriptive Fields: None

Participation Constraint: Both video game and genre entities have total participation. A video game must belong to exactly one genre. A genre must contain one or more video games to be part of this database.

GENRE (GENERAL AND SPECIFIC)

Description: This recursive relationship will define a hierarchy of genres and sub-genres. Each genre can be divided into any number of more specific sub-genres.

Entity sets involved: Genre

Mapping Cardinality: 1..N

Descriptive Fields: None.

Participation Constraint: Partial for both general and specific. The top level genres will not be part of any higher level general genres, while the bottom level sub-genres will not be divided into any further specific sub-genres.

CONSOLE IS IN A REGION

Description: In this binary relationship set, each console must be released in one or more regions. The relationship set will hold attributes for both a release date and website.

Entity sets involved: Console, Region

Mapping Cardinality: M..N

Descriptive Fields: Release Date, Website.

Participation Constraint: Both console and region entity sets have total participation. A console must exist in at least one region. At least one console must be released in a region for it to be considered in this database.

VIDEO GAME IS ON A CONSOLE

Description: In this binary relationship set, each video game must be released on exactly one console.

Entity sets involved: Video Game, Console Is In A Region (Relationship Set)

Mapping Cardinality: N..1

Descriptive Fields: None.

Participation Constraint: The video game entity set has total participation, because each release is developed for exactly one console in one region. Additionally, the console is in a region relationship set has total participation, because a console and region relationship entity is only added when it is needed by a new video game project.

VIDEO GAME IS IN A PHASE

Description: Each video game release must be in exactly one phase at any given time. In this binary relationship set, both start date and end date attributes will indicate the duration of each phase for each video game entity.

Entity sets involved: Video Game, Phase

Mapping Cardinality: M..N

Descriptive Fields: Start Date, End Date.

Participation Constraint: The video game entity set has total participation, because each video game must be in exactly one phase at any given time. However, the phase entity set has partial participation, because a phase might not be applicable for a while after creation.

PHASE (GENERAL AND SPECIFIC)

Description: This recursive relationship will define a hierarchy of phases and sub-phases. Each phase can be divided into any number of more specific sub-phases.

Entity sets involved: Phase

Mapping Cardinality: 1..N

Descriptive Fields: None.

Participation Constraint: Partial for both general and specific. The top level phases will not be part of any higher level general phases, while the bottom level sub-phases will not be divided into any further specific sub-phases.

VIDEO GAME HAS REACHED A MILESTONE

Description: Each video game release will typically reach one or more milestones during the course of development. In this binary relationship set, a single date stamp will indicate when each milestone is reached.

Entity sets involved: Video Game, Milestone

Mapping Cardinality: M..N

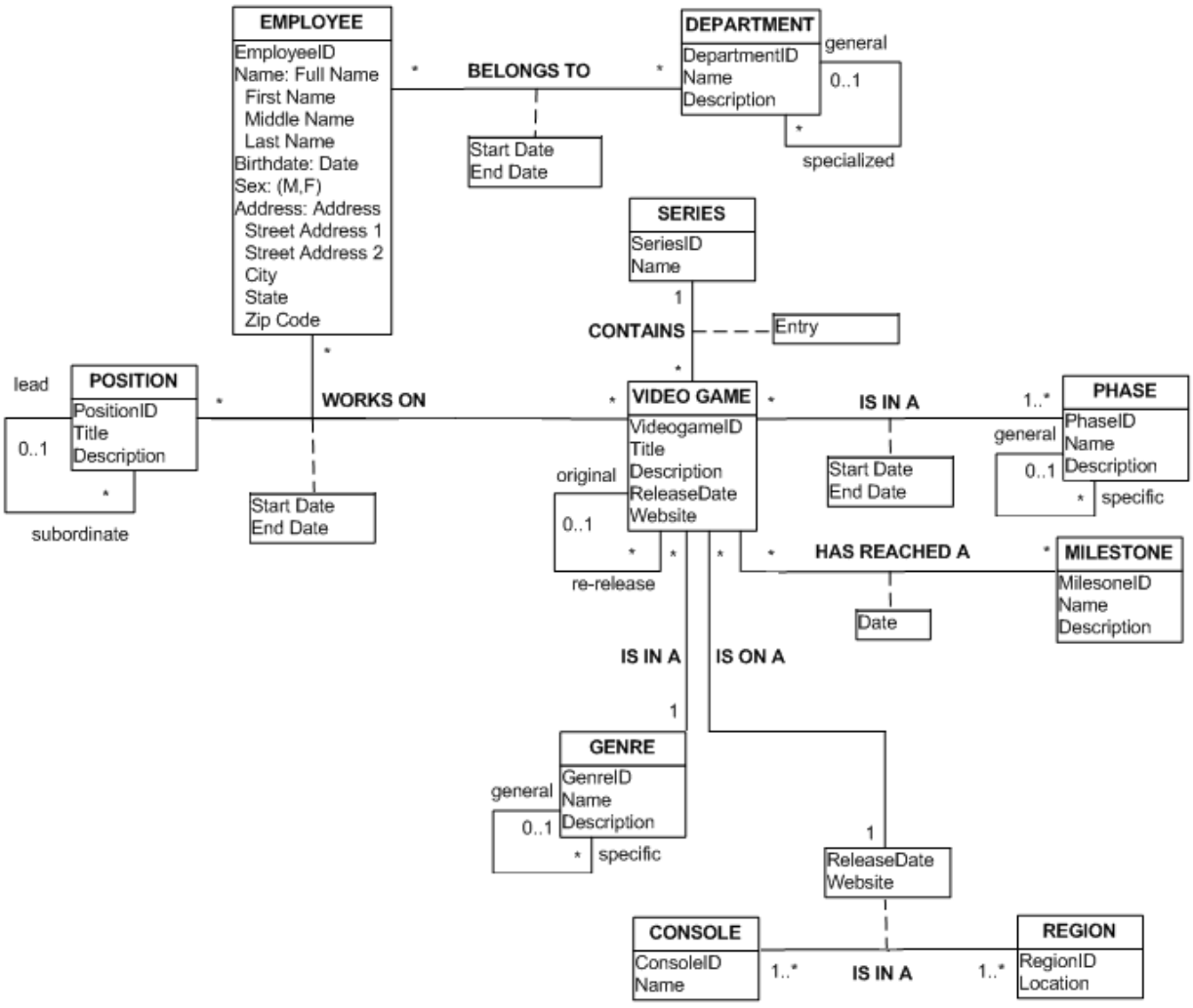
Descriptive Fields: Date.

Participation Constraint: The video game entity set has partial participation, because milestones may not be tracked on every video game release. The milestone entity set also has partial participation, because a milestone might not be applicable for a while after creation.

RELATED ENTITY SET

Rather than use super classes and sub classes, this database is designed with several recursive tables. This will allow for a flexible design that can address the needs of many different development companies.

UML DIAGRAM


EMPLOYEE
EmployeeID
Name: Full Name
First Name
Middle Name
Last Name
Birthdate: Date
Sex: (M,F)
Address: Address
Street Address 1
Street Address 2
City
State
Zip Code
BELONGS TO
Start Date
End Date
DEPARTMENT
DepartmentID
Name
Description
general
0..1
specialized
SERIES
SeriesID
Name
CONTAINS
Entry
lead
POSITION
PositionID
Title
Description
0..1
subordinate
WORKS ON
Start Date
End Date
VIDEO GAME
VideogameID
Title
Description
ReleaseDate
Website
original
0..1
re-release
IS IN A
Start Date
End Date
PHASE
PhaseID
Name
Description
general
0..1
specific
HAS REACHED A
Date
MILESTONE
MilesoneID
Name
Description
IS IN A
IS ON A
GENRE
GenreID
Name
Description
general
0..1
specific
ReleaseDate
Website
CONSOLE
ConsoleID
Name
IS IN A
REGION
RegionID
Location

PHASE II

ER MODEL AND RELATIONAL MODEL

DESCRIPTION OF ER MODEL AND RELATIONAL MODEL

The entity-relationship model is a high level data model that describes a conceptual schema. Generally speaking, an ER model is created during and after gathering requirements and data analysis from potential database users and other knowledgeable personnel. In this model, real world objects are represented by entities, and each entity contains attributes that describe the object. Relationships and constraints that exist between or on the objects are also represented in this model.

The relational model, introduced in 1970 by Ted Codd of IBM Research, uses the concept of a mathematical relation as its foundation. Operations are based in set theory and first-order predicate logic. Generally speaking, the relational model is created by mapping the conceptual schema onto a database schema during the logical design of a database. This model represents the implementation of real world objects and relationships as a collection of relations, each of which contains attributes as defined by its relation schema. Constraints such as domain constraints and integrity constraints can also be represented in this model.

COMPARISON OF ER MODEL AND RELATIONAL MODEL

The ER model is a high level conceptual model that can be understood by a wide range of personnel, including non-technical users. This model is one of the most accurate descriptions of the objects (entities) and relationships as they exist in the real world, as well as one of the most accurate descriptions of the defined data requirements. Implementation details are not included, and this model does not have an associated query language.

The relational model is a logical model that represents one possible implementation of a given set of requirements. While this model may only somewhat accurately describe real world objects and relationships, this model describes a given relational database generally more accurately than a conceptual model. This model is typically only used by database personnel, such as SQL programmers and DBAs. The relational model has an associated query languages.

CONVERSION FROM ER MODEL TO RELATIONAL MODEL

While the ER model may accurately describe the data requirements, at the time of this writing a database cannot be implemented directly from this model. Therefore, this model must be mapped onto an implementation model, such as the relational model. However, careful consideration is needed when converting an ER model into a relational model.

1. For strong entities, a relation is created that includes all simple attributes. When considering component attributes, simple component attributes are also included in the relation. One or more simple attributes that form a key are chosen to be the primary key. Knowledge of other (secondary) keys may be kept for indexing and other purposes.

2. Weak entities are handled similarly, in that a relation is created that includes all simple and simple component attributes of the weak entity. The primary key of the owner entity is included in the relation as both a foreign key and part of the primary key. The partial key, if one exists, is also included as part of the primary key. In general, these foreign keys are set to cascade delete and update operations.
3. Binary 1:1 relationships may be mapped in one of three ways.
    a. Include the primary key of one of the two relations as a foreign key in the other. This method works well if at least one of the two relations has total participation. The foreign key should be implemented in the relation with the highest level of participation, because nulls will exist in the foreign key for any tuple not included in the relationship. Any relationship attributes will be migrated to the relation containing the foreign key.
    b. Both relations may be merged into a single relation if both relations have total participation in the relationship.
    c. The two relations may be cross-referenced by creating a third relation. This third relation will contain, at a minimum, foreign keys that reference the primary keys of both relations participating in the relationship. This combination of foreign keys will form the primary key in the new relation. This may be appropriate if the participation level is low for both relations. This cross-reference relation will also contain any relationship attributes.
4. Binary 1:N or N:1 relationships may be mapped in at least one of two ways.
    a. Include the primary key of the "1-side" relation as a foreign key in the "N-side" relation. Any relationship attributes will be migrated to the "N-side" relation.
    b. If the level of participation of the "N-side" relation is low, the two relations may instead be cross-reference as described above in the binary 1:1 relationship entry.
5. Binary M:N relationships must be cross-referenced as described above. Typically, the foreign keys are set to cascade delete and update operations.
6. In all of the above and following cases, multi-valued attributes must be implemented in a new relation. The relation will contain a foreign key that references the primary key of the relation that represents the entity or relationship that originally contained the attribute. The relation will also contain the attribute in a single tuple for every single value. As always, component attributes will be split into simple attributes.
7. Ternary and N-ary relationships must be cross-referenced as described above. The primary key from each relation participating in the relationship will be implemented as a foreign key in the cross-reference table.
8. Specialization or generalization may be mapped in one of four ways.
    a. Create a relation for each super class and each sub class. The primary key of the super class will be implemented as a foreign key in each corresponding sub class. This will work with any type of specialization or generalization, but often requires joins to produce results of the desired attribute sets.
    b. Create a relation for each sub class. Move all attributes of the corresponding super class into the relation, producing a full attribute set for each specialization. This can only be implemented if participation in the specialization is total for every subclass. If the specialization is overlapping, duplicate entries may be produced.
    c. Create a single relation with a single type attribute. In this case, every super class and sub class will be merged into the single relation, which may result in many nulls in the relation if each sub class has a high number of attributes. To implement this method, the sub classes must be disjoint, because the sub class is designated by the type attribute. For example, a "grad" sub class will be designated by a "grad" entry in the type attribute.

d. Create a single relation with multiple type attributes. This is similar to creating a single relation with a single type attribute with the following difference – instead of a single type attribute, each type is implemented as a separate (Boolean) type attribute. Additionally in this case, sub classes may overlap.

9. Categories or union types may be mapped in one of two ways.
    a. A new relation is created that contains any category attributes as well as a new key attribute, called a surrogate key, implemented as the primary key for the new relation. This surrogate key is then implemented in each super class as a foreign key.
    b. If all super classes in the category have the same primary key, then a surrogate key is not needed. Instead, the common primary key is used. Otherwise, the implementation is as described above.
10. For each recursive relationship, implement a new foreign key in the relation that references the primary key of the same relation.

CONSTRAINTS

Many of the constraints implemented in the ER model will be implemented as constraints in the relational database, as well as constraints not explicitly defined in the ER model. For example, domain constraints are implemented on each attribute in each relation which ensures that every attribute in every tuple is of the correct data type. A null constraint ensures that for a given attribute in every tuple in a given relation, the value will not be null.

The entity integrity constraint ensures that every primary key within a given relation is unique and not null. Additionally, the unique constraint ensures, when used, that the given attribute will be unique for every tuple in the given relation. The referential integrity constraint ensures that for any defined foreign key attribute, either the FK will be null or a corresponding primary key attribute must exist in the given corresponding relation.

All of the above constraints are consider schema-based or explicit constraints, and are typically implemented by the DBMS through settings in the database schema. However, other types of constraints exist, such as business rules. Business rules are constraints generally fairly specific and defined by the business, such as “age must be over 18.”

Business rules may be implemented in a variety of ways, such as check constraints and triggers. Check constraints automatically check to make sure a set of conditions are satisfied before committing any changes to a relation. Triggers contain SQL code that automatically runs instead of or after an insert, update or delete statement.

RELATION SCHEMAS

EMPLOYEE

Employee (**EmployeeID**, NameFirst, NameMiddle, NameLast, Birthdate, Sex, AddressStreet1, AddressStreet2, AddressCity, AddressState, AddressZip)

1. **EmployeeID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. NameFirst – Domain: Varchar(20)
    a. NOT NULL
3. NameMiddle – Domain: Varchar(20)
4. NameLast – Domain: Varchar(40)
    a. NOT NULL
5. Birthdate – Domain: Date (if time is used, default to midnight)
    a. NOT NULL
6. Sex – Domain: Char(1)
    a. Check Constraint: Must be 'M' or 'F'
    b. NOT NULL
7. AddressStreet1 – Domain: Varchar(40)
    a. NOT NULL
8. AddressStreet2 – Domain: Varchar(40)
9. AddressCity – Domain: Varchar(25)
    a. NOT NULL
10. AddressState – Domain: Char(2)
    a. NOT NULL
11. AddressZip – Domain: Varchar(20)
    a. NOT NULL

Other constraints not already mentioned:

1. The set { NameLast, NameFirst, NameMiddle } is unique.

Candidate keys:

1. **EmployeeID** (primary key)
2. NameLast, NameFirst, NameMiddle

DEPARTMENT

Department (**DepartmentID**, Name, Description, ParentID)

1. **DepartmentID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Name – Domain: Varchar(20)
    a. NOT NULL
3. Description – Domain: Varchar(256)
4. ParentID – Domain: Integer
    a. Foreign Key to Department.DepartmentID (recursive)

No other constraints.

Candidate keys:

1. **DepartmentID** (primary key)

POSITION

Position (**PositionID**, Title, Description, ParentID)

1. **PositionID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Title – Domain: Varchar(32)
    a. NOT NULL
3. Description – Domain: Varchar(256)
4. ParentID – Domain: Integer
    a. Foreign Key to Position.PositionID (recursive)

No other constraints.

Candidate keys:

1. **PositionID** (primary key)

VIDEO GAME

Videogame (**VideogameID**, Title, Description, ReleaseDate, Website, ParentID, SeriesID, Entry, GenreID, ConsoleRegionID)

1. **VideogameID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Title – Domain: Varchar(20)
    a. NOT NULL
3. Description – Domain: Varchar(256)
4. ReleaseDate – Domain: Date (if time is used default to midnight)
5. Website – Domain: Varchar(50)
6. ParentID – Domain: Integer
    a. Foreign Key to Videogame.VideogameID (recursive)
7. SeriesID – Domain: Integer
    a. Foreign Key to Series.SeriesID
    b. NOT NULL
8. Entry – Domain: Integer
9. GenreID – Domain: Integer
    a. Foreign Key to Genre.GenreID
    b. NOT NULL
10. ConsoleRegionID – Domain: Integer
    a. Foreign Key to ConsoleIsInRegion.ConsoleRegionID
    b. NOT NULL

No other constraints.

Candidate keys:

1. **VideogameID** (primary key)

PHASE

Phase (**PhaseID**, Name, Description, ParentID)

1. **PhaseID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Name – Domain: Varchar(20)
    a. NOT NULL
    b. Unique
3. Description – Domain: Varchar(256)
4. ParentID – Domain: Integer
    a. Foreign Key to Phase.PhaseID (recursive)

No other constraints.

Candidate keys:

1. **PhaseID** (primary key)
2. Name

MILESTONE

Milestone (**MilestoneID**, Name, Description)

1. **MilestoneID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Name – Domain: Varchar(32)
    a. NOT NULL
    b. Unique
3. Description – Domain: Varchar(256)

No other constraints.

Candidate keys:

1. **MilestoneID** (primary key)
2. Name

GENRE

Genre (**GenreID**, Name, Description, ParentID)

1. **GenreID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Name – Domain: Varchar(32)
    a. NOT NULL
    b. Unique
3. Description – Domain: Varchar(256)
4. ParentID – Domain: Integer
    a. Foreign Key to Genre.GenreID (recursive)

No other constraints.

Candidate keys:

1. **GenreID** (primary key)
2. Name

SERIES

Series (**SeriesID**, Name)

1. **SeriesID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Name – Domain: Varchar(20)
    a. NOT NULL
    b. Unique

No other constraints.

Candidate keys:

1. **SeriesID** (primary key)
2. Name

CONSOLE

Console (**ConsoleID**, Name)

1. **ConsoleID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Name – Domain: Varchar(32)
    a. NOT NULL
    b. Unique

No other constraints.

Candidate keys:

1. **ConsoleID** (primary key)
2. Name

REGION

Region (**RegionID**, Location)

1. **RegionID** – Domain: Integer
    a. Default Value: (Max ID + 1)
    b. Primary Key (implies Unique and NOT NULL)
2. Location – Domain: Varchar(20)
    a. NOT NULL
    b. Unique

No other constraints.

Candidate keys:

1. **RegionID** (primary key)
2. Location

WORKS ON

WorksOn (**VideogameID, EmployeeID**, **PositionID**, StartDate, EndDate)

1. **VideogameID** – Domain: Integer
    a. Foreign Key to Videogame.VideogameID
    b. Part of primary key (implies NOT NULL)
2. **EmployeeID** – Domain: Integer
    a. Foreign Key to Employee.EmployeeID
    b. Part of primary key (implies NOT NULL)
3. **PositionID** – Domain: Integer
    a. Foreign Key to Position.PositionID
    b. Part of primary key (implies NOT NULL)
4. StartDate – Domain: Date (if time is used, default to midnight)
    a. Default Value: Today
    b. NOT NULL
5. EndDate – Domain: Date (if time is used, default to midnight)
    a. Check Constraint: Must be null or greater than or equal to StartDate

Other constraints not already mentioned:

1. The set { **VideogameID, EmployeeID, PositionID** } is unique, because it is the primary key.

Candidate keys:

1. **VideogameID**, **EmployeeID, PositionID** (primary key)

EMPLOYEE BELONGS TO A DEPARTMENT

EmployeeBelongsToDepartment (**EmployeeID**, **DepartmentID**, StartDate, EndDate)

1. **EmployeeID** – Domain: Integer
    a. Foreign Key to Employee.EmployeeID
    b. Part of primary key (implies NOT NULL)
2. **DepartmentID** – Domain: Integer
    a. Foreign Key to Department.DepartmentID
    b. Part of primary key (implies NOT NULL)
3. **StartDate** – Domain: Date (if time is used, default to midnight)
    a. Default Value: Today
    b. Part of primary key (implies NOT NULL)
4. EndDate – Domain: Date (if time is used, default to midnight)
    a. Check Constraint: Must be null or greater than or equal to StartDate

Other constraints not already mentioned:

1. The set { **EmployeeID, StartDate, DepartmentID** } is unique, because it is the primary key.

Candidate keys:

1. **EmployeeID, StartDate, DepartmentID** (primary key)

CONSOLE IS IN A REGION

ConsoleIsInRegion (**ConsoleRegionID**, ConsoleID, RegionID, ReleaseDate, Website)

1. **ConsoleRegionID** – Domain: Integer
    a. Primary key (implies unique and NOT NULL)
2. ConsoleID – Domain: Integer
    a. Foreign Key to Console.ConsoleID
    b. NOT NULL
3. RegionID – Domain: Integer
    a. Foreign Key to Region.RegionID
    b. NOT NULL
4. ReleaseDate – Domain: Date (if time is used default to midnight)
5. Website – Domain: Varchar(50)

Other constraints not already mentioned:

1. The set { ConsoleID, RegionID } is unique.

Candidate keys:

1. **ConsoleRegionID** (primary key)
2. ConsoleID**,** RegionID

VIDEO GAME IS IN A PHASE

VideogameIsInPhase (**VideogameID**, **PhaseID**, StartDate, EndDate)

1. **VideogameID** – Domain: Integer
    a. Foreign Key to Videogame.VideogameID
    b. Part of primary key (implies NOT NULL)
2. **PhaseID** – Domain: Integer
    a. Foreign Key to Phase.PhaseID
    b. Part of primary key (implies NOT NULL)
3. StartDate – Domain: Date (if time is used, default to midnight)
    a. Default Value: Today
    b. NOT NULL
4. EndDate – Domain: Date (if time is used, default to midnight)
    a. Check Constraint: Must be null or greater than or equal to StartDate

Other constraints not already mentioned:

1. The set { **VideogameID, PhaseID** } is unique, because it is the primary key.

Candidate keys:

1. **VideogameID, PhaseID** (primary key)

VIDEO GAME HAS REACHED A MILESTONE

VideogameReachedMilestone (**VideogameID**, **MilestoneID**, Date)

1. **VideogameID** – Domain: Integer
    a. Foreign Key to Videogame.VideogameID
    b. Part of primary key (implies NOT NULL)
2. **MilestoneID** – Domain: Integer
    a. Foreign Key to Milestone.MilestoneID
    b. Part of primary key (implies NOT NULL)
3. DateReached – Domain: Date (if time is used, default to midnight)
    a. Default Value: Today
    b. NOT NULL

Other constraints not already mentioned:

1. The set { **VideogameID, MilestoneID** } is unique, because it is the primary key.

Candidate keys:

1. **VideogameID, MilestoneID** (primary key)

RELATION INSTANCES

EMPLOYEE

Employee (**EmployeeID**, NameFirst, NameMiddle, NameLast, Birthdate, Sex, AddressStreet1, AddressStreet2, AddressCity, AddressState, AddressZip)

EmployeeID	NameFirst	NameMiddle	NameLast	Birthdate	Sex	AddressStreet1	AddressStreet2	AddressCity	AddressState	AddressZip
1	Travis	Ray	Ragle	12/3/79	M	123 Fake St.	Apt. 243	Bakersfield	CA	93307
2	Jennifer	Lynn	Gardner	4/5/66	F	314 Roy St.	NULL	Houston	TX	77077
3	Jerry	Lee	Lewis	3/1/56	M	78 One Way	NULL	Bakersfield	CA	93312
4	Chelsy	Bee	Ray	4/25/83	F	123 Wrong Way	Apt. 27	Los Angeles	CA	90001
5	Homer	Jay	Simpson	11/11/69	M	742 Evergreen Terrace	NULL	Springfield	IL	62701
6	Spencer	Drew	Crawford	1/2/80	M	8694 Rocker St.	NULL	Los Angeles	CA	90001
7	Clancy	Raymond	Wiggim	4/8/72	M	45 Carter Park	NULL	Springfield	IL	62701
8	Terra	Renee	Elizabeth	6/6/84	F	87 Fairfield Rd.	NULL	Bakersfield	CA	93311
9	Karen	Sue	Nancy	9/23/76	F	908 China Grade Lp	NULL	Bakersfield	CA	93309
10	Ryan	Donald	Rickner	2/21/85	M	71 Smith Way	NULL	Berkeley	CA	94701

DEPARTMENT

Department (**DepartmentID**, Name, Description, ParentID)

DepartmentID	Name	Description	ParentID
1	Art	NULL	NULL
2	Sound	NULL	NULL
3	Concept Art	NULL	1
4	Model Design	This group creates 3d models based on concept designs.	1
5	Game Programmers	NULL	NULL
6	IT	NULL	NULL
7	Webmasters	All web work will be handled by this group.	6
8	DBAs	All database work will be handled by this group.	6
9	Game Tools	Development tools, etc.	5
10	Gameplay	Specific work on specific games.	5
11	Quality Assurance	All testing will be handled by this group.	NULL

POSITION

Position (**PositionID**, Title, Description, ParentID)

PositionID	Title	Description	ParentID
1	Lead Programmer	NULL	NULL
2	Physics Programmer	Simulates physics, collision, etc.	1
3	AI Programmer	Rule-based decisions, scripting	1
4	Sound Programmer	Speech, effects, music	1
5	UI Programmer	Menus, etc.	1
6	Lead Artist	NULL	NULL
7	Texture Artist	NULL	6
8	3D Character Artist	NULL	6
9	3D Environment Artist	NULL	6
10	Sprite Artist	NULL	6
11	Lead Quality Assurance	General testing	NULL

VIDEO GAME

Videogame (**VideogameID**, Title, Description, ReleaseDate, Website, ParentID, SeriesID, Entry, GenreID, ConsoleRegionID)

VideogameID	Title	Description	ReleaseDate	Website	ParentID	SeriesID	Entry	GenreID	ConsoleRegionID
151	Lizard Quest	NULL	5/4/85	www.lizardquest.com/lq/nes	NULL	1	1	1	1
152	Lizard Quest II	NULL	8/8/86	www.lizardquest.com/lq2/nes	NULL	1	2	1	1
153	Lizard Quest III	NULL	5/4/88	www.lizardquest.com/lq3/nes	NULL	1	3	1	1
201	Lizard Quest	SNES Remake	1/1/92	www.lizardquest.com/lq/snes	NULL	1	1	1	16
301	Lizard Quest	PS1 Remake	1/1/97	www.lizardquest.com/lq/ps1	151	1	1	2	41
334	Lizard Quest	PSP Remake	1/1/07	www.lizardquest.com/lq/psp	151	1	1	2	49
335	Lizard Quest II	PSP Remake	2/1/07	www.lizardquest.com/lq2/psp	152	1	2	2	49
345	Vito	NULL	2/4/07	NULL	NULL	21	1	5	49
347	Rainfall	NULL	3/23/07	NULL	NULL	25	1	8	49
354	Autumn Knight	NULL	7/7/07	NULL	NULL	26	1	3	49

PHASE

Phase (**PhaseID**, Name, Description, ParentID)

PhaseID	Name	Description	ParentID
1	Phase I	NULL	NULL
2	Design	Phase I – Design only	1
3	Phase II	NULL	NULL
4	Programming	Phase II – Programming only	3
5	Level Creation	Phase II – Level Creation only	3
6	Phase III	NULL	NULL
7	Testing	Phase III – Testing only	6
8	Bug Fix	Phase III – Bug Fixes only	6
9	Clean-up	Phase III – Clean-up only	6
10	Released	NULL	NULL
11	Patching	NULL	10

MILESTONE

Milestone (**MilestoneID**, Name, Description)

MilestoneID	Name	Description
1	Approved For Production	NULL
2	Voice Work Complete	NULL
3	First Model Designed	NULL
4	First Level Designed	NULL
5	Concept Art Complete	NULL
6	First Playable	First version that is playable
7	Alpha	Key gameplay implemented
8	Beta	Only bugs are being worked on, no new code
9	Code Release	Game is ready to ship
10	Gold Master	Final build

GENRE

Genre (**GenreID**, Name, Description, ParentID)

GenreID	Name	Description	ParentID
1	Role-playing	NULL	NULL
2	Traditional Role-playing	Ultima-esque RPGs	1
3	Action Role-playing	RPGs featuring action gameplay	1
4	Action	NULL	NULL
5	Platform	Includes 2d and 3d platformers	4
6	Third Person Shooter	NULL	4
7	Adventure	NULL	NULL
8	Interactive Movie	Adventure game that uses full motion video	7
9	Visual Novel	Interactive fiction game with mostly static graphics	7
10	Dating Sim	NULL	7

SERIES

Series (**SeriesID**, Name)

SeriesID	Name
1	Lizard Quest
5	First Shot
6	Racing!
7	Puzzle Boy
8	The Last Fighter
9	Pure Paranoia
10	NOW
21	Vito
25	Rainfall
26	Autumn Knight

CONSOLE

Console (**ConsoleID**, Name)

ConsoleID	Name
1	Nintendo Entertainment System
2	Famicom
3	Sega Master System
4	Super Nintendo
5	Super Famicom
6	Sega Genesis
7	Nintendo 64
8	Playstation
9	Playstation Portable
10	Playstation 3

REGION

Region (**RegionID**, Location)

RegionID	Location
1	Japan
2	North America
3	Asia
4	Australia
5	England
6	Germany
7	Italy
8	France
9	Spain
10	Middle East

WORKS ON

WorksOn (**VideogameID, EmployeeID**, **PositionID**, StartDate, EndDate)

VideogameID	EmployeeID	PositionID	StartDate	EndDate
151	**2**	**1**	11/1/84	5/4/85
151	**2**	**6**	11/1/84	5/4/85
151	**3**	**1**	11/1/84	5/4/85
151	**3**	**6**	11/1/84	5/4/85
152	**2**	**6**	11/1/85	8/8/86
152	**2**	**1**	11/1/85	8/8/86
152	**3**	**1**	11/1/85	8/8/86
152	**3**	**6**	11/1/85	8/8/86
153	**2**	**1**	2/1/87	5/4/88
153	**2**	**6**	2/1/87	5/4/88
153	**3**	**1**	2/1/87	5/4/88
153	**3**	**6**	2/1/87	5/4/88
201	**2**	**6**	10/26/90	11/12/91
201	**2**	**10**	2/4/91	10/10/91
201	**3**	**1**	5/5/90	12/18/91
201	**3**	**2**	8/8/90	10/1/91
201	**3**	**3**	2/4/91	6/5/91
201	**5**	**2**	8/8/90	10/1/91
201	**5**	**5**	5/5/91	10/1/91
201	**7**	**4**	4/13/91	11/11/91
201	**7**	**7**	2/4/91	9/5/91
201	**7**	**10**	2/4/91	8/8/91
301	**2**	**6**	5/7/95	12/12/96
301	**2**	**7**	6/5/95	8/7/96
301	**3**	**1**	5/7/95	12/20/96
301	**3**	**2**	5/7/95	11/11/96
301	**3**	**3**	5/7/95	11/18/96
301	**5**	**2**	5/7/95	12/20/96
301	**5**	**4**	3/4/96	11/22/96
301	**5**	**5**	4/2/96	11/22/96
301	**7**	**8**	9/9/95	11/10/96
301	**7**	**9**	9/9/95	9/21/96
301	**9**	**8**	11/12/95	11/10/96
301	**9**	**9**	11/12/95	9/21/96
334	**2**	**6**	4/1/05	12/12/06
334	**2**	**7**	4/1/05	9/3/06
334	**4**	**7**	5/5/05	9/2/06
334	**4**	**8**	5/5/05	10/2/06
334	**4**	**9**	5/5/05	10/17/06
334	**5**	**1**	4/1/05	12/22/06
334	**5**	**2**	4/1/05	11/16/06
334	**5**	**5**	6/12/06	12/1/06
334	**6**	**3**	6/5/05	12/14/06
334	**6**	**4**	7/7/06	12/2/06

334	**7**	**8**	4/1/05	10/23/06
334	**7**	**9**	4/1/05	9/12/06
334	**8**	**8**	6/7/05	10/15/06
334	**9**	**8**	4/1/05	10/12/06
334	**9**	**9**	4/1/05	10/13/06
334	**10**	**2**	7/6/05	12/1/06
334	**10**	**3**	7/6/05	12/1/06
335	**2**	**6**	4/1/05	12/12/06
335	**2**	**7**	4/1/05	9/3/06
335	**4**	**7**	5/5/05	9/2/06
335	**4**	**8**	5/5/05	10/2/06
335	**4**	**9**	5/5/05	10/17/06
335	**5**	**1**	4/1/05	12/22/06
335	**5**	**2**	4/1/05	11/16/06
335	**5**	**5**	6/12/06	12/1/06
335	**6**	**3**	6/5/05	12/14/06
335	**6**	**4**	7/7/06	12/2/06
335	**7**	**8**	4/1/05	10/23/06
335	**7**	**9**	4/1/05	9/12/06
335	**8**	**8**	6/7/05	10/15/06
335	**9**	**8**	4/1/05	10/12/06
335	**9**	**9**	4/1/05	10/13/06
335	**10**	**2**	7/6/05	12/1/06
335	**10**	**3**	7/6/05	12/1/06

EMPLOYEE BELONGS TO A DEPARTMENT

EmployeeBelongsToDepartment (**EmployeeID**, **DepartmentID**, StartDate, EndDate)

EmployeeID	DepartmentID	StartDate	EndDate
1	**1**	4/5/2004	5/3/2006
1	**9**	9/10/2009	NULL
1	**10**	5/4/2006	9/9/2009
2	**1**	11/1/1984	2/2/2003
3	**5**	11/1/1984	5/3/2006
3	**3**	5/4/2006	NULL
4	**1**	6/7/2004	5/3/2006
4	**4**	5/4/2006	10/10/10
4	**11**	10/11/10	NULL
5	**5**	9/9/1989	5/3/2006
5	**10**	5/4/2006	NULL
6	**5**	2/8/2005	5/3/2006
6	**10**	5/4/2006	NULL
7	**1**	10/5/1989	5/3/2006
7	**4**	5/4/2006	NULL
8	**1**	7/7/2004	5/3/2006
8	**4**	5/4/2006	9/8/2009
8	**3**	9/9/2009	NULL
9	**1**	10/21/1989	5/3/2006
9	**4**	5/4/2006	NULL
10	**5**	1/3/2005	5/3/2006
10	**10**	5/4/2006	6/8/2008
10	**8**	6/9/2008	2/1/2010
10	**10**	2/2/2010	2/23/2010
10	**9**	2/24/2010	NULL
11	**2**	6/3/2007	NULL
12	**7**	5/4/2008	7/7/2010
12	**8**	7/8/2010	NULL
13	**11**	2/2/2008	NULL
14	**4**	2/3/2008	10/11/2010
14	**3**	10/12/2010	NULL
15	**7**	4/6/2008	NULL
16	**4**	4/23/2008	NULL
17	**10**	6/3/2008	NULL
18	**10**	6/3/2008	8/8/2008
19	**10**	6/3/2008	NULL
20	**10**	6/3/2008	NULL
21	**4**	6/3/2008	6/6/2008
22	**4**	6/3/2008	NULL
23	**11**	7/5/2008	NULL
24	**11**	7/24/2008	9/3/2009
25	**2**	9/4/2009	NULL
26	**7**	8/5/2008	NULL
27	**8**	10/1/2008	10/1/2010

27	**10**	10/2/2010	NULL
28	**10**	2/1/2009	NULL
29	**10**	2/4/2009	7/8/2009
29	**9**	7/9/2009	NULL
30	**4**	3/5/2009	NULL
31	**4**	4/5/2009	NULL
32	**11**	5/5/2009	NULL
33	**8**	4/4/2009	4/5/2009
33	**10**	4/6/2009	5/6/2009
33	**8**	5/7/2009	NULL
34	**10**	3/3/2010	NULL
35	**10**	3/3/2010	NULL
36	**10**	3/3/2010	NULL
37	**10**	3/3/2010	3/20/2010
38	**4**	3/3/2010	6/3/2010
39	**10**	3/3/2010	NULL
40	**4**	3/3/2010	NULL
41	**4**	3/3/2010	3/20/2010
42	**10**	3/3/2010	NULL
43	**4**	3/3/2010	NULL
44	**7**	6/7/2010	NULL
45	**8**	8/5/2010	NULL
46	**11**	10/10/2010	NULL

CONSOLE IS IN A REGION

ConsoleIsInRegion (**ConsoleRegionID**, ConsoleID, RegionID, ReleaseDate, Website)

ConsoleRegionID	ConsoleID	RegionID	ReleaseDate	Website
1	1	2	10/18/1985	www.nintendo.com
2	1	5	9/1/1986	www.nintendo.com
3	1	6	9/1/1986	www.nintendo.com
4	1	7	9/1/1986	www.nintendo.com
5	1	8	9/1/1986	www.nintendo.com
6	1	9	9/1/1986	www.nintendo.com
7	1	4	9/1/1987	www.nintendo.com
8	2	1	7/15/1983	www.nintendo.com
9	3	1	7/5/1989	www.sega.com
10	3	2	4/3/1992	www.sega.com
11	3	5	9/1/1996	www.sega.com
12	3	6	9/1/1996	www.sega.com
13	3	7	9/1/1996	www.sega.com
14	3	8	9/1/1996	www.sega.com
15	3	9	9/1/1996	www.sega.com
16	4	2	8/23/1991	www.nintendo.com
17	4	5	4/11/1992	www.nintendo.com
18	4	6	4/11/1992	www.nintendo.com
19	4	7	4/11/1992	www.nintendo.com
20	4	8	4/11/1992	www.nintendo.com
21	4	9	4/11/1992	www.nintendo.com
22	4	4	4/3/1992	www.nintendo.com
23	5	1	11/21/1990	www.nintendo.com
24	6	1	10/29/1988	www.sega.com
25	6	2	8/14/1989	www.sega.com
26	6	4	11/30/1990	www.sega.com
27	6	5	11/30/1990	www.sega.com
28	6	6	11/30/1990	www.sega.com
29	6	7	11/30/1990	www.sega.com
30	6	8	11/30/1990	www.sega.com
31	6	9	11/30/1990	www.sega.com
32	7	1	6/23/1996	www.nintendo.com
33	7	2	9/29/1996	www.nintendo.com
34	7	4	3/1/1997	www.nintendo.com
35	7	5	3/1/1997	www.nintendo.com
36	7	6	3/1/1997	www.nintendo.com
37	7	7	3/1/1997	www.nintendo.com
38	7	8	3/1/1997	www.nintendo.com
39	7	9	3/1/1997	www.nintendo.com
40	8	1	12/3/1994	jp.playstation.com
41	8	2	9/9/1995	us.playstation.com
42	8	5	9/29/1995	uk.playstation.com
43	8	6	9/29/1995	de.playstation.com
44	8	7	9/29/1995	it.playstation.com

45	8	8	9/29/1995	fr.playstation.com
46	8	9	9/29/1995	es.playstation.com
47	8	4	11/15/1995	au.playstation.com/psp
48	9	1	12/12/2004	jp.playstation.com/psp
49	9	2	3/24/2005	us.playstation.com/psp
50	9	4	9/1/2005	au.playstation.com/psp
51	9	5	9/1/2005	uk.playstation.com/psp
52	9	6	9/1/2005	de.playstation.com/psp
53	9	7	9/1/2005	it.playstation.com/psp
54	9	8	9/1/2005	fr.playstation.com/psp
55	9	9	9/1/2005	es.playstation.com/psp
56	10	1	11/11/2006	jp.playstation.com/ps3
57	10	2	11/17/2006	us.playstation.com/ps3
58	10	3	11/17/2006	asia.playstation.com
59	10	4	3/23/2007	au.playstation.com/ps3
60	10	5	3/16/2007	uk.playstation.com/ps3

VIDEO GAME IS IN A PHASE

VideogameIsInPhase (**VideogameID**, **PhaseID**, StartDate, EndDate)

VideogameID	PhaseID	StartDate	EndDate
334	**2**	4/1/2004	5/7/2004
334	**1**	5/8/2004	10/2/2004
334	**5**	10/3/2004	2/3/2005
334	**3**	2/4/2005	5/4/2005
334	**4**	5/5/2005	7/6/2006
334	**6**	7/7/2006	9/7/2006
334	**7**	9/8/2006	10/23/2006
334	**8**	10/24/2006	12/1/2006
334	**9**	12/1/2006	12/31/2006
334	**10**	1/1/2007	1/1/2007
335	**2**	4/1/2004	5/7/2004
335	**1**	5/8/2004	10/2/2004
335	**5**	10/3/2004	2/3/2005
335	**3**	2/4/2005	5/4/2005
335	**4**	5/5/2005	7/6/2006
335	**6**	7/7/2006	9/7/2006
335	**7**	9/8/2006	10/23/2006
335	**8**	10/24/2006	12/1/2006
335	**9**	12/1/2006	12/31/2006
335	**10**	2/1/2007	2/1/2007
345	**2**	2/1/2004	4/3/2004
345	**1**	4/4/2004	9/7/2004
345	**5**	9/8/2004	3/4/2005
345	**3**	3/5/2005	5/8/2005
345	**4**	5/9/2005	7/1/2006
345	**6**	7/2/2006	9/1/2006
345	**7**	9/2/2006	10/31/2006
345	**8**	11/1/2006	12/19/2006
345	**9**	12/20/2006	1/15/2007
345	**10**	2/4/2007	2/4/2007
347	**2**	10/1/2003	12/15/2003
347	**1**	12/16/2003	3/22/2004
347	**5**	3/23/2004	1/13/2005
347	**3**	1/14/2005	3/22/2005
347	**4**	3/23/2005	6/1/2006
347	**6**	6/2/2006	8/27/2006
347	**7**	8/28/2006	10/1/2006
347	**8**	10/2/2006	12/29/2006
347	**9**	12/30/2006	1/31/2007
347	**10**	3/23/2007	3/23/2007
354	**2**	2/1/2004	5/5/2004
354	**1**	5/6/2004	7/22/2004
354	**5**	7/23/2004	1/31/2005
354	**3**	2/1/2005	3/22/2005

354	**4**	3/23/2005	9/2/2005
354	**6**	9/3/2005	1/31/2006
354	**7**	2/1/2006	4/1/2006
354	**8**	4/2/2006	5/29/2006
354	**9**	5/30/2006	6/30/2007
354	**10**	7/7/2007	7/7/2007
356	**2**	6/7/2008	12/11/2008
356	**1**	12/12/2008	1/1/2009
356	**5**	1/2/2009	4/2/2009
356	**3**	4/3/2009	6/22/2009
356	**4**	6/23/2009	1/11/2010
356	**6**	1/12/2010	2/22/2010
356	**7**	2/23/2010	6/1/2010
356	**8**	6/2/2010	8/22/2010
356	**9**	8/23/2010	9/10/2010
356	**10**	10/10/2010	10/10/2010

VIDEO GAME HAS REACHED A MILESTONE

VideogameReachedMilestone (**VideogameID**, **MilestoneID**, Date)

VideogameID	MilestoneID	Date
334	**1**	2/1/2004
334	**2**	11/12/2004
334	**3**	10/13/2004
334	**4**	2/3/2005
334	**5**	11/11/2004
334	**6**	6/8/2006
334	**7**	9/7/2006
334	**8**	10/24/2006
334	**9**	12/14/2006
334	**10**	12/15/2006
335	**1**	2/1/2004
335	**2**	11/12/2004
335	**3**	10/13/2004
335	**4**	2/3/2005
335	**5**	11/11/2004
335	**6**	6/8/2006
335	**7**	9/7/2006
335	**8**	10/24/2006
335	**9**	12/14/2006
335	**10**	12/15/2006
345	**1**	12/12/2003
345	**2**	11/12/2004
345	**3**	7/5/2004
345	**4**	3/4/2005
345	**5**	2/1/2005
345	**6**	6/23/2006
345	**7**	9/10/2006
345	**8**	11/1/2006
345	**9**	1/15/2007
345	**10**	1/23/2007
347	**1**	7/4/2003
347	**2**	12/15/2004
347	**3**	2/1/2004
347	**4**	1/13/2005
347	**5**	12/16/2004
347	**6**	4/1/2006
347	**7**	8/18/2006
347	**8**	10/2/2006
347	**9**	2/1/2007
347	**10**	3/3/2007
354	**1**	12/12/2003
354	**2**	11/11/2004
354	**3**	3/1/2004
354	**4**	1/30/2005

354	**5**	11/12/2004
354	**6**	8/6/2005
354	**7**	1/15/2006
354	**8**	4/2/2006
354	**9**	6/20/2007
354	**10**	6/22/2007
356	**1**	5/5/2008
356	**2**	9/9/2008
356	**3**	6/2/2009
356	**4**	4/2/2009
356	**5**	4/1/2009
356	**6**	1/2/2010
356	**7**	2/13/2010
356	**8**	6/2/2006
356	**9**	9/10/2010
356	**10**	9/22/2010

SAMPLE QUERIES

Select all female employees born after January 1, 1980

Select all games that Travis Ragle has worked on

Select all employees that have worked on every game that Travis Ragle has worked on.

Select the second oldest employee

Select all employees that have not worked on a Lizard Quest game

Select all games worked on in 2005

Select all employees employed since 2005 (or before) or that were hired on or after 1/1/2010

Select all employees and their current projects (games)

Select oldest game

Select all employees that have worked in a subordinate position

SAMPLE QUERY EXPRESSIONS

Select all female employees born after January 1, 1980

Relational Algebra

$\sigma_{Sex = "F" \Lambda \text{ Birthdate} > "1/1/1980"} Employee$

Tuple Relational Calculus

$\{e \mid Employee(e) \Lambda \text{ e. Sex} = \text{F} \Lambda \text{ e. Birthdate} > "1/1/1980"\}$

Domain Relational Calculus

$\{f, m, l \mid Employee(\_, f, m, l, > "1/1/1980", "F", \_, \_, \_, \_, \_)\}$

Select all games that Travis Ragle has worked on

Relational Algebra

$\pi_{Title}(Videogame * WorksOn * (\sigma_{NameFirst="Travis" \wedge NameLast="Ragle"} Employee))$

Tuple Relational Calculus

$\{v \mid Videogame(v) \wedge (\exists w)(\exists e)(\text{WorksOn}(w) \wedge \text{Employee}(e) \wedge v.\text{VideogameID} = w.\text{VideogameID} \wedge e.\text{EmployeeID} = w.\text{EmployeeID} \wedge e.\text{NameFirst} = \text{"Travis"} \wedge e.\text{NameLast} = \text{"Ragle"})\}$

Domain Relational Calculus

$\{t \mid (\exists v)(Videogame(v, t, \_, \_, \_, \_, \_, \_, \_, \_) \wedge (\exists e)(\text{Employee}(e, \text{"}Travis\text{"}, \_, \text{"}Ragle\text{"}, \_, \_, \_, \_, \_, \_) \wedge \text{WorksOn}(v, e, \_, \_, \_)))\}$

Select all employees that have worked on every game that Travis Ragle has worked on.

Relational Algebra

$\pi_{NameFirst,NameMiddle,NameLast}((\sigma_{NameFirst\neq"Travis" \wedge NameLast\neq"Ragle"} Employee * \pi_{VideogameID,EmployeeID} WorksOn) \div \pi_{VideogameID}(WorksOn * (\sigma_{NameFirst="Travis" \wedge NameLast="Ragle"} Employee)))$

Tuple Relational Calculus

$\{e \mid Employee(e) \wedge (\exists w)(\forall w2)(\exists e2)(\text{WorksOn}(w2) \rightarrow \text{WorksOn}(w) \wedge \text{Employee}(e2) \wedge e2.\text{NameFirst} = \text{"Travis"} \wedge e2.\text{NameLast} = \text{"Ragle"} \wedge e2.\text{EmployeeID} = w2.\text{EmployeeID} \wedge e.\text{EmployeeID} = w.\text{EmployeeID} \wedge w.\text{VideogameID} = w2.\text{VideogameID} \wedge e.\text{EmployeeID} \neq e2.\text{EmployeeID})\}$

Domain Relational Calculus

$\{f, m, l \mid (\exists e)(\exists e2)(\text{Employee}(e, f, m, l, \_, \_, \_, \_, \_, \_) \wedge \text{Employee}(e2, \text{"}Travis\text{"}, \_, \text{"}Ragle\text{"}, \_, \_, \_, \_, \_, \_) \wedge e \neq e2 \wedge (\forall v)(\text{WorksOn}(v, e2, \_, \_, \_) \rightarrow \text{WorksOn}(v, e, \_, \_, \_)))\}$

Select the second oldest employee

Relational Algebra

$$Employee * (\pi_{e1.EmployeeID}(\sigma_{e1.Birthday > e2.Birthdate}\left(\begin{matrix}e1\\Employee\end{matrix} \times \begin{matrix}e2\\Employee\end{matrix}\right)) - \pi_{e3.EmployeeID}(\sigma_{e3.Birthday > e4.Birthdate\ \Lambda\ \text{e4.Birthdate} > \text{e5.Birthdate}}\left(\begin{matrix}e3\\Employee\end{matrix} \times \begin{matrix}e4\\Employee\end{matrix} \times \begin{matrix}e5\\Employee\end{matrix}\right)))$$

Tuple Relational Calculus

$$\{e \mid Employee(e)\ \Lambda\ (\exists o)(\text{Employee(o)}\ \Lambda\ \text{e. Birthdate} \neq \text{o. Birthdate}\ \Lambda\ (\forall e2)(\text{Employee(e2)} \rightarrow \text{o. Birthdate} \leq \text{e2. Birthdate}\ \Lambda\ (\text{e. Birthdate} \leq \text{e2. Birthdate V e2. Birthdate} = \text{o. Birthdate})))\}$$

Domain Relational Calculus

$$\{f, m, l \mid (\exists a)(\exists o)(\text{Employee}(\_, f, m, l, a, \_, \_, \_, \_, \_, \_)\ \Lambda\ \text{Employee}(\_, \_, \_, \_, o, \_, \_, \_, \_, \_, \_)\ \Lambda\ \text{o} \neq \text{a}\ \Lambda\ (\forall a2)(\text{Employee}(\_, \_, \_, \_, a2, \_, \_, \_, \_, \_, \_) \rightarrow \text{o} \leq \text{a2}\ \Lambda\ (\text{a} \leq \text{a2 V o} = \text{a2})))\}$$

Select all employees that have not worked on a Lizard Quest game

Relational Algebra

$$Employee * (\pi_{EmployeeID} Empoyee - \pi_{EmployeeID}(WorksOn * (Videogame * \sigma_{Name="Lizard\ Quest"} Series)))$$

Tuple Relational Calculus

$$\{e \mid Employee(e)\ \Lambda\ (\exists s)(\exists v)(\text{Series(s)}\ \Lambda\ \text{Videogame(v)}\ \Lambda\ \text{s. Name} = \text{"Lizard Quest"}\ \Lambda\ \text{v. SeriesID} = \text{s. SeriesID}\ \Lambda\ \neg(\exists w)(\text{WorksOn(w)}\ \Lambda\ \text{w. VideogameID} = \text{v. VideogameID}\ \Lambda\ \text{w. EmployeeID} = \text{e. EmployeeID}))\}$$

Domain Relational Calculus

$$\{f, m, l \mid (\exists e)(\exists s)(\text{Employee}(e, f, m, l, \_, \_, \_, \_, \_, \_, \_)\ \Lambda\ \text{Series(s, "Lizard Quest")}\ \Lambda\ \neg(\exists v)(\text{Videogame(v}, \_, \_, \_, \_, \_, \text{s}, \_, \_, \_)\ \Lambda\ \text{WorksOn(v, e}, \_, \_, \_))))\}$$

Select all games worked on in 2005

Relational Algebra

$Videogame * \pi_{VideogameID}(\sigma_{StartDate \leq \text{"12/31/2005"} \Lambda \text{ EndDate} \geq \text{"1/1/2005"}} WorksOn)$

Tuple Relational Calculus

$\{v \mid Videogame(v) \Lambda (\exists w)(\text{WorksOn(w)} \Lambda \text{ w. VideogameID} = \text{v. VideogameID} \Lambda \text{ w. StartDate} \leq \text{"12/31/2005"} \Lambda \text{ w. EndDate} \geq \text{"1/1/2005"})\}$

Domain Relational Calculus

$\{t \mid (\exists v)(\text{Videogame(v, t, \_, \_, \_, \_, \_, \_, \_, \_)} \Lambda \text{ WorksOn(v, \_, \_, } \leq \text{ "12/31/2005", } \geq \text{ "1/1/2005"}))\}$

Select all employees employed since 2005 (or before) or that were hired on or after 1/1/2010

Relational Algebra

$Employee * (\pi_{EmployeeID}(\sigma_{StartDate \leq \text{"12/31/2005"}} EmployeeBelongsToDepartment) \cup \pi_{EmployeeID}(\sigma_{StartDate \geq \text{"1/1/2010"}} EmployeeBelongsToDepartment))$

Tuple Relational Calculus

$\{e \mid Employee(e) \Lambda (\exists d)(\text{EmployeeBelongsToDepartment(d)} \Lambda \text{ d. EmployeeID} = \text{e. EmployeedID} \Lambda (\text{d. StartDate} \leq \text{"12/31/2005" V d.StartDate} \geq \text{"1/1/2010"}))\}$

Domain Relational Calculus

$\{f, m, l \mid (\exists e)(\exists d)(\text{Employee}(e, f, m, l, \_, \_, \_, \_, \_, \_) \Lambda EmployeeBelongsToDepartment(\text{e, \_, d, \_}) \Lambda (\text{d} \leq \text{ "12/31/2005" V d } \geq \text{ "1/1/2010"}))\}$

Select all employees and their current projects (games)

Relational Algebra

$$\pi_{e.NameFirst,e.NameLast,v.Title}\left(\overset{\text{e}}{\text{Employee}} \rtimes_{\text{e.EmployeeID=w.EmployeeID}} \left(\sigma_{\text{EndDate=NULL}}\overset{\text{w}}{\text{WorksOn}} * \overset{\text{v}}{\text{Videogame}}\right)\right)$$

Tuple Relational Calculus

$\{e.NameFirst, e.NameLast, v.Title \mid Employee(e)\ \Lambda\ (\exists v)(\exists w)(\text{WorksOn(w)}\ \Lambda\ \text{Videogame(v)}\ \Lambda\ \text{w. EmployeeID} = \text{e. EmployeeID}\ \Lambda\ \text{w. VideogameID} = \text{v. VideogameID}\ \Lambda\ \text{w. EndDate} = \text{NULL})\}$

Domain Relational Calculus

$\{f, l, t \mid (\exists e)(\exists v)(\text{Employee}(e, f, \_, l, \_, \_, \_, \_, \_, \_, \_)\ \Lambda\ \text{Videogame(v, t,}\ \_, \_, \_, \_, \_, \_, \_, \_)\ \Lambda\ \text{WorksOn(v, e,}\ \_, \_, \text{NULL}))\}$

Select oldest game

Relational Algebra

$$Videogame * (\pi_{VideogameID} Videogame - \pi_{v1.VideogameID}(\sigma_{v1.ReleaseDate > v2.ReleaseDate}\left(\overset{v1}{Videogame} \times \overset{v2}{Videogame}\right)))$$

Tuple Relational Calculus

$\{v \mid Videogame(v)\ \Lambda\ (\forall x)(\text{Videogame(x)} \rightarrow \text{v. ReleaseDate} \geq \text{x. ReleaseDate})\}$

Domain Relational Calculus

$\{t \mid (\exists v)(\exists d)(\text{Videogame(v, t,}\ \_, \text{d,}\ \_, \_, \_, \_, \_, \_)\ \Lambda\ \neg\text{Videogame}(\neq \text{v,}\ \_, \_, < d, \_, \_, \_, \_, \_, \_))\}$

Select all employees that have worked in a subordinate position

Relational Algebra

$Employee * \pi_{EmployeeID}(\sigma_{ParentID \neq NULL} Position * WorksOn * Employee)$

Tuple Relational Calculus

$\{e \mid Employee(e) \Lambda (\exists w)(\exists p)(\text{Position(p) } \Lambda \text{ WorksOn(w) } \Lambda \text{ p. ParentID } \neq \text{ NULL } \Lambda \text{ w. PositionID} = \text{p. PositionID } \Lambda \text{ w. EmployeeID} = \text{e. EmployeeID})\}$

Domain Relational Calculus

$\{f, m, l \mid (\exists e)(\exists p)(\text{Employee}(e, f, m, l, \_, \_, \_, \_, \_, \_) \Lambda WorksOn(\_, e, p, \_, \_) \Lambda \text{ Position(p, \_, \_} \neq \text{ NULL}))\}$

PHASE III

SQL*PLUS

SQL*PLUS, formerly known as *UFI* (User Friendly Interface) and *Advanced UFI*, is a command line tool for use with the Oracle DBMS. This tool allows standard SQL and PL/SQL code to be written and executed, as well as internal commands (such as SET an environment variable) and script files. For example, this tool can commonly be used to create and manage schema objects (such as tables and indexes), manipulate existing data and query existing data. Additionally, a hierarchal set of script files containing the complete definition of a database schema can be created and later executed in SQL*PLUS. This allows such tasks as creating an entire database to be accomplished very quickly.

SCHEMA OBJECTS IN ORACLE

Several schema objects exist in Oracle. Each schema object is logically stored in a given tablespace. However, since the data of a tablespace will typically be stored in several datafiles, each schema object may be physically stored in one or more datafiles.

TABLES

Tables are the primary means for storing data in a RDBMS. Tables are approximately equivalent to relations in the relational model. Each attribute is represented as a column, and each tuple is represented as a row of data. Additionally, each table has associated metadata that can typically be used to define constraints such as entity integrity, referential integrity, domain constraints and null constraints.

However, a few differences exist between tables and relations. For example, a relation theoretically cannot contain duplicate tuples while a table may contain duplicate rows. Additionally, a relation is an unordered set of tuples, while in practice a representation of a given table is ordered.

The following partial example syntax can be used to create a table in Oracle:

```
CREATE TABLE tableName
(
        columnName1   datatype1     defaultExpression1     constraintsOnColumn1,
        columnName2   datatype2     defaultExpression3     constraintsOnColumn2,
        …             …             …                      …
        columnNameN   datatypeN     defaultExpressionN     constraintsOnColumnN,
        tableConstraints
);
```

Tables in this database:
tr_Console
tr_ConsoleIsInRegion
tr_Department
tr_Employee
tr_EmployeeBelongsToDepartment
tr_Genre
tr_Milestone
tr_Phase
tr_Position
tr_Region
tr_Series
tr_Videogame
tr_VideogameIsInPhase
tr_VideogameReachedMilestone
tr_WorksOn

VIEWS

A view is a stored query that is treated as a virtual table. Views have a variety of uses, such as limiting the result set to a subset of data contained in one or more underlying tables, simplifying multiple joins to a single virtual table and implementing virtual tables based on aggregate functions. Generally speaking the data in a view is not stored directly; the data is stored in the underlying tables or computed at query time. However, Oracle allows materialized views which are stored snapshots.

The following partial example syntax can be used to create a view in Oracle:

CREATE OR REPLACE VIEW *viewName* AS *selectStatement*;

INDEXES

An index is a stored subset of data from a given table that is used to optimize the speed in which certain queries can be executed against the table, or implement certain constraints such as uniqueness or entity integrity. The index contains the projection of one or more columns from a given table into an ordered result set, which can then be used as a lookup table for the original table. Indexes can be unique or contain duplicate values. Indexes can also be either clustered or non-clustered. Clustered indexes physically order the contents of a given table, while non-clustered indexes logically order the contents even though the physical contents may not be ordered.

The following partial example syntax can be used to create an index in Oracle:

CREATE INDEX *indexName* ON *tableName* TABLESPACE *tablespace*;

Indexes in this database:
ix_tr_Videogame_Title
pk_tr_Console
pk_tr_ConsoleIsInRegion
pk_tr_Department
pk_tr_EmpBelongsToDept
pk_tr_Employee
pk_tr_Genre
pk_tr_Milestone
pk_tr_Phase
pk_tr_Position
pk_tr_Region
pk_tr_Series
pk_tr_VgameReachedMilestone
pk_tr_Videogame
pk_tr_VideogameIsInPhase
pk_tr_WorksOn
uk_tr_ConsoleIsInRegion_fk
uk_tr_Console_Name
uk_tr_Employee_Name
uk_tr_Genre_Name
uk_tr_Milestone_Name
uk_tr_Phase_Name
uk_tr_Region_Location
uk_tr_Series_Name

SEQUENCES

A sequence is used to automatically generate a sequential series of numbers in Oracle. The sequence is not explicitly tied to a specific table, so a single sequence may be used across several tables. The most common use of a sequence is to aid in implementing an ID column for one or more tables, which can then in turn be used as a primary key.

The following partial example syntax can be used to create a sequence in Oracle:

CREATE SEQUENCE *seqName*
    MINVALUE *minimumValue*
    MAXVALUE *maximumValue*
    START WITH *initialValue*
    INCREMENT BY *incrementValue*
    CACHE *numberOfValuesToCache;*

SYNONYMS

A synonym is an alternative name for a schema object. One typical use of synonyms is to create transparency for objects that are external to a given schema that would otherwise have to be qualified. A synonym may be public, which would allow any database user to access the synonym, but not necessarily the underlying data unless the user has proper privileges.

The following partial example syntax can be used to create a synonym in Oracle:

CREATE OR REPLACE PUBLIC SYNONYM *synonymName* FOR *schemaObject;*

CLUSTERS

A cluster stores data from one or more tables. Each table that participates in a cluster must contain columns that match the definition of each column defined in the cluster. For example, for a given cluster that contains two VARCHAR(20) columns named "lName" and "fName", each table that participates in the cluster must also contain two VARCHAR(20) columns named "lName" and "fName." The cluster can then be used by the DBMS to optimize queries which join two or more tables that participate in the cluster.

The following partial example syntax can be used to create a cluster in Oracle:

CREATE CLUSTER *clusterName (column1, column2, …, columnN)*
SIZE *dataBlockSize*
STORAGE (INITIAL *initialSizeOfCluster* NEXT *sizeToAllocate*);

DATABASE LINKS

A database link is used to connect a given database to another database or schema.

The following partial example syntax can be used to create a database link in Oracle:

CREATE DATABASE LINK *linkName*
CONNECT TO *schema* BY *password*
USING *connectString*;

SNAPSHOTS

A snapshot is a special type of table which contains the result set of a query ran at a specific time. In later versions of Oracle snapshots are called materialized views.

The following partial example syntax can be used to create a materialized view in Oracle:

CREATE MATERIALIZED VIEW *mViewName* AS *selectStatement;*

PROCEDURES

In Oracle, a stored procedure is a group of PL/SQL statements that is compiled and stored in a given database. The stored procedure can then be executed by name. One common use of stored procedures is to aid in implementing business logic during certain tasks, such as while inserting a new record into a table.

The following partial example syntax can be used to create a stored procedure in Oracle:

```
CREATE OR REPLACE PROCEDURE procedureName (paramaterList) AS
        variableDeclarations
        cursorDeclarations
BEGIN
        PL/SQL_statements
EXCEPTION
        exceptionHandling
END;
```

FUNCTIONS

Functions are similar to stored procedures, with a few differences. A function must return a value, while a stored procedure can return a value, a result set or nothing. Some functions can potentially be used inside a select statement, while stored procedures must always be executed. Additionally, a function can only have input parameters, while a stored procedure may have input, output and/or input/output parameters.

The following partial example syntax can be used to create a function in Oracle:

```
CREATE OR REPLACE FUNCTION functionName(parameterList)
RETURN returnVariable
IS
        variableDeclarations
        cursorDeclarations
BEGIN
        PL/SQL_statements
EXCEPTION
        exceptionHandling
END;
```

The following partial example syntax can be used to test some functions in Oracle:

```
SELECT myFunction(parameters) FROM DUAL;
```

PACKAGES

In Oracle, a package is a collection of certain related schema objects, such as stored procedures and functions. One of the numerous advantages of using packages is that certain schema objects such as stored procedures can be overloaded. Packages also allow for better performance, since the entire package is loaded into memory whenever any package subprogram is called for the first time.

The following partial example syntax can be used to create a package in Oracle (body omitted):

CREATE OR REPLACE PACKAGE *packageName* AS
    listOfProcedures
    listOfFunctions
    listOfExceptions
END *packageName*;

RELATION SCHEMA AND CONTENT

EMPLOYEE

```
CS342 SQL> desc tr_Employee;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------
 EMPLOYEEID                                                              NOT NULL NUMBER
 NAMEFIRST                                                               NOT NULL VARCHAR2(20)
 NAMEMIDDLE                                                                       VARCHAR2(20)
 NAMELAST                                                                NOT NULL VARCHAR2(40)
 BIRTHDATE                                                               NOT NULL DATE
 SEX                                                                     NOT NULL CHAR(1)
 ADDRESSSTREET1                                                          NOT NULL VARCHAR2(40)
 ADDRESSSTREET2                                                                   VARCHAR2(40)
 ADDRESSCITY                                                             NOT NULL VARCHAR2(25)
 ADDRESSSTATE                                                            NOT NULL CHAR(2)
 ADDRESSZIP                                                              NOT NULL VARCHAR2(20)

CS342 SQL> select * from tr_Employee;

EMPLOYEEID NAMEFIRST NAMEMIDDLE NAMELAST   BIRTHDATE S ADDRESSSTREET1        ADDRESSSTREET2 ADDRESSCITY  AD ADDRESSZIP
---------- --------- ---------- ---------- --------- - -------------------- -------------- ------------ -- ----------
         1 Travis    Ray        Ragle      03-DEC-79 M 123 Fake St.         Apt. 243       Bakersfield  CA 93307
         2 Jennifer  Lynn       Gardner    05-APR-66 F 314 Roy St.                         Houston      TX 77077
         3 Jerry     Lee        Lewis      01-MAR-56 M 78 One Way                          Bakersfield  CA 93312
         4 Chelsy    Bee        Ray        25-APR-83 F 123 Wrong Way        Apt. 27        Los Angeles  CA 90001
         5 Homer     Jay        Simpson    11-NOV-69 M 742 Evergreen Terrace               Springfield  IL 62701
         6 Spencer   Drew       Crawford   02-JAN-80 M 8694 Rocker St.                     Los Angeles  CA 90001
         7 Clancy    Raymond    Wiggim     08-APR-72 M 45 Carter Park                      Springfield  IL 62701
         8 Terra     Renee      Elizabeth  06-JUN-84 F 87 Fairfield Rd.                    Bakersfield  CA 93311
         9 Karen     Sue        Nancy      23-SEP-76 F 908 China Grade Lp                  Bakersfield  CA 93309
        10 Ryan      Donald     Rickner    21-FEB-85 M 71 Smith Way                        Berkeley     CA 94701
        11 Cloud     Michael    Strife     02-APR-69 M 333 Rocker Street                   Berkeley     CA 94701
        12 Tifa      Anya       Reck       11-MAR-86 F 75 Smith Way                        Bakersfield  CA 93301
        13 Lisa      Nancy      Far        21-JAN-84 F 3 What Way                          Berkeley     CA 94701
        14 Hank      Andrew     Goode      21-FEB-75 M 32 Hill Street                      Bakersfield  CA 93312
        15 Wade      Louis      Black      22-AUG-78 M 132 Smith Way                       Berkeley     CA 94701
        16 Marge                Simpson    13-MAY-72 F 742 Evergreen Terace                Springfield  IL 62701
        17 Bart                 Simpson    21-FEB-85 M 742 Evergreen Terrace               Springfield  IL 62701
        18 Hank                 Hill       11-APR-65 M 309 Cheater Way                     Houston      TX 77077
        19 Lisa      Marie      Simpson    21-JAN-87 F 742 Evergreen Terrace               Springfield  IL 62701
        20 Peggy                Hill       06-MAR-70 F 309 Cheater Way                     Houston      TX 77077
        21 Dale                 Cribbel    01-DEC-72 M 304 Cheater Way                     Houston      TX 77077
        22 Nacy                 Cribbel    11-NOV-76 F 304 Cheater Way                     Houston      TX 77077
        23 Raymond              Love       11-MAR-82 M 111 Smith Way                       Berkeley     CA 94701
        24 Kyle                 Broflowski 27-JUL-87 M 23 Roof Street                      Denver       CO 80203
        25 Stan                 Marsh      08-JUN-87 M 24 Roof Street                      Denver       CA 80203
        26 Kenny                Kenny      15-FEB-87 M 27 Roof Street                      Denver       CA 80203
        27 Erik                 Cartman    13-JAN-87 M 32 Roof Street                      Denver       CA 80203
        28 Shannon              Niece      21-MAY-81 F 304 Royanne Street                  Bakersfield  CA 93307
        29 Brittany             Murphy     11-FEB-78 F 101 Centennial Way                  Bakersfield  CA 93312
        30 Jeff      Scott      Angel      12-JUN-83 M 345 Akers Way                       Bakersfield  CA 93306
        31 Roger     Lucas      Mark       06-JUN-74 M 4005 Panama Lane                    Bakersfield  CA 93306
        32 Wayland              Smithers   25-AUG-73 M 101 Charles Way                     Springfield  IL 62701
        33 Naomi     Watts      Smith      03-MAR-86 F 321 Smith Way                       Berkeley     CA 94701
        34 Anne                 Ray        12-MAY-76 F 2003 White Lane                     Bakersfield  CA 93306
        35 David                Andrew     23-MAR-83 M 176 University Lane                 Bakersfield  CA 93307
        36 Lucky                Rivers     16-JUN-86 M 2020 Fairfax Road                   Bakersfield  CA 93307
        37 Buzz                 Night      03-JAN-81 M 6067 Stockdale Lane                 Bakersfield  CA 93306

EMPLOYEEID NAMEFIRST NAMEMIDDLE NAMELAST   BIRTHDATE S ADDRESSSTREET1        ADDRESSSTREET2 ADDRESSCITY  AD ADDRESSZIP
---------- --------- ---------- ---------- --------- - -------------------- -------------- ------------ -- ----------
        38 Lenny                Williams   24-APR-74 M 1040 Heart Way                      Springfield  IL 62701
        39 Carl                 Williams   26-JUN-76 M 1040 Heart Way                      Springfield  IL 62701
```

```
        40 Jason     Lee        Smitts     14-APR-84 M 999 Smith Way                    Berkeley     CA 94701
        41 Khalisa              Evans      21-FEB-78 F 314 Rush Street                  Bakersfield  CA 93307
        42 Herbet               Lewis      11-JUL-81 M 1011 El Paso Rd.                 Bakersfield  CA 93311
        43 Matt                 Stone      13-DEC-76 M 111 Thin Way                     Denver       CO 80203
        44 Arnold               Strong     09-MAR-67 M 607 Strong Way                   Bakersfield  CA 93313
        45 John                 Paul       21-AUG-78 M 333 Simpson Street               Bakersfield  CA 93312
        46 Jack                 Pot        21-FEB-85 M 447 Smith Way                    Bakersfield  CA 93313

46 rows selected.
```

DEPARTMENT

```
CS342 SQL> desc tr_Department;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 DEPARTMENTID                                              NOT NULL NUMBER
 NAME                                                      NOT NULL VARCHAR2(20)
 DESCRIPTION                                                        VARCHAR2(256)
 PARENTID                                                           NUMBER

CS342 SQL> select * from tr_Department;

DEPARTMENTID NAME                 DESCRIPTION                                                    PARENTID
------------ -------------------- ------------------------------------------------------------ --------
           1 Art
           2 Sound
           3 Concept Art                                                                           1
           4 Model Design         This group creates 3d models based on concept designs.           1
           5 Game Programmers
           6 IT
           7 Webmasters           All web work will be handled by this group.                      6
           8 DBAs                 All database work will be handled by this group.                 6
           9 Game Tools           Development tools, etc.                                          5
          10 Gameplay             Specific work on specific games.                                 5
          11 Quality Assurance    All testing will be handled by this group.

11 rows selected.
```

POSITION

```
CS342 SQL> desc tr_Position;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 POSITIONID                                                NOT NULL NUMBER
 TITLE                                                     NOT NULL VARCHAR2(32)
 DESCRIPTION                                                        VARCHAR2(256)
 PARENTID                                                           NUMBER

CS342 SQL> select * from tr_Position;

POSITIONID TITLE                          DESCRIPTION                                                 PARENTID
---------- ------------------------------ ------------------------------------------------------------ --------
         1 Lead Programmer
         2 Physics Programmer             Simulates physics, collision, etc.                              1
         3 AI Programmer                  Rule-based decisions, scripting                                 1
         4 Sound Programmer               Speech, effects, music                                          1
         5 UI Programmer                  Menus, etc.                                                     1
         6 Lead Artist
         7 Texture Artist                                                                                 6
         8 3D Character Artist                                                                            6
         9 3D Environment Artist                                                                          6
        10 Sprite Artist                                                                                  6
        11 Lead Quality Assurance         General testing

11 rows selected.
```

VIDEO GAME

```
CS342 SQL> desc tr_Videogame;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 VIDEOGAMEID                                                             NOT NULL NUMBER
 TITLE                                                                   NOT NULL VARCHAR2(20)
 DESCRIPTION                                                                      VARCHAR2(256)
 RELEASEDATE                                                                      DATE
 WEBSITE                                                                          VARCHAR2(50)
 PARENTID                                                                         NUMBER
 SERIESID                                                                NOT NULL NUMBER
 ENTRY                                                                            NUMBER
 GENREID                                                                 NOT NULL NUMBER
 CONSOLEREGIONID                                                         NOT NULL NUMBER

CS342 SQL> select * from tr_Videogame;

VIDOGAMID TITLE            DESCRIPTION RELEASEDATE WEBSITE                    PARENTID SERIESID ENTRY GENREID CONSOLEREGIONID
--------- ---------------- ----------- ----------- -------------------------- -------- -------- ----- ------- ---------------
      151 Lizard Quest                04-MAY-85   www.lizardquest.com/lq/nes                1     1       1               1
      152 Lizard Quest II             08-AUG-86   www.lizardquest.com/lq2/nes               1     2       1               1
      153 Lizard Quest III            04-MAY-88   www.lizardquest.com/lq3/nes               1     3       1               1
      201 Lizard Quest     SNES Remake 01-JAN-92   www.lizardquest.com/lq/snes               1     1       1              16
      301 Lizard Quest     PS1 Remake  01-JAN-97   www.lizardquest.com/lq/ps1       151      1     1       2              41
      334 Lizard Quest     PSP Remake  01-JAN-07   www.lizardquest.com/lq/psp       151      1     1       2              49
      335 Lizard Quest II  PSP Remake  01-FEB-07   www.lizardquest.com/lq2/psp      152      1     2       2              49
      345 Vito                        04-FEB-07                                     21      1     5       2              49
      347 Rainfall                    23-MAR-07                                     25      1     8       2              49
      354 Autumn Knight               07-JUL-07                                     26      1     3       2              49
      356 Autumn Knight II                                                          26      2     3       2              49

11 rows selected.
```

PHASE

```
CS342 SQL> desc tr_Phase;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 PHASEID                                                                 NOT NULL NUMBER
 NAME                                                                    NOT NULL VARCHAR2(20)
 DESCRIPTION                                                                      VARCHAR2(256)
 PARENTID                                                                         NUMBER

CS342 SQL> select * from tr_Phase;

PHASEID NAME                           DESCRIPTION                                          PARENTID
------- ------------------------------ ---------------------------------------------------- --------
      1 Phase I
      2 Design                         Phase I ? Design only                                       1
      3 Phase II
      4 Programming                    Phase II ? Programming only                                 3
      5 Level Creation                 Phase II ? Level Creation only                              3
      6 Phase III
      7 Testing                        Phase III ? Testing only                                    6
      8 Bug Fix                        Phase III ? Bug Fixes only                                  6
      9 Clean-up                       Phase III ? Clean-up only                                   6
     10 Released
     11 Patching                                                                                 10
```

```
11 rows selected.
```

MILESTONE

```
CS342 SQL> desc tr_Milestone;
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- ------------------------------------------------
 MILESTONEID                                               NOT NULL NUMBER
 NAME                                                      NOT NULL VARCHAR2(32)
 DESCRIPTION                                                        VARCHAR2(256)

CS342 SQL> select * from tr_Milestone;

MILESTONEID NAME                           DESCRIPTION
----------- ------------------------------ --------------------------------------------------
          1 Approved For Production
          2 Voice Work Complete
          3 First Model Designed
          4 First Level Designed
          5 Concept Art Complete
          6 First Playable                 First version that is playable
          7 Alpha                          Key gameplay implemented
          8 Beta                           Only bugs are being worked on, no new code
          9 Code Release                   Game is ready to ship
         10 Gold Master                    Final build

10 rows selected.
```

GENRE

```
CS342 SQL> desc tr_Genre;
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- ------------------------------------------------
 GENREID                                                   NOT NULL NUMBER
 NAME                                                      NOT NULL VARCHAR2(32)
 DESCRIPTION                                                        VARCHAR2(256)
 PARENTID                                                           NUMBER

CS342 SQL> select * from tr_Genre;

GENREID NAME                           DESCRIPTION                                                                  PARENTID
------- ------------------------------ ---------------------------------------------------------------------------- --------
      1 Role-playing
      2 Traditional Role-playing       Ultima-esque RPGs                                                               1
      3 Action Role-playing            RPGs featuring action gameplay                                                  1
      4 Action
      5 Platform                       Includes 2d and 3d platformers                                                  4
      6 Third Person Shooter                                                                                           4
      7 Adventure
      8 Interactive Movie              Adventure game that uses full motion video                                      7
      9 Visual Novel                   Interactive fiction game with mostly static graphics                            7
     10 Dating Sim                                                                                                    7

10 rows selected.
```

SERIES

```
CS342 SQL> desc tr_Series;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 SERIESID                                                                NOT NULL NUMBER
 NAME                                                                    NOT NULL VARCHAR2(20)

CS342 SQL> select * from tr_Series;

SERIESID NAME
-------- ------------------------------
       1 Lizard Quest
       5 First Shot
       6 Racing!
       7 Puzzle Boy
       8 The Last Fighter
       9 Pure Paranoia
      10 NOW
      21 Vito
      25 Rainfall
      26 Autumn Knight

10 rows selected.
```

CONSOLE

```
CS342 SQL> desc tr_Console;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 CONSOLEID                                                               NOT NULL NUMBER
 NAME                                                                    NOT NULL VARCHAR2(32)

CS342 SQL> select * from tr_Console;

 CONSOLEID NAME
---------- --------------------------------
         1 Nintendo Entertainment System
         2 Famicom
         3 Sega Master System
         4 Super Nintendo
         5 Super Famicom
         6 Sega Genesis
         7 Nintendo 64
         8 Playstation
         9 Playstation Portable
        10 Playstation 3

10 rows selected.
```

REGION

```
CS342 SQL> desc tr_Region;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 REGIONID                                                                NOT NULL NUMBER
 LOCATION                                                                NOT NULL VARCHAR2(20)

CS342 SQL> select * from tr_Region;

  REGIONID LOCATION
---------- --------------------
         1 Japan
         2 North America
         3 Asia
         4 Australia
         5 England
         6 Germany
         7 Italy
         8 France
         9 Spain
        10 Middle East

10 rows selected.
```

WORKS ON

```
CS342 SQL> desc tr_WorksOn;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 VIDEOGAMEID                                                             NOT NULL NUMBER
 EMPLOYEEID                                                              NOT NULL NUMBER
 POSITIONID                                                              NOT NULL NUMBER
 STARTDATE                                                               NOT NULL DATE
 ENDDATE                                                                          DATE

CS342 SQL> select * from tr_WorksOn;

VIDEOGAMEID EMPLOYEEID POSITIONID STARTDATE ENDDATE
----------- ---------- ---------- --------- ---------
        151          2          1 01-NOV-84 04-MAY-85
        151          2          6 01-NOV-84 04-MAY-85
        151          3          1 01-NOV-84 04-MAY-85
        151          3          6 01-NOV-84 04-MAY-85
        152          2          6 01-NOV-85 08-AUG-86
        152          2          1 01-NOV-85 08-AUG-86
        152          3          1 01-NOV-85 08-AUG-86
        152          3          6 01-NOV-85 08-AUG-86
        153          2          1 01-FEB-87 04-MAY-88
        153          2          6 01-FEB-87 04-MAY-88
        153          3          1 01-FEB-87 04-MAY-88
        153          3          6 01-FEB-87 04-MAY-88
        201          2          6 26-OCT-90 12-NOV-91
        201          2         10 04-FEB-91 10-OCT-91
        201          3          1 05-MAY-90 18-DEC-91
        201          3          2 08-AUG-90 01-OCT-91
        201          3          3 04-FEB-91 05-JUN-91
        201          5          2 08-AUG-90 01-OCT-91
        201          5          5 05-MAY-91 01-OCT-91
        201          7          4 13-APR-91 11-NOV-91
```

```
        201          7          7 04-FEB-91 05-SEP-91
        201          7         10 04-FEB-91 08-AUG-91
        301          2          6 07-MAY-95 12-DEC-96
        301          2          7 05-JUN-95 07-AUG-96
        301          3          1 07-MAY-95 20-DEC-96
        301          3          2 07-MAY-95 11-NOV-96
        301          3          3 07-MAY-95 18-NOV-96
        301          5          2 07-MAY-95 20-DEC-96
        301          5          4 04-MAR-96 22-NOV-96
        301          5          5 02-APR-96 22-NOV-96
        301          7          8 09-SEP-95 10-NOV-96
        301          7          9 09-SEP-95 21-SEP-96
        301          9          8 12-NOV-95 10-NOV-96
        301          9          9 12-NOV-95 21-SEP-96
        334          2          6 01-APR-05 12-DEC-06
        334          2          7 01-APR-05 03-SEP-06
        334          4          7 05-MAY-05 02-SEP-06

VIDEOGAMEID EMPLOYEEID POSITIONID STARTDATE ENDDATE
----------- ---------- ---------- --------- ---------
        334          4          8 05-MAY-05 02-OCT-06
        334          4          9 05-MAY-05 17-OCT-06
        334          5          1 01-APR-05 22-DEC-06
        334          5          2 01-APR-05 16-NOV-06
        334          5          5 12-JUN-06 01-DEC-06
        334          6          3 05-JUN-05 14-DEC-06
        334          6          4 07-JUL-06 02-DEC-06
        334          7          8 01-APR-05 23-OCT-06
        334          7          9 01-APR-05 12-SEP-06
        334          8          8 07-JUN-05 15-OCT-06
        334          9          8 01-APR-05 12-OCT-06
        334          9          9 01-APR-05 13-OCT-06
        334         10          2 06-JUL-05 01-DEC-06
        334         10          3 06-JUL-05 01-DEC-06
        335          1          4 01-JUL-06 06-JUL-06
        335          2          6 01-APR-05 12-DEC-06
        335          2          7 01-APR-05 03-SEP-06
        335          4          7 05-MAY-05 02-SEP-06
        335          4          8 05-MAY-05 02-OCT-06
        335          4          9 05-MAY-05 17-OCT-06
        335          5          1 01-APR-05 22-DEC-06
        335          5          2 01-APR-05 16-NOV-06
        335          5          5 12-JUN-06 01-DEC-06
        335          6          3 05-JUN-05 14-DEC-06
        335          6          4 07-JUL-06 02-DEC-06
        335          7          8 01-APR-05 23-OCT-06
        335          7          9 01-APR-05 12-SEP-06
        335          8          8 07-JUN-05 15-OCT-06
        335          9          8 01-APR-05 12-OCT-06
        335          9          9 01-APR-05 13-OCT-06
        335         10          2 06-JUL-05 01-DEC-06
        335         10          3 06-JUL-05 01-DEC-06
        356         34          1 10-OCT-10

70 rows selected.
```

EMPLOYEE BELONGS TO A DEPARTMENT

```
CS342 SQL> desc tr_EmployeeBelongsToDepartment;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 EMPLOYEEID                                                              NOT NULL NUMBER
 DEPARTMENTID                                                            NOT NULL NUMBER
 STARTDATE                                                               NOT NULL DATE
 ENDDATE                                                                          DATE

CS342 SQL> select * from tr_EmployeeBelongsToDepartment;

EMPLOYEEID DEPARTMENTID STARTDATE ENDDATE
---------- ------------ --------- ---------
         1            1 05-APR-04 03-MAY-06
         1            9 10-SEP-09
         1           10 04-MAY-06 09-SEP-09
         2            1 01-NOV-84 02-FEB-03
         3            5 01-NOV-84 03-MAY-06
         3            3 04-MAY-06
         4            1 07-JUN-04 03-MAY-06
         4            4 04-MAY-06 10-OCT-10
         4           11 11-OCT-10
         5            5 09-SEP-89 03-MAY-06
         5           10 04-MAY-06
         6            5 08-FEB-05 03-MAY-06
         6           10 04-MAY-06
         7            1 05-OCT-89 03-MAY-06
         7            4 04-MAY-06
         8            1 07-JUL-04 03-MAY-06
         8            4 04-MAY-06 08-SEP-09
         8            3 09-SEP-09
         9            1 21-OCT-89 03-MAY-06
         9            4 04-MAY-06
        10            5 03-JAN-05 03-MAY-06
        10           10 04-MAY-06 08-JUN-08
        10            8 09-JUN-08 01-FEB-10
        10           10 02-FEB-10 23-FEB-10
        10            9 24-FEB-10
        11            2 03-JUN-07
        12            7 04-MAY-08 07-JUL-10
        12            8 08-JUL-10
        13           11 02-FEB-08
        14            4 03-FEB-08 11-OCT-10
        14            3 12-OCT-10
        15            7 06-APR-08
        16            4 23-APR-08
        17           10 03-JUN-08
        18           10 03-JUN-08 08-AUG-08
        19           10 03-JUN-08
        20           10 03-JUN-08

EMPLOYEEID DEPARTMENTID STARTDATE ENDDATE
---------- ------------ --------- ---------
        21            4 03-JUN-08 06-JUN-08
        22            4 03-JUN-08
        23           11 05-JUL-08
        24           11 24-JUL-08 03-SEP-09
        25            2 04-SEP-09
        26            7 05-AUG-08
        27            8 01-OCT-08 01-OCT-10
        27           10 02-OCT-10
        28           10 01-FEB-09
        29           10 04-FEB-09 08-JUL-09
        29            9 09-JUL-09
        30            4 05-MAR-09
        31            4 05-APR-09
```

```
        32           11 05-MAY-09
        33            8 04-APR-09 05-APR-09
        33           10 06-APR-09 06-MAY-09
        33            8 07-MAY-09
        34           10 03-MAR-10
        35           10 03-MAR-10
        36           10 03-MAR-10
        37           10 03-MAR-10 20-MAR-10
        38            4 03-MAR-10 03-JUN-10
        39           10 03-MAR-10
        40            4 03-MAR-10
        41            4 03-MAR-10 20-MAR-10
        42           10 03-MAR-10
        43            4 03-MAR-10
        44            7 07-JUN-10
        45            8 05-AUG-10
        46           11 10-OCT-10

67 rows selected.
```

CONSOLE IS IN A REGION

```
CS342 SQL> desc tr_ConsoleIsInRegion;
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- ------------------------------------------
 CONSOLEREGIONID                                           NOT NULL NUMBER
 CONSOLEID                                                 NOT NULL NUMBER
 REGIONID                                                  NOT NULL NUMBER
 RELEASEDATE                                                        DATE
 WEBSITE                                                            VARCHAR2(50)

CS342 SQL> select * from tr_ConsoleIsInRegion;

CONSOLEREGIONID  CONSOLEID   REGIONID RELEASEDATE WEBSITE
--------------- ---------- ---------- ----------- --------------------------
              1          1          2 18-OCT-85   www.nintendo.com
              2          1          5 01-SEP-86   www.nintendo.com
              3          1          6 01-SEP-86   www.nintendo.com
              4          1          7 01-SEP-86   www.nintendo.com
              5          1          8 01-SEP-86   www.nintendo.com
              6          1          9 01-SEP-86   www.nintendo.com
              7          1          4 01-SEP-87   www.nintendo.com
              8          2          1 15-JUL-83   www.nintendo.com
              9          3          1 05-JUL-89   www.sega.com
             10          3          2 03-APR-92   www.sega.com
             11          3          5 01-SEP-96   www.sega.com
             12          3          6 01-SEP-96   www.sega.com
             13          3          7 01-SEP-96   www.sega.com
             14          3          8 01-SEP-96   www.sega.com
             15          3          9 01-SEP-96   www.sega.com
             16          4          2 23-AUG-91   www.nintendo.com
             17          4          5 11-APR-92   www.nintendo.com
             18          4          6 11-APR-92   www.nintendo.com
             19          4          7 11-APR-92   www.nintendo.com
             20          4          8 11-APR-92   www.nintendo.com
             21          4          9 11-APR-92   www.nintendo.com
             22          4          4 03-APR-92   www.nintendo.com
             23          5          1 21-NOV-90   www.nintendo.com
             24          6          1 29-OCT-88   www.sega.com
             25          6          2 14-AUG-89   www.sega.com
             26          6          4 30-NOV-90   www.sega.com
             27          6          5 30-NOV-90   www.sega.com
             28          6          6 30-NOV-90   www.sega.com
             29          6          7 30-NOV-90   www.sega.com
             30          6          8 30-NOV-90   www.sega.com
             31          6          9 30-NOV-90   www.sega.com
             32          7          1 23-JUN-96   www.nintendo.com
             33          7          2 29-SEP-96   www.nintendo.com
             34          7          4 01-MAR-97   www.nintendo.com
             35          7          5 01-MAR-97   www.nintendo.com
             36          7          6 01-MAR-97   www.nintendo.com
             37          7          7 01-MAR-97   www.nintendo.com

CONSOLEREGIONID  CONSOLEID   REGIONID RELEASEDATE WEBSITE
--------------- ---------- ---------- ----------- --------------------------
             38          7          8 01-MAR-97   www.nintendo.com
             39          7          9 01-MAR-97   www.nintendo.com
             40          8          1 03-DEC-94   jp.playstation.com
             41          8          2 09-SEP-95   us.playstation.com
             42          8          5 29-SEP-95   uk.playstation.com
             43          8          6 29-SEP-95   de.playstation.com
             44          8          7 29-SEP-95   it.playstation.com
             45          8          8 29-SEP-95   fr.playstation.com
             46          8          9 29-SEP-95   es.playstation.com
             47          8          4 15-NOV-95   au.playstation.com/psp
             48          9          1 12-DEC-04   jp.playstation.com/psp
             49          9          2 24-MAR-05   us.playstation.com/psp
```

```
        50          9          4 01-SEP-05   au.playstation.com/psp
        51          9          5 01-SEP-05   uk.playstation.com/psp
        52          9          6 01-SEP-05   de.playstation.com/psp
        53          9          7 01-SEP-05   it.playstation.com/psp
        54          9          8 01-SEP-05   fr.playstation.com/psp
        55          9          9 01-SEP-05   es.playstation.com/psp
        56         10          1 11-NOV-06   jp.playstation.com/ps3
        57         10          2 17-NOV-06   us.playstation.com/ps3
        58         10          3 17-NOV-06   asia.playstation.com
        59         10          4 23-MAR-07   au.playstation.com/ps3
        60         10          5 16-MAR-07   uk.playstation.com/ps3

60 rows selected.
```

VIDEO GAME IS IN A PHASE

```
CS342 SQL> desc tr_VideogameIsInPhase;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 VIDEOGAMEID                                                             NOT NULL NUMBER
 PHASEID                                                                 NOT NULL NUMBER
 STARTDATE                                                               NOT NULL DATE
 ENDDATE                                                                          DATE

CS342 SQL> select * from tr_VideogameIsInPhase;

VIDEOGAMEID PHASEID STARTDATE ENDDATE
----------- ------- --------- ---------
        334       2 01-APR-04 07-MAY-04
        334       1 08-MAY-04 02-OCT-04
        334       5 03-OCT-04 03-FEB-05
        334       3 04-FEB-05 04-MAY-05
        334       4 05-MAY-05 06-JUL-06
        334       6 07-JUL-06 07-SEP-06
        334       7 08-SEP-06 23-OCT-06
        334       8 24-OCT-06 01-DEC-06
        334       9 01-DEC-06 31-DEC-06
        334      10 01-JAN-07 01-JAN-07
        335       2 01-APR-04 07-MAY-04
        335       1 08-MAY-04 02-OCT-04
        335       5 03-OCT-04 03-FEB-05
        335       3 04-FEB-05 04-MAY-05
        335       4 05-MAY-05 06-JUL-06
        335       6 07-JUL-06 07-SEP-06
        335       7 08-SEP-06 23-OCT-06
        335       8 24-OCT-06 01-DEC-06
        335       9 01-DEC-06 31-DEC-06
        335      10 01-FEB-07 01-FEB-07
        345       2 01-FEB-04 03-APR-04
        345       1 04-APR-04 07-SEP-04
        345       5 08-SEP-04 04-MAR-05
        345       3 05-MAR-05 08-MAY-05
        345       4 09-MAY-05 01-JUL-06
        345       6 02-JUL-06 01-SEP-06
        345       7 02-SEP-06 31-OCT-06
        345       8 01-NOV-06 19-DEC-06
        345       9 20-DEC-06 15-JAN-07
        345      10 04-FEB-07 04-FEB-07
        347       2 01-OCT-03 15-DEC-03
        347       1 16-DEC-03 22-MAR-04
        347       5 23-MAR-04 13-JAN-05
        347       3 14-JAN-05 22-MAR-05
        347       4 23-MAR-05 01-JUN-06
        347       6 02-JUN-06 27-AUG-06
        347       7 28-AUG-06 01-OCT-06

VIDEOGAMEID PHASEID STARTDATE ENDDATE
----------- ------- --------- ---------
        347       8 02-OCT-06 29-DEC-06
        347       9 30-DEC-06 31-JAN-07
        347      10 23-MAR-07 23-MAR-07
        354       2 01-FEB-04 05-MAY-04
        354       1 06-MAY-04 22-JUL-04
        354       5 23-JUL-04 31-JAN-05
        354       3 01-FEB-05 22-MAR-05
        354       4 23-MAR-05 02-SEP-05
        354       6 03-SEP-05 31-JAN-06
        354       7 01-FEB-06 01-APR-06
        354       8 02-APR-06 29-MAY-06
        354       9 30-MAY-06 30-JUN-07
        354      10 07-JUL-07 07-JUL-07
```

```
       356       2 07-JUN-08 11-DEC-08
       356       1 12-DEC-08 01-JAN-09
       356       5 02-JAN-09 02-APR-09
       356       3 03-APR-09 22-JUN-09
       356       4 23-JUN-09 11-JAN-10
       356       6 12-JAN-10 22-FEB-10
       356       7 23-FEB-10 01-JUN-10
       356       8 02-JUN-10 22-AUG-10
       356       9 23-AUG-10 10-SEP-10
       356      10 10-OCT-10 10-OCT-10

60 rows selected.
```

VIDEO GAME HAS REACHED A MILESTONE

```
CS342 SQL> desc tr_VideogameReachedMilestone;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------------------
 VIDEOGAMEID                                                             NOT NULL NUMBER
 MILESTONEID                                                             NOT NULL NUMBER
 DATEREACHED                                                             NOT NULL DATE

CS342 SQL> select * from tr_VideogameReachedMilestone;

VIDEOGAMEID MILESTONEID DATEREACHED
----------- ----------- -----------
        334           1 01-FEB-04
        334           2 12-NOV-04
        334           3 13-OCT-04
        334           4 03-FEB-05
        334           5 11-NOV-04
        334           6 08-JUN-06
        334           7 07-SEP-06
        334           8 24-OCT-06
        334           9 14-DEC-06
        334          10 15-DEC-06
        335           1 01-FEB-04
        335           2 12-NOV-04
        335           3 13-OCT-04
        335           4 03-FEB-05
        335           5 11-NOV-04
        335           6 08-JUN-06
        335           7 07-SEP-06
        335           8 24-OCT-06
        335           9 14-DEC-06
        335          10 15-DEC-06
        345           1 12-DEC-03
        345           2 12-NOV-04
        345           3 05-JUL-04
        345           4 04-MAR-05
        345           5 01-FEB-05
        345           6 23-JUN-06
        345           7 10-SEP-06
        345           8 01-NOV-06
        345           9 15-JAN-07
        345          10 23-JAN-07
        347           1 04-JUL-03
        347           2 15-DEC-04
        347           3 01-FEB-04
        347           4 13-JAN-05
        347           5 16-DEC-04
        347           6 01-APR-06
        347           7 18-AUG-06

VIDEOGAMEID MILESTONEID DATEREACHED
----------- ----------- -----------
        347           8 02-OCT-06
        347           9 01-FEB-07
        347          10 03-MAR-07
        354           1 12-DEC-03
        354           2 11-NOV-04
        354           3 01-MAR-04
        354           4 30-JAN-05
        354           5 12-NOV-04
        354           6 06-AUG-05
        354           7 15-JAN-06
        354           8 02-APR-06
        354           9 20-JUN-07
        354          10 22-JUN-07
        356           1 05-MAY-08
```

```
       356           2 09-SEP-08
       356           3 02-JUN-09
       356           4 02-APR-09
       356           5 01-APR-09
       356           6 02-JAN-10
       356           7 13-FEB-10
       356           8 02-JUN-06
       356           9 10-SEP-10
       356          10 22-SEP-10

60 rows selected.
```

QUERIES

Select all female employees born after January 1, 1980

```
SELECT NameLast || ', ' || NameFirst || NVL(' ' || NameMiddle, '') AS Name
FROM   tr_Employee
WHERE  Sex = 'F' AND
       Birthdate > to_date('01/01/1980','mm/dd/yyyy')
ORDER BY 1;
```

```
NAME
----------------------------------------
Elizabeth, Terra Renee
Far, Lisa Nancy
Niece, Shannon
Ray, Chelsy Bee
Reck, Tifa Anya
Simpson, Lisa Marie
Smith, Naomi Watts

7 rows selected.
```

Select all games that Travis Ragle has worked on

```
SELECT UNIQUE v.Title
FROM   tr_Videogame v INNER JOIN
       tr_WorksOn w ON w.VideogameID = v.VideogameID INNER JOIN
       tr_Employee e ON e.EmployeeID = w.EmployeeID
WHERE  e.NameFirst = 'Travis' AND
       e.NameLast = 'Ragle'
ORDER BY 1;
```

```
TITLE
------------------
Lizard Quest II
```

Select all employees that have worked on every game that Travis Ragle has worked on.

```
SELECT  e.NameLast || ', ' || e.NameFirst || NVL(' ' || e.NameMiddle, '') AS Name
FROM    tr_Employee e
WHERE   (e.NameLast != 'Ragle' OR e.NameFirst != 'Travis') AND NOT EXISTS (
        SELECT *
        FROM    tr_Employee e2 INNER JOIN
                tr_WorksOn w2 ON e2.EmployeeID = w2.EmployeeID
        WHERE   e2.NameLast = 'Ragle' AND
                e2.NameFirst = 'Travis' AND NOT EXISTS (
                SELECT  *
                FROM    tr_WorksOn w
                WHERE   w.EmployeeID = e.EmployeeID AND
                        w.VideogameID = w2.VideogameID
                )
        )
ORDER BY 1;
```

```
NAME
----------------------------------------
Crawford, Spencer Drew
Elizabeth, Terra Renee
Gardner, Jennifer Lynn
Nancy, Karen Sue
Ray, Chelsy Bee
Rickner, Ryan Donald
```

```
Simpson, Homer Jay
Wiggim, Clancy Raymond

8 rows selected.
```

Select the second oldest employee

```
SELECT e.NameLast || ', ' || e.NameFirst || NVL(' ' || e.NameMiddle, '') AS Name
       ,e.Birthdate
FROM   tr_Employee e INNER JOIN
       (
       SELECT e1.EmployeeID
       FROM   tr_Employee e1
              ,tr_Employee e2
       WHERE  e1.Birthdate > e2.Birthdate
       MINUS
       SELECT e3.EmployeeID
       FROM   tr_Employee e3
              ,tr_Employee e4
              ,tr_Employee e5
       WHERE  e3.Birthdate > e4.Birthdate AND
              e4.Birthdate > e5.Birthdate
       ) o ON e.EmployeeID = o.EmployeeID;
```

```
NAME                                   BIRTHDATE
-------------------------------------- ---------
Hill, Hank                             11-APR-65
```

Select all employees that have not worked on a Lizard Quest game

```
SELECT e.NameLast || ', ' || e.NameFirst || NVL(' ' || e.NameMiddle, '') AS Name
FROM   tr_Employee e
WHERE  NOT EXISTS (
       SELECT *
       FROM   tr_WorksOn w INNER JOIN
              tr_Videogame v ON w.VideogameID = v.VideogameID INNER JOIN
              tr_Series s ON s.SeriesID = v.SeriesID
       WHERE  w.EmployeeID = e.EmployeeID AND
              s.Name = 'Lizard Quest'
       )
ORDER BY 1;
```

```
NAME
----------------------------------------
Andrew, David
Angel, Jeff Scott
Black, Wade Louis
Broflowski, Kyle
Cartman, Erik
Cribbel, Dale
Cribbel, Nacy
Evans, Khalisa
Far, Lisa Nancy
Goode, Hank Andrew
Hill, Hank
Hill, Peggy
Kenny, Kenny
Lewis, Herbet
Love, Raymond
Mark, Roger Lucas
Marsh, Stan
Murphy, Brittany
Niece, Shannon
Night, Buzz
```

```
Paul, John
Pot, Jack
Ray, Anne
Reck, Tifa Anya
Rivers, Lucky
Simpson, Bart
Simpson, Lisa Marie
Simpson, Marge
Smith, Naomi Watts
Smithers, Wayland
Smitts, Jason Lee
Stone, Matt
Strife, Cloud Michael
Strong, Arnold
Williams, Carl
Williams, Lenny

36 rows selected.
```

Select all games worked on in 2005

```
SELECT UNIQUE v.Title
FROM   tr_Videogame v INNER JOIN
       tr_WorksOn w ON w.VideogameID = v.VideogameID
WHERE  w.StartDate < to_date('01/01/2006','mm/dd/yyyy') OR
       w.EndDate >= to_date('01/01/2005','mm/dd/yyyy')
ORDER BY 1;
```

```
TITLE
------------------
Lizard Quest
Lizard Quest II
Lizard Quest III
```

Select all employees employed since 2005 (or before) or that were hired on or after 1/1/2010

```
SELECT e.NameLast || ', ' || e.NameFirst || NVL(' ' || e.NameMiddle, '') AS Name
FROM   tr_Employee e INNER JOIN
       tr_EmployeeBelongsToDepartment d ON e.EmployeeID = d.EmployeeID
WHERE  d.StartDate < to_date('01/01/2006','mm/dd/yyyy')
UNION
SELECT e.NameLast || ', ' || e.NameFirst || NVL(' ' || e.NameMiddle, '') AS Name
FROM   tr_Employee e INNER JOIN
       tr_EmployeeBelongsToDepartment d ON e.EmployeeID = d.EmployeeID
WHERE  d.StartDate >= to_date('01/01/2010','mm/dd/yyyy');
```

```
NAME
----------------------------------------
Andrew, David
Cartman, Erik
Crawford, Spencer Drew
Elizabeth, Terra Renee
Evans, Khalisa
Gardner, Jennifer Lynn
Goode, Hank Andrew
Lewis, Herbet
Lewis, Jerry Lee
Nancy, Karen Sue
Night, Buzz
Paul, John
Pot, Jack
Ragle, Travis Ray
Ray, Anne
Ray, Chelsy Bee
```

```
Reck, Tifa Anya
Rickner, Ryan Donald
Rivers, Lucky
Simpson, Homer Jay
Smitts, Jason Lee
Stone, Matt
Strong, Arnold
Wiggim, Clancy Raymond
Williams, Carl
Williams, Lenny

26 rows selected.
```

Select all employees and their current projects (games)

```
SELECT e.NameLast
       ,e.NameFirst
       ,a.Title
FROM   tr_Employee e LEFT OUTER JOIN
       (
       SELECT UNIQUE w.EmployeeID
              ,v.Title
       FROM   tr_WorksOn w INNER JOIN
              tr_Videogame v ON w.VideogameID = v.VideogameID
       WHERE  w.EndDate IS NULL) a ON a.EmployeeID = e.EmployeeID
ORDER BY e.NameLast, e.NameFirst, a.Title;
```

```
NAMELAST             NAMEFIRST    TITLE
-------------------- ------------ ------------------
Andrew               David
Angel                Jeff
Black                Wade
Broflowski           Kyle
Cartman              Erik
Crawford             Spencer
Cribbel              Dale
Cribbel              Nacy
Elizabeth            Terra
Evans                Khalisa
Far                  Lisa
Gardner              Jennifer
Goode                Hank
Hill                 Hank
Hill                 Peggy
Kenny                Kenny
Lewis                Herbet
Lewis                Jerry
Love                 Raymond
Mark                 Roger
Marsh                Stan
Murphy               Brittany
Nancy                Karen
Niece                Shannon
Night                Buzz
Paul                 John
Pot                  Jack
Ragle                Travis
Ray                  Anne         Autumn Knight II
Ray                  Chelsy
Reck                 Tifa
Rickner              Ryan
Rivers               Lucky
Simpson              Bart
Simpson              Homer
Simpson              Lisa
Simpson              Marge
```

```
NAMELAST             NAMEFIRST    TITLE
-------------------- ------------ ------------------
Smith                Naomi
Smithers             Wayland
Smitts               Jason
Stone                Matt
Strife               Cloud
Strong               Arnold
Wiggim               Clancy
Williams             Carl
Williams             Lenny

46 rows selected.
```

Select oldest game

```
SELECT v.Title
FROM   tr_Videogame v
WHERE  v.ReleaseDate IS NOT NULL AND
       NOT EXISTS (
       SELECT *
       FROM   tr_Videogame v2
       WHERE  v2.ReleaseDate < v.ReleaseDate);
```

```
TITLE
------------------
Lizard Quest
```

Select all employees that have worked in a subordinate position

```
SELECT UNIQUE NameLast || ', ' || NameFirst || NVL(' ' || NameMiddle, '') AS Name
FROM   tr_Employee e INNER JOIN
       tr_WorksOn w ON w.EmployeeID = e.EmployeeID INNER JOIN
       tr_Position p ON w.PositionID = p.PositionID
WHERE  p.ParentID IS NOT NULL
ORDER BY 1;
```

```
NAME
----------------------------------------
Crawford, Spencer Drew
Elizabeth, Terra Renee
Gardner, Jennifer Lynn
Lewis, Jerry Lee
Nancy, Karen Sue
Ragle, Travis Ray
Ray, Chelsy Bee
Rickner, Ryan Donald
Simpson, Homer Jay
Wiggim, Clancy Raymond

10 rows selected.
```

Select all employees that have worked on at least 3 video games

```
-- Select all employees that have worked on at least 3 video games
SELECT e.NameLast
       ,e.NameFirst
       ,n.NumberOfGames
FROM   (
       SELECT EmployeeID
              ,COUNT(DISTINCT VideogameID) AS NumberOfGames
       FROM   tr_WorksOn
       GROUP BY EmployeeID
       HAVING COUNT(DISTINCT VideogameID) >= 3) n INNER JOIN
       tr_Employee e ON e.EmployeeID = n.EmployeeID
ORDER BY e.NameLast, e.NameFirst;
```

```
NAMELAST             NAMEFIRST    NUMBEROFGAMES
-------------------- ------------ -------------
Gardner              Jennifer                7
Lewis                Jerry                   5
Nancy                Karen                   3
Simpson              Homer                   4
Wiggim               Clancy                  4
```

Create a new table for families that have a family member that has worked on a video game

```
-- Create a new table of each family that has a member that has worked on
-- a video game
CREATE TABLE tr_Family
AS
SELECT UNIQUE e.NameLast AS FamilyName
FROM   tr_Employee e INNER JOIN
       tr_WorksOn w ON e.EmployeeID = w.EmployeeID
ORDER BY 1;

SELECT *
FROM   tr_Family;

DROP TABLE tr_Family PURGE;
```

```
Table created.


FAMILYNAME
----------------------------------------
Crawford
Elizabeth
Gardner
Lewis
Nancy
Ragle
Ray
Rickner
Simpson
Wiggim

10 rows selected.


Table dropped.
```

DATA LOADER

METHODS

There are a number of methods that can be used to load data into tables in a RDBMS. Records can be inserted one at a time using an INSERT INTO … VALUES (…) statement, while multiple records can be inserted using an INSERT INTO … *query* statement. Additionally, some commercial and custom tools can be used to quickly load data.

Some tools can be used directly or indirectly to create a script that can be executed at a later date. For example, some spreadsheet applications allow formulas to be created that can create multiple INSERT INTO … VALUES (…) statements by concatenating values from other columns. Some database management systems, such as Microsoft SQL Server, contain tools that will attempt to load data directly from a file (such as CSV or tab-delimited) into a table. Custom tools can also be created in such languages as Java. Also, some similar tools will allow data to be extracted from a database and written to one or more files in a different format.

JAVA DATALOADER

Dr. Huaqing Wang provided the 2010 CS342 class at California State University, Bakersfield with a custom data loader program written in Java. This program will read data from a text file and load it into one or more tables in a given database, as specified in the file. The program will allow the user to select the delimiting character (such as "|"). One possible format for the import file is as follows:

TABLENAME | *tableName* | *numberOfColumns*
row1col1value | *row1col2value* | … | *row1colNvalue*
row2col1value | *row2col2value* | … | *row2colNvalue*
...
rowNcol1value | *rowNcol2value* | … | *rowNcolNvalue*

All data in the video game developer database was loaded using this tool. However, one very small modification was made to the tool. A NULL value can be inserted by hard coding the word "NULL" where the value should be. The program will insert a NULL of the correct data type into the table. For example, the following import file will load 2 rows of data into MyTable, and in each row the second column is NULL.

TABLENAME | MyTable| 2
1 | NULL
2 | NULL

PHASE IV

COMMON FEATURES IN ORACLE PL/SQL AND MS TRANS-SQL

Oracle PL/SQL and Microsoft Transact SQL (or T-SQL) has many differences. Some differences are as minor as syntax changes, such as string concatenation, while others are much more significant. A small subset of some of the differences of each language is discussed below.

T-SQL generally allows flexibility in how various statements can be used. For example, infinite recursive queries may be created to aid in such tasks as row generation. Select statements can be used in many places without the need for intermediate variables and statements (for example, a select statement can be part of an "if" condition). Local temporary tables allow intermediate result sets to be easily stored and manipulated. T-SQL allows select statements that do not contain a "from" clause, removing the need for a DUAL table. Though not directly part of T-SQL, Microsoft provides many tools to aid in robust management and development, such as SQL Server Integration Services and LINQ to Entities.

PL-SQL generally allows flexibility in what tools are provided to the programmer as a basic part of the DBMS. For example, packages can be used to improve performance and overload stored procedures. Row variables and column %TYPE variables can be used to reduce future maintenance. Exceptions can be easily managed in a single exception area of a given code block. Sequences are included in Oracle, which allow multiple tables to have one common unique identifier without the need for GUIDs. Nested tables are also included in Oracle, allowing for more flexibility in design.

Though there are many differences between T-SQL and PL/SQL, there are also many similarities. For example, each language is used to create stored subprograms, or named code blocks. Stored subprograms are pre-compiled, so there is generally a performance improvement over sending dynamic SQL from a front-end to the DBMS. Additionally, several statements may be contained within a single stored subprogram, and a subprogram can be executed over the network through a single call. Without using sub programs, each statement would need to be sent as a separate call over the network, increasing traffic and decreasing performance.

In addition to performance and network benefits, stored subprograms can also make a database much more manageable. For example, complex aggregate logic can be included in a single stored function and then called from any query, which allows code to be much more readable. Triggers allow code to be executed immediately and automatically upon certain database events. Business logic for such tasks as insertions can be included in a stored procedure, which allows the business logic to be managed at a single point rather than through multiple front end applications. Additionally, stored procedures can be used to limit the actions that a user can perform against a given database, as well as the amount and type of information that is viewable. Stored procedures may also aid in automating certain database tasks.

ORACLE PL/SQL

PL/SQL consists of code blocks. Each code block contains three basic parts, though not every part is mandatory. Variables, constants, subtypes, cursors and user-defined exceptions are declared in the declaration part. The executable part consists of PL/SQL statements that provide the main functionality of the code block. The exception handling part can be used to execute certain code when a pre-defined or user-defined warning or error is encountered. The basic syntax for a code block is given below.

```
DECLARE
        Declarations
BEGIN
        Statements
EXCEPTION
        WHEN Exception1 THEN Statements
        WHEN Exception2 THEN Statements
        …
END;
```

Cursors, which are declared in the declaration section, allow for data to be manipulated one row at a time. Oracle will create cursors implicitly, but they may be declared explicitly as well for multiple rows. The basic syntax for a cursor appears below.

```
CURSOR cursorName (parameter list)
IS selectStatement;
```

Several control structures are allowed in the code block, such as conditional controls, iterative controls and sequential controls. Both “while” and “for” loops are allowed, as well as generic loops. An example conditional control and iterative control appears below, though many other possibilities exist.

```
IF Condition THEN
        Statements
ELSEIF Condition THEN
        Statements
ELSE
        Statements
END IF;
```

```
FOR Counter IN LowerBound .. UpperBound LOOP
        Statements
END LOOP;
```

Two types of subprograms, or named code blocks that can be called once defined, are stored procedures and functions. A stored procedure is defined in the same way as a generic code block, with a few changes. The reserved word “DECLARE” is replaced with the procedure specification (with an optional parameter list). Each parameter can be declared as “IN” (input only), “OUT” (output only) or “IN OUT” (both input and output). Also, each parameter can be declared with an optional default value. A stored procedure may also return a value. The basic syntax to declare a procedure is below.

CREATE OR REPLACE PROCEDURE *procedureName* (*paramaterList)* AS
    Declarations
BEGIN
    Statements
EXCEPTION
    WHEN *Exception1* THEN *Statements*
    WHEN *Exception2* THEN *Statements*
    …
END;

A function is defined in the same way as a stored procedure, with one major difference. A return type must be specified, because the single return value is not optional. The basic syntax to declare a function is below.

CREATE OR REPLACE FUNCTION *functionName(parameterList)*
RETURN *returnType*
    Declarations
BEGIN
    Statements
EXCEPTION
    WHEN *Exception1* THEN *Statements*
    WHEN *Exception2* THEN *Statements*
    …
END;

In Oracle, a package is a collection of certain related schema objects, such as stored procedures and functions. The package has both a package specification part and package body part. The package declaration part is used to declare public constants, variables, types, exceptions and cursors as well as prototypes for public stored procedures and functions. The package body part is used to define the implementation of public stored procedures and functions, as well as declare and implement private constants, variables, types, exceptions, cursors, stored procedures and functions. The basic syntax to declare and define a package is below.

CREATE PACKAGE *packageName* AS
    Public Declarations
END;

CREATE PACKAGE BODY *packageName* AS
    Private Declarations and Implementation
    Implementation of Public Stored Procedures and Functions
END;

A trigger is a code block that is executed automatically when certain events occur. One use of triggers is to execute code before, after, or instead of an insert, update or delete statement against a table. These can be used to automatically log certain events, ensure complex business rules are automatically and always implemented, as well as others tasks. Triggers can optionally be executed for each row. Also, triggers can be executed for only certain columns. The basic syntax to declare a trigger on a table is below, though not all parts are mandatory.

CREATE OR REPLACE TRIGGER *triggerName*
WhenToExecute
OF *Columns*
ON *tableName*
FOR EACH ROW
DECLARE
    Declarations
BEGIN
    Statements
EXCEPTION
    WHEN *Exception1* THEN *Statements*
    WHEN *Exception2* THEN *Statements*
    …
END;

Below are three examples of *WhenToExecute* as indicated above.

BEFORE UPDATE
AFTER INSERT OR DELETE
INSTEAD OF UPDATE

ORACLE PL/SQL SUBPROGRAM

This database will use at least two stored procedures, two functions and a single trigger. The stored procedures are primarily used to aid in database manipulation, specifically inserting and deleting certain records. The functions are used to quickly return aggregate information upon request, specifically the average age of a subset of employees. The triggers are used primarily to aid in maintaining a log of database activity, specifically record updates and deletions.

USP_TR_INSERTEMPLOYEE

The following stored procedure will be used to insert a new employee into the database. All strings are trimmed to ensure that searches, orderings and other future tasks will perform as expected. Additionally, this stored procedure generates the EmployeeID by finding the smallest available number that is not already used in the tr_Employee table.

```
CREATE OR REPLACE PROCEDURE usp_tr_InsertEmployee
(
    first_name   IN tr_Employee.NameFirst%TYPE
    ,middle_name IN tr_Employee.NameMiddle%TYPE
    ,last_name   IN tr_Employee.NameLast%TYPE
    ,birth_date  IN tr_Employee.Birthdate%TYPE
    ,user_sex    IN tr_Employee.Sex%TYPE
    ,street_1    IN tr_Employee.AddressStreet1%TYPE
    ,street_2    IN tr_Employee.AddressStreet2%TYPE
    ,city        IN tr_Employee.AddressCity%TYPE
    ,state       IN tr_Employee.AddressState%TYPE
    ,zipcode     IN tr_Employee.AddressZip%TYPE
)
IS
    new_id       tr_Employee.EmployeeID%TYPE;
BEGIN
    -- This table does not use a sequence.  Set new_id to the
    -- smallest available id
    WITH SmallestID AS
    (
        SELECT 1 AS ID FROM DUAL
        UNION ALL
        SELECT ROWNUM + 1 FROM tr_Employee
    )
    SELECT MIN(s.ID)
    INTO   new_id
    FROM   SmallestID s
    WHERE  NOT EXISTS (
        SELECT EmployeeID
        FROM   tr_Employee
        WHERE  EmployeeID = s.ID);

    -- Insert into tr_Employee, trimming all strings
    INSERT INTO tr_Employee(
        EmployeeID
        ,NameFirst
        ,NameMiddle
        ,NameLast
        ,Birthdate
        ,Sex
        ,AddressStreet1
        ,AddressStreet2
        ,AddressCity
        ,AddressState
        ,AddressZip
```

```
    ) VALUES (
        new_id
       ,TRIM(first_name)
       ,TRIM(middle_name)
       ,TRIM(last_name)
       ,birth_date
       ,UPPER(TRIM(user_sex))
       ,TRIM(street_1)
       ,TRIM(street_2)
       ,UPPER(TRIM(city))
       ,TRIM(state)
       ,TRIM(zipcode)
    );

    COMMIT;

    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            DBMS_OUTPUT.PUT_LINE( SQLCODE || ', ' || SQLERRM );
            COMMIT;
END;
```

USP_TR_DELETEEMPLOYEE

This stored procedure will be used to delete an employee from the database. Rather than cascade deletions, this stored procedure will delete any records from tables that contain a foreign key which reference the employee before finally deleting the employee record.

```
CREATE OR REPLACE PROCEDURE usp_tr_DeleteEmployee
(
    delete_id IN tr_Employee.EmployeeID%TYPE
)
IS
BEGIN
    -- Delete from foreign key tables first
    DELETE
    FROM  tr_WorksOn
    WHERE EmployeeID = delete_id;

    DELETE
    FROM  tr_EmployeeBelongsToDepartment
    WHERE EmployeeID = delete_id;

    DELETE
    FROM  tr_Employee
    WHERE EmployeeID = delete_id;

    COMMIT;

    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            DBMS_OUTPUT.PUT_LINE( SQLCODE || ', ' || SQLERRM );
            COMMIT;
END;
```

UF_TR_OLDESTAVGAGE

This function will return the average age for the top N oldest employees. If no parameters are passed in, the oldest employee's age is returned.

```
CREATE OR REPLACE FUNCTION uf_tr_OldestAvgAge
(
    N NUMBER DEFAULT 1
)
RETURN NUMBER
IS
    average_age NUMBER;
BEGIN
    WITH OrderByAge AS
    (
        SELECT FLOOR( MONTHS_BETWEEN(SYSDATE, Birthdate) / 12 ) AS Age
        FROM   tr_Employee
        ORDER BY Birthdate
    )
    SELECT AVG(Age)
    INTO   average_age
    FROM   OrderByAge
    WHERE  ROWNUM <= N;

    RETURN average_age;

    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE( SQLCODE || ', ' || SQLERRM );
END;
```

UF_TR_YOUNGESTAVGAGE

This function will return the average age for the top N youngest employees. If no parameters are passed in, the youngest employee's age is returned.

```
CREATE OR REPLACE FUNCTION uf_tr_YoungestAvgAge
(
    N NUMBER DEFAULT 1
)
RETURN NUMBER
IS
    average_age NUMBER;
BEGIN
    WITH OrderByAge AS
    (
        SELECT FLOOR( MONTHS_BETWEEN(SYSDATE, Birthdate) / 12 ) AS Age
        FROM   tr_Employee
        ORDER BY Birthdate DESC
    )
    SELECT AVG(Age)
    INTO   average_age
    FROM   OrderByAge
    WHERE  ROWNUM <= N;

    RETURN average_age;

    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE( SQLCODE || ', ' || SQLERRM );
END;
```

TRG_TR_SERIES

This trigger will create a string log entry into tr_logTable anytime a Series is updated or deleted.

```
CREATE OR REPLACE TRIGGER trg_tr_Series
BEFORE UPDATE OR DELETE
ON tr_Series
FOR EACH ROW
BEGIN
    -- Insert old and new values into log
    INSERT INTO tr_logTable (
        oldVal
        ,newVal
    ) VALUES (
         :old.SeriesID || ' ' || :old.Name
        ,:new.SeriesID || ' ' || :new.Name
    );

    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            DBMS_OUTPUT.PUT_LINE( SQLCODE || ', ' || SQLERRM );
            COMMIT;
END;
```

PHASE V

DAILY ACTIVITIES OF USER GROUP

Although this database is presented as a generic, flexible foundation that companies may customize and specialize as needed, some general guidelines can be given that will apply to most case scenarios. In a general sense, the purpose of this database is to allow managers to determine the best use of a given staff for a current set of projects. However, other groups may benefit from the information stored in this database. We will present three possible user groups.

An expanded version of this database may aid employees on their current projects. Although a typical employee will likely have little need to edit data in the current implementation, additional facilities could be added to allow an employee to log his or her weekly progress, mark tasks as complete, and/or submit general information for a given project. The information in an expanded version of this database could also contain important information for a given project, such as contact information, goals, deadlines and other important data. An employee could generate a report of this information for a project he or she is assigned.

Although not a direct user group, the information in this database could be used in a distributed environment as a "base" set of data. With such technique as data virtualization, the base information in this database could be a foundation for customer websites and other indirect users. In this way, information on a new videogame could be made readily available to others automatically if desired, or at least with significant reduction in work hours. Additionally, since the data could truly be synchronized to one source, mismatches and erroneous data could be significantly minimized.

However, the primary user group for this database is project managers. Common functionality needed for most specific applications are adding an employee to a current video game project, updating employee and video game information, and recording special dates such as the ending and starting of phases. Managers may also find various reports to be extremely useful, such as the average time spent on a subset of videogames for a specific phase, position, employee, etc. Managers would likely also desire to have information on current projects and current employees, to see which resources are under or over utilized and which projects are likely to need additional help.

For this specific demonstration application, only the last group will be considered.

RELATIONS, VIEWS AND SUBPROGRAMS OF APPLICATION

No additional relations were needed for the demonstration application. However, two views were created to aid in presenting information to the user. Also, seven additional stored procedures were created to aid in various insert, delete and update tasks.

VIEWS

The following views were created for this demonstration application:

1. **v_tr_Videogames** – This view joins information from the tr_Videogame, tr_Genre, tr_ConsoleIsInRegion, tr_Console, tr_Region and tr_Series relations to present a complete set of information for each videogame in the database. This view is used primarily on the Videogame tab in the demonstration application.
2. **v_tr_Projects** – This view joins information from the tr_WorksOn, tr_Employee, tr_Position, tr_Videogame, tr_Genre, tr_ConsoleIsInRegion and tr_Console relations to present a complete set of information for each project in the database. This view is used primarily on the Projects tab.

SUBPROGRAMS

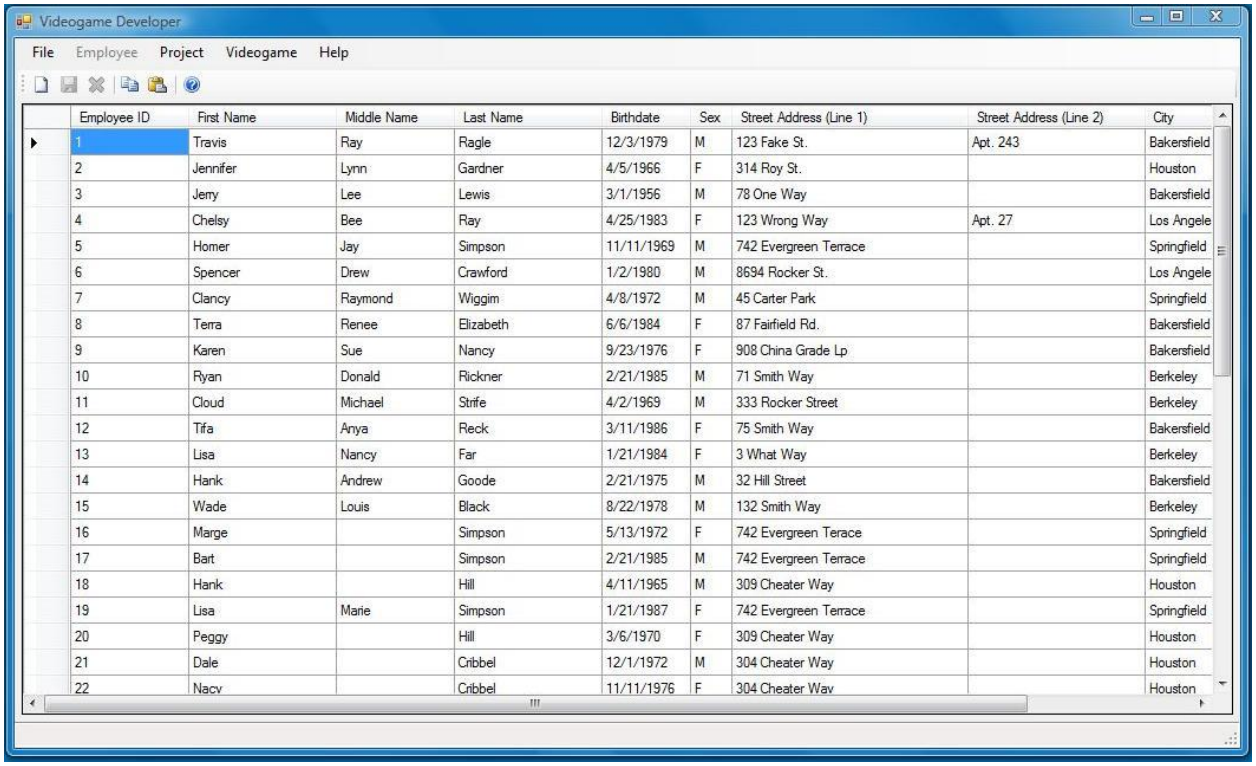
The following stored procedures were created for this demonstration application:

1. **usp_tr_DeleteEmployee**, **usp_tr_DeleteProject** and **usp_tr_DeleteVideogame** – These three stored procedures were created to delete an entry from the Employee, Project and Videogame tab respectively. This approach was chosen to allow any cascade deletions or other special rules to be implemented within the database, hiding the details from any front-end application.
2. **usp_tr_InsertEmployee**, **usp_tr_InsertProject** and **usp_tr_InsertVideogame** – These three stored procedures were created to insert a new entry from the Employee, Project and Videogame tab respectively. Some of these stored procedures will generate new ids for primary key attributes.
3. **usp_tr_UpdateEmployee**, **usp_tr_UpdateProject** and **usp_tr_UpdateVideogame** – These three stored procedures were created to update an Employee, Project and Videogame respectively. Using stored procedures ensures that only certain attributes can be updated, and protects others.

In addition to some of the specialized reasons given above, there are a few general reasons to use subprograms rather than implementing code from a front-end. As mentioned previously in this document, stored subprograms are precompiled and generally offer greater performance than dynamic code from a front-end. Front-end code is often simplified because only the necessary parameters need to be considered, rather than a specialized implementation that may require maintenance if the underlying database or DBMS is ever changed. Also, a simple view can be used in the select command in the front-end without the need to write lengthy update, delete or insert routines on multiple relations.

SCREENSHOTS FROM APPLICATION

EMPLOYEE MAIN FORM



Employee ID	First Name	Middle Name	Last Name	Birthdate	Sex	Street Address (Line 1)	Street Address (Line 2)	City
1	Travis	Ray	Ragle	12/3/1979	M	123 Fake St.	Apt. 243	Bakersfield
2	Jennifer	Lynn	Gardner	4/5/1966	F	314 Roy St.		Houston
3	Jerry	Lee	Lewis	3/1/1956	M	78 One Way		Bakersfield
4	Chelsy	Bee	Ray	4/25/1983	F	123 Wrong Way	Apt. 27	Los Angele
5	Homer	Jay	Simpson	11/11/1969	M	742 Evergreen Terrace		Springfield
6	Spencer	Drew	Crawford	1/2/1980	M	8694 Rocker St.		Los Angele
7	Clancy	Raymond	Wiggim	4/8/1972	M	45 Carter Park		Springfield
8	Terra	Renee	Elizabeth	6/6/1984	F	87 Fairfield Rd.		Bakersfield
9	Karen	Sue	Nancy	9/23/1976	F	908 China Grade Lp		Bakersfield
10	Ryan	Donald	Rickner	2/21/1985	M	71 Smith Way		Berkeley
11	Cloud	Michael	Strife	4/2/1969	M	333 Rocker Street		Berkeley
12	Tifa	Anya	Reck	3/11/1986	F	75 Smith Way		Bakersfield
13	Lisa	Nancy	Far	1/21/1984	F	3 What Way		Berkeley
14	Hank	Andrew	Goode	2/21/1975	M	32 Hill Street		Bakersfield
15	Wade	Louis	Black	8/22/1978	M	132 Smith Way		Berkeley
16	Marge		Simpson	5/13/1972	F	742 Evergreen Terace		Springfield
17	Bart		Simpson	2/21/1985	M	742 Evergreen Terrace		Springfield
18	Hank		Hill	4/11/1965	M	309 Cheater Way		Houston
19	Lisa	Marie	Simpson	1/21/1987	F	742 Evergreen Terrace		Springfield
20	Peggy		Hill	3/6/1970	F	309 Cheater Way		Houston
21	Dale		Cribbel	12/1/1972	M	304 Cheater Way		Houston
22	Nacy		Cribbel	11/11/1976	F	304 Cheater Way		Houston

The employee main form contains several features that are also found in the Project and Videogame tabs. The data contained in the grid is directly from the tr_Employee relation. All data can be directly deleted, updated and inserted within the grid itself, similar to Microsoft Excel. The main functionality is found within the tool strip.

New

The new button will simply scroll the data grid to the last row, since all insertions can be made directly on the grid.

Save

The save button will only be enabled if changes exist on the grid. This button will update the dataset with all insertions, updates and deletions made on the grid by using the usp_tr_InsertEmployee, usp_tr_UpdateEmployee and usp_tr_DeleteEmployee stored procedures, respectively. If the EmployeeID is not entered during an insertion, a new one is automatically created. After all changes are applied, the data grid is updated to reflect the current state of the database.

Cancel

The cancel button will cancel any changes that exist on the grid that have not been saved. The grid will automatically revert to the state of the last load or save.

 **Copy**

The copy button or CTRL-C will copy the current selection to the clipboard, similar to pressing CTRL-C in Microsoft Excel. The selection can then be paste into Notepad, Excel or other compatible programs. This allows for data to be easily manipulated outside of the demonstration application.

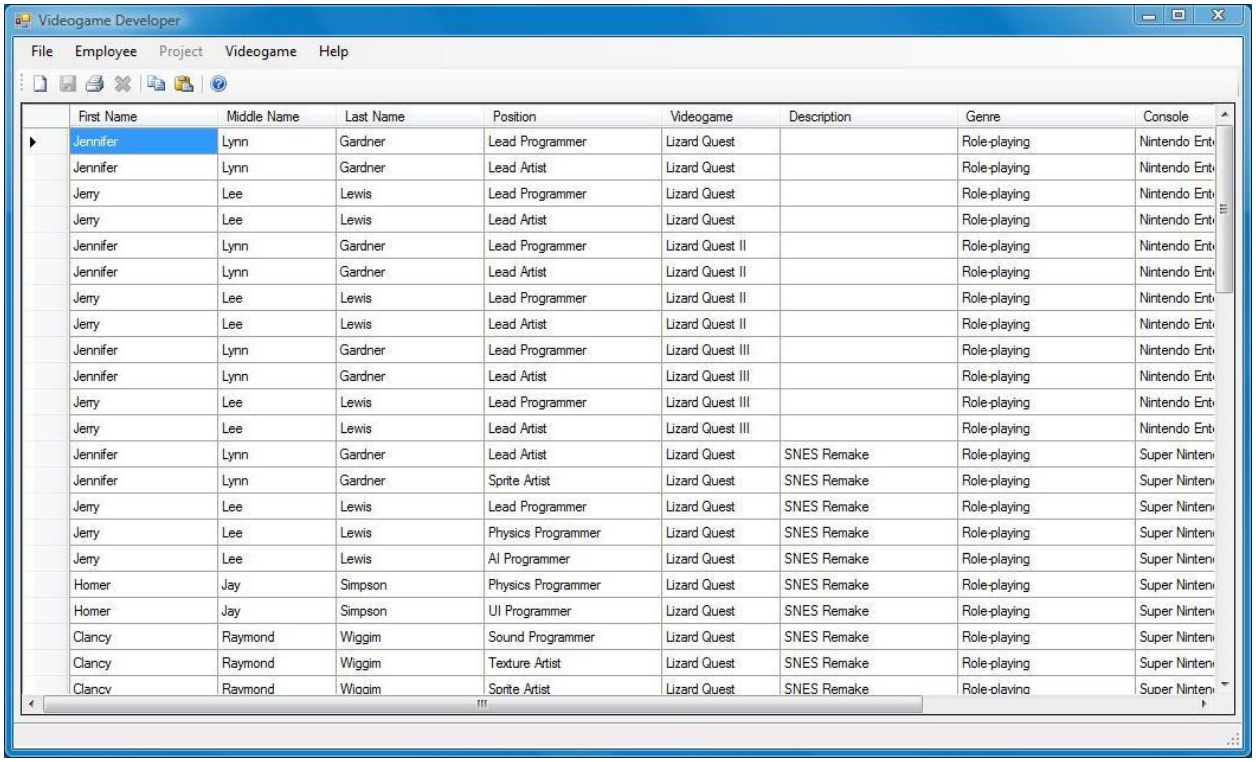
 **Paste**

The paste button or CTRL-P will paste the contents of the clipboard onto the data grid, beginning with the currently selected cell as the top-left cell. This allows a significant amount of data to be cut from a spreadsheet application, paste into the demonstration application and then saved into the database.

 **Help**

This button will show a simply pop-up which states that all edits, additions and deletions can be made directly on the grid.

PROJECT MAIN FORM

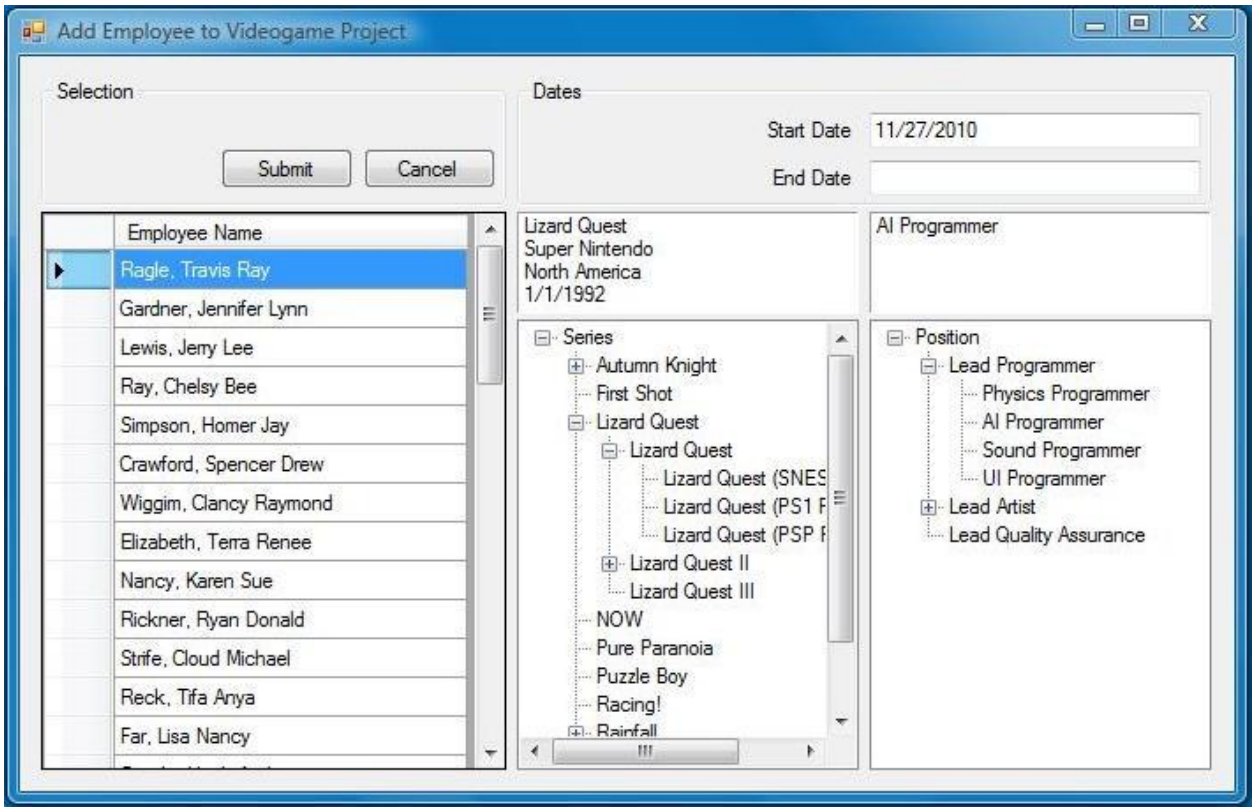


First Name	Middle Name	Last Name	Position	Videogame	Description	Genre	Console
Jennifer	Lynn	Gardner	Lead Programmer	Lizard Quest		Role-playing	Nintendo Ent
Jennifer	Lynn	Gardner	Lead Artist	Lizard Quest		Role-playing	Nintendo Ent
Jerry	Lee	Lewis	Lead Programmer	Lizard Quest		Role-playing	Nintendo Ent
Jerry	Lee	Lewis	Lead Artist	Lizard Quest		Role-playing	Nintendo Ent
Jennifer	Lynn	Gardner	Lead Programmer	Lizard Quest II		Role-playing	Nintendo Ent
Jennifer	Lynn	Gardner	Lead Artist	Lizard Quest II		Role-playing	Nintendo Ent
Jerry	Lee	Lewis	Lead Programmer	Lizard Quest II		Role-playing	Nintendo Ent
Jerry	Lee	Lewis	Lead Artist	Lizard Quest II		Role-playing	Nintendo Ent
Jennifer	Lynn	Gardner	Lead Programmer	Lizard Quest III		Role-playing	Nintendo Ent
Jennifer	Lynn	Gardner	Lead Artist	Lizard Quest III		Role-playing	Nintendo Ent
Jerry	Lee	Lewis	Lead Programmer	Lizard Quest III		Role-playing	Nintendo Ent
Jerry	Lee	Lewis	Lead Artist	Lizard Quest III		Role-playing	Nintendo Ent
Jennifer	Lynn	Gardner	Lead Artist	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Jennifer	Lynn	Gardner	Sprite Artist	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Jerry	Lee	Lewis	Lead Programmer	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Jerry	Lee	Lewis	Physics Programmer	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Jerry	Lee	Lewis	AI Programmer	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Homer	Jay	Simpson	Physics Programmer	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Homer	Jay	Simpson	UI Programmer	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Clancy	Raymond	Wiggim	Sound Programmer	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Clancy	Raymond	Wiggim	Texture Artist	Lizard Quest	SNES Remake	Role-playing	Super Nintend
Clancy	Raymond	Wiggim	Sprite Artist	Lizard Quest	SNES Remake	Role-playing	Super Nintend

The project main form displays the information in the v_tr_Projects view. This form operates in the same way as the Employee main form with a few exceptions. Although the copy and paste buttons operate in a similar fashion, all columns in the grid are read only except for the End Date column (part of the tr_WorksOn relation). Though any row can be deleted, only the End Date column can be updated. Additionally, insertions cannot be made directly on the grid. Other exceptions are as follows.

New

This button will pop up the form below, which will add a new entry to tr_WorksOn through the usp_tr_InsertProject stored procedure.



The controls on this form, from left to right and then top to bottom, are as follows:

1. **Submit** – this button will attempt to save a new project to the database. If unsuccessful, the user will be notified of any errors. If successful, the pop up will close and the data grid on the main form will automatically be updated to reflect the current database state.
2. **Cancel** – this button will close the pop up without saving any changes.
3. **Start Date** – This is the date that a given employee started on a given position for a given video game (stored in the tr_WorksOn relation).
4. **End Date** – This is the date that a given employee finished his or her work on a given position for a given video game (stored in the tr_WorksOn relation).
5. **Employee Name** – This data grid view presents the first, middle and last names from the tr_Employee relation in a single column, which is used to select the Employee ID for the tr_WorksOn relation.
6. **(Series/Videogames)** – The tree view (bottom) presents a hierarchy which starts with the hardcoded string “Series” as the top level parent, followed by the series found in tr_Series, followed by the hierarchy found in the recursive relation tr_Videogame. The label (top) displays the following information for the selected videogame: Title (tr_Videogame), Console Name (tr_Console), Location (tr_Region) and Release Date (tr_Videogame). This is used to select the VideogameID for the tr_WorksOn relation.

7. **(Position)** – The tree view (bottom) present a hierarchy with the hardcoded string “Position” as the top level parent, followed by the recursive relation tr_Position. The label (top) displays the selected position. This is used to select the PositionID for the tr_WorksOn relation.

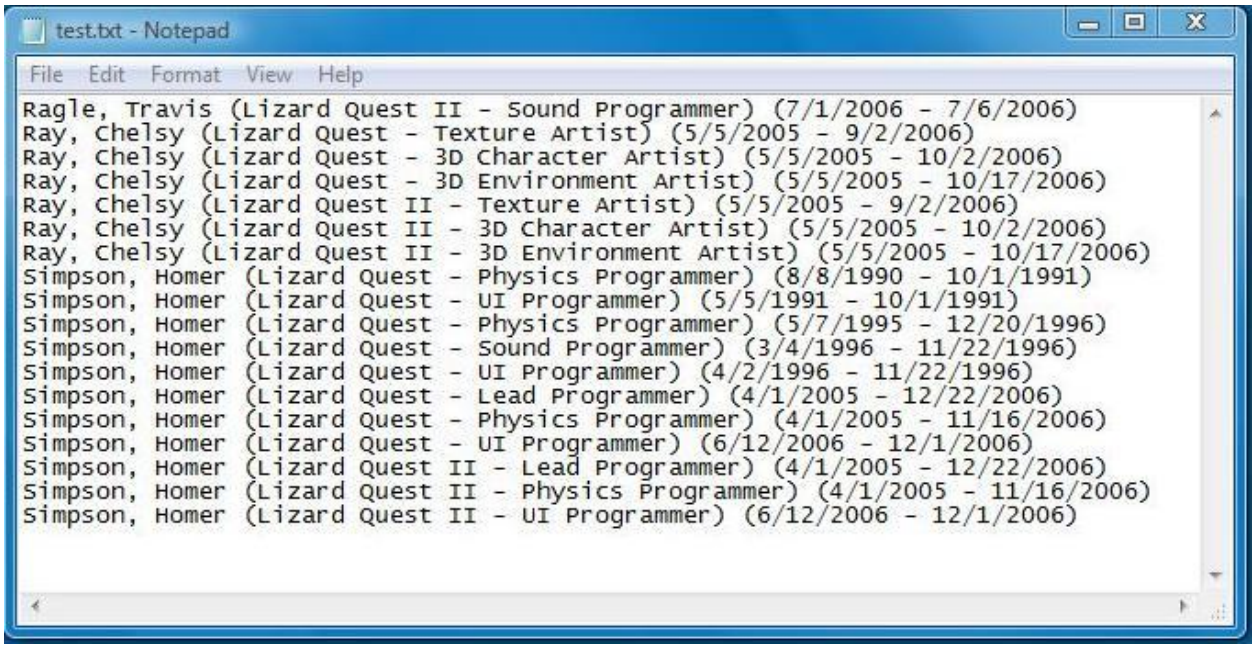
Print

This button will pop up the form below, which can be used to generate a simple text file based upon the v_tr_Projects view.

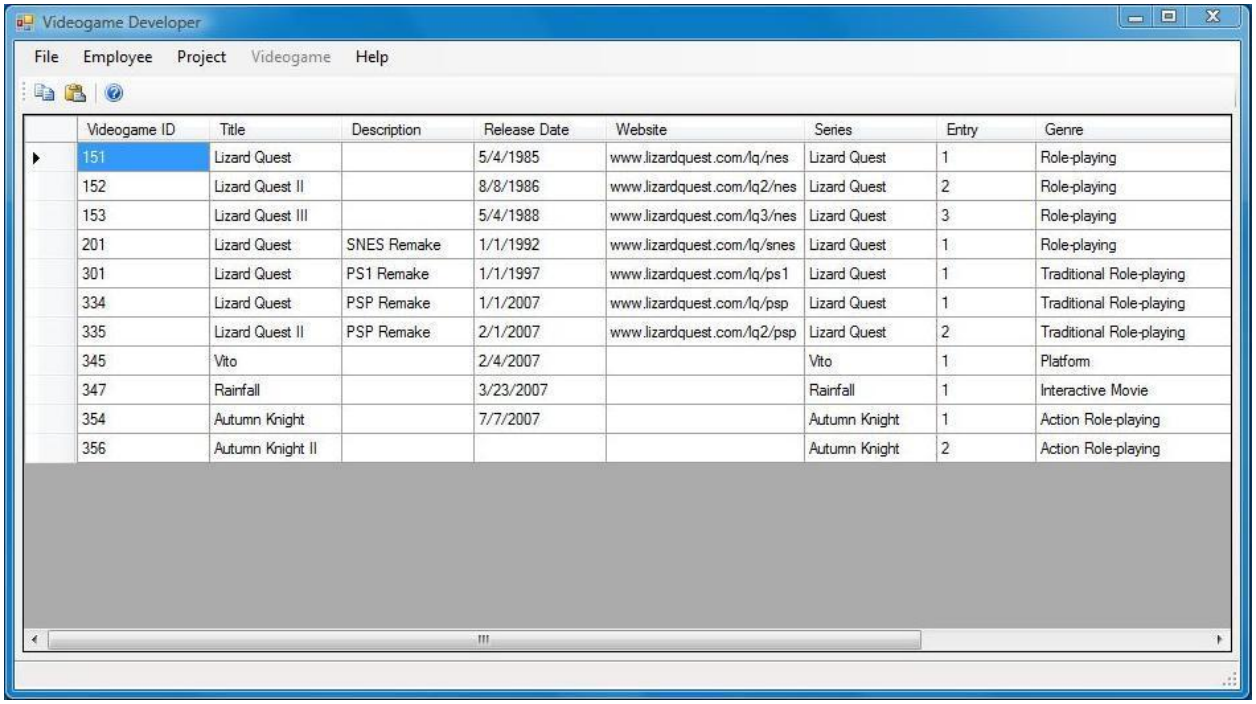
The controls on this form, from left to right and then top to bottom, are as follows:

1. **Select File** – This is the file path for the generated text file.
2. **Select Year** – This combo box is populated with selections for “All” years, the “Current” year and every year from the earliest start date found in tr_WorksOn to the current year. Selecting “Current” will display only current projects, while selecting a year will select all project entries that start on or before that year and ended on or after that year.
3. **Employee Name** – This data grid is populated with a single calculated column based on the first, middle and last names in tr_Employees, and is used to select one or more employees.
4. **Generate Report** – This button will generate the text file and automatically open it in Notepad, if found.
5. **Close** – This button will close the pop up.

An example report is below.



VIDEOGAME MAIN FORM



Videogame ID	Title	Description	Release Date	Website	Series	Entry	Genre
151	Lizard Quest		5/4/1985	www.lizardquest.com/lq/nes	Lizard Quest	1	Role-playing
152	Lizard Quest II		8/8/1986	www.lizardquest.com/lq2/nes	Lizard Quest	2	Role-playing
153	Lizard Quest III		5/4/1988	www.lizardquest.com/lq3/nes	Lizard Quest	3	Role-playing
201	Lizard Quest	SNES Remake	1/1/1992	www.lizardquest.com/lq/snes	Lizard Quest	1	Role-playing
301	Lizard Quest	PS1 Remake	1/1/1997	www.lizardquest.com/lq/ps1	Lizard Quest	1	Traditional Role-playing
334	Lizard Quest	PSP Remake	1/1/2007	www.lizardquest.com/lq/psp	Lizard Quest	1	Traditional Role-playing
335	Lizard Quest II	PSP Remake	2/1/2007	www.lizardquest.com/lq2/psp	Lizard Quest	2	Traditional Role-playing
345	Vito		2/4/2007		Vito	1	Platform
347	Rainfall		3/23/2007		Rainfall	1	Interactive Movie
354	Autumn Knight		7/7/2007		Autumn Knight	1	Action Role-playing
356	Autumn Knight II				Autumn Knight	2	Action Role-playing

The Videogame main form displays the information in the v_tr_Videogames view. This form operates in the same way as the Employee main form with a few exceptions. Although the copy and paste buttons operate in a similar fashion, all columns in the grid are read only. In fact, videogames cannot be added, deleted or added through the demonstration application. At present this is a DBA activity. However, the stored procedures for all three tasks were created, so this functionality can be added in the future.

DESCRIPTION OF CODE

DESIGNING AN INTERFACE

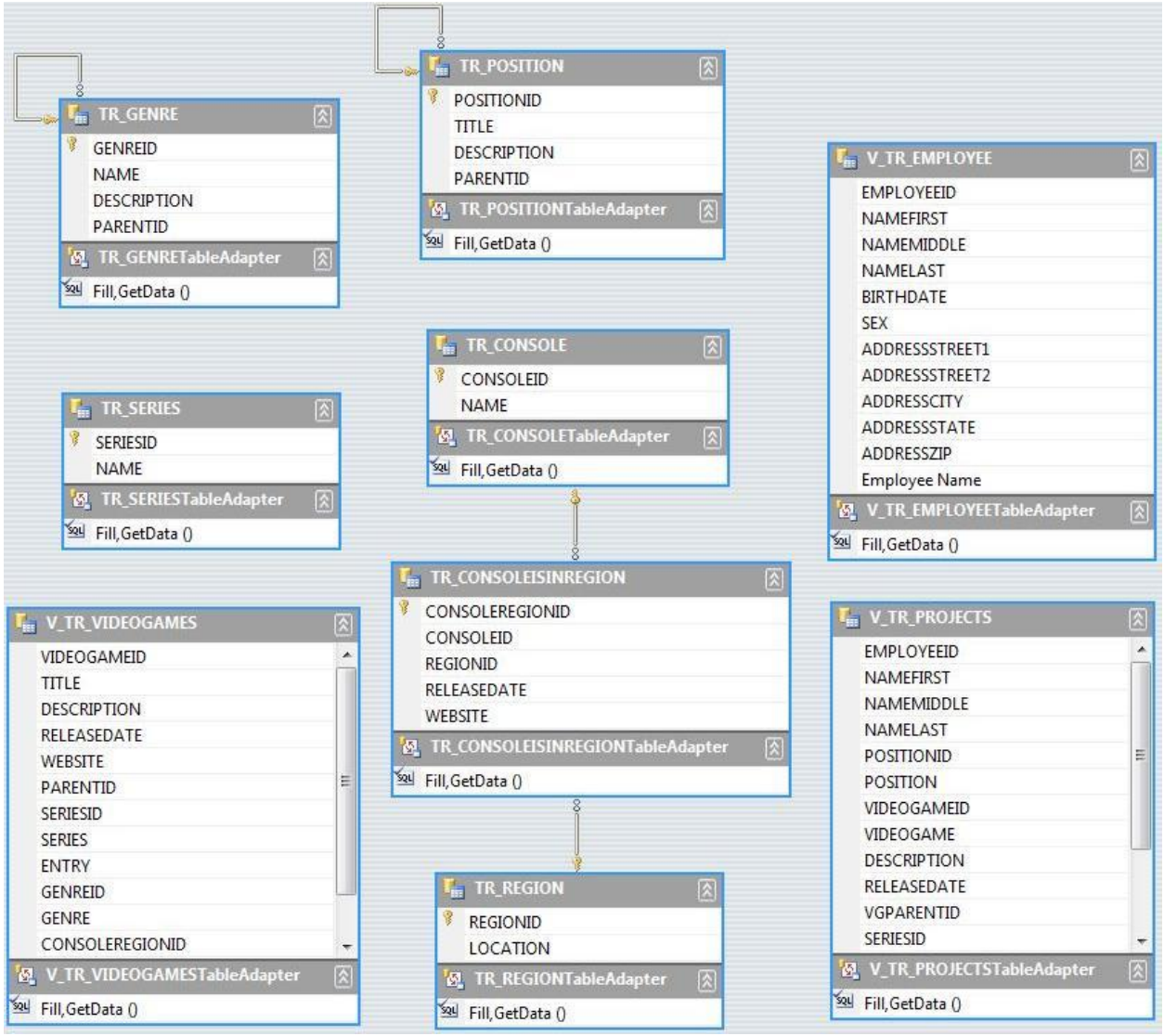
The first step and possibly most important step in interface design is adopting the user role. Often an application developer is very familiar with the underlying data and functionality, so certain things that may appear as obvious to him or her are not obvious to others. Additionally, it may be easy for an application developer to take for granted certain tasks, since he or she often only uses the application during development. It is very important to consider that some users may use this application on a daily basis, and some users may or may not possess great technical skills.

For these reasons, careful consideration should be given to make a UI intuitive, friendly and efficient. For example, before coding begins a general UI design can be written on a piece of paper or quickly created with a graphical tool and presented to others for feedback. Feedback from users that rarely if ever use a computer might be even more useful than feedback from fellow programmers.

Another useful technique during the initial design is to perform each task a number of times in a row. If the task appears to feel overly complex or inefficient, consider a redesign. Perhaps allowing a user to perform the same task in multiple ways can be beneficial, however cluttering the screen with every possible variant can often overcomplicate a design.

MAJOR CLASSES

Ideally, an Entity Data Model for this database would have been created to fully utilize the power of LINQ to Entities. However, at the time of this writing only third parties support using LINQ to Entities with an Oracle database. As an alternative, a single dataset named dsVideogameDB and consisting of multiple data tables and table adapters was utilized, with LINQ code written against the data in a few special cases. The dataset appears below.



Unless otherwise listed below, the data tables and relationships were generated by Microsoft Visual Studio based upon the relations, primary key and foreign key constraints that exist inside the database. In these cases the insert, delete and update commands were ignored because they were not used. Instead, LINQ was used to query information from these data tables to fill combo boxes, tree views and other objects as listed earlier, or in some cases the data was bound to the control. However, three data tables and their associated table adapters require special attention.

1. V_TR_VIDEOGAMES (used with Videogame part of application to fill data grid)
    a. Select command – selects all records and attributes from v_tr_Videogames
    b. Update command – usp_tr_UpdateVideogame
    c. Delete command – usp_tr_DeleteVideogame
    d. Insert command – usp_tr_InsertVideogame
2. V_TR_EMPLOYEES (used with Employee part of application to fill data grid and manipulate data)
    a. Select command – select all records and attributes from v_tr_Employee, plus the calculated column "Employee Name" (Last Name, First Name & Middle Name)
    b. Update command – usp_tr_UpdateEmployee
    c. Delete command – usp_tr_DeleteEmployee
    d. Insert command – usp_tr_InsertEmployee
3. V_TR_PROJECTS (used with Project part of application to fill data grid and manipulate data)
    a. Select command – selects all records and attributes from v_tr_Projects
    b. Update command – usp_tr_UpdateProject
    c. Delete command – usp_tr_DeleteProject
    d. Insert command – usp_tr_InsertProject

Also, an example portion of LINQ is presented below. The first part of this code occurs during the load event for the add project pop up to populate the series / videogames tree view. The recursive function is of course defined outside the load event, but called from within the load event.

```
// ..... previous on load event code above

// Fill Series Treeview (series first)
foreach (dsVideogameDB.TR_SERIESRow s in this.dsVideogameDB.TR_SERIES.OrderBy(o => o.NAME))
{
    TreeNode seriesNode = new TreeNode(s.NAME.ToString());
    seriesNode.Tag = (int)s.SERIESID;

    // Add all games of series
    foreach (dsVideogameDB.V_TR_VIDEOGAMESRow vgRow in
        this.dsVideogameDB.V_TR_VIDEOGAMES.Where(v => v.IsPARENTIDNull() == true
        && v.SERIESID == s.SERIESID))
        addGameNode(vgRow, seriesNode);

    treeViewVideogame.Nodes["Series"].Nodes.Add(seriesNode);
}

// ..... continued on load event code below
```

```
// Recursive function for recursive relation
        private void addGameNode(dsVideogameDB.V_TR_VIDEOGAMESRow vgRow, TreeNode tn)
        {
            TreeNode currentTree = new TreeNode(vgRow.TITLE.ToString()
                + ((vgRow.IsDESCRIPTIONNull() == false)
                ? " (" + vgRow.DESCRIPTION.ToString() + ")" : ""));
            currentTree.Tag = vgRow;

            foreach (dsVideogameDB.V_TR_VIDEOGAMESRow v in
                this.dsVideogameDB.V_TR_VIDEOGAMES
                .OrderBy(o => o.VIDEOGAMEID)
                .Where(o => o.IsPARENTIDNull() == false && o.PARENTID == vgRow.VIDEOGAMEID))
                addGameNode(v, currentTree);

            tn.Nodes.Add(currentTree);
        }
```

MAJOR FEATURES OF GUI

Some major features should be considered in any GUI that provides a front end for database access. In a very general sense, perhaps the most important feature of a GUI is the ability to combine several isolated relations into a single set of data. This allows users to view the entire picture when analyzing data, and in some cases allows users to manipulate one set of complete data that is actually several relations in the database. For example, the Projects part of this application uses a database view to give users a complete, human readable presentation of the data on a data grid. Additionally, users can use friendly tree views and an employee name list when adding a new entry to tr_WorksOn, while the actual relation in the database relies on numerical id columns.

Another very important feature of a GUI is the ability to report refined subsets of pertinent data through friendly controls, rather than viewing the entire set of data for all tasks. This demonstration application allows the user to generate a text project report on a subset of employees and a refined time frame. However, further refinements would be useful in a future version of this application, such as the ability to refine results by position or genre.

One final major feature of a GUI to be considered is the ability to view a given set of data from multiple perspectives. In this demonstration application, the user can sort each data grid by any column in ascending or descending order. This indirectly allows users to group projects by series, or by employee, or by position or by a number of other methods. A future version of this application could generate different reports for different needs. For example, while considering only the information available in the project part of this application, a report could be created to show the work experience for a given employee. Another report could show how many employees are assigned to a given game. Yet another report could show how many employees on staff have experience with a given position. While all of this information is available in the data grid already, customized reports could significantly reduce the work needed by the user to find the information he or she needs.

LEARNING A NEW DEVELOPMENT TOOL AND WRITING CODE IN A NEW LANGUAGE

Learning a new development tool and writing code in a new language are two common challenges a computer scientist must face multiple times in his or her career. In both cases, there are some general guidelines. Additionally, some very specific guidelines were followed during the course of this project.

From a development tool perspective, learning Microsoft Visual Studio proved to be much more challenging than initially anticipated. Developing in a GUI environment based heavily on configuring existing objects is quite a bit different than writing pages of code in a simple text editor. To make matters much worse, a lot of individuals in the computer science community believe that dragging and dropping objects onto a form is the "easy" part of creating an application. While this may be true to a certain extent, it can often be challenging to find out exactly which object to use, exactly which properties and events to configure and how to work around any inherent limitations. There are quite a bit of details that need to be learned before best approaches can be decided upon. Additionally, since Visual Studio generates a lot of code for the programmer, cutting and pasting code can sometimes prove to be disastrous if something becomes corrupted and a recent backup does not exist.

For this project, both MSDN and LearnVisualStudio.net proved to be extremely helpful. Additionally, tutorials and examples given in class provided a strong basis to build upon. However, a very thorough study of Visual Studio and a fair amount of experience would be needed to fully realize all that it has to offer.

Learning to write code in C# proved to be much less difficult. The foundation that is provided by university level computer science courses helps prepare programmers to understand underlying, foundational concepts. Syntax details are generally easy to pick up with such a foundation, and even easier when considering that C# shares a lot in common with C++. Additionally, devices such as interfaces and delegates are not too difficult to understand with a proper background. Even relatively new techniques such as programming with LINQ can be readily understood with such a foundation and some background in declarative programming, such as SQL. Though MSDN and LearnVisualStudio.net were again very helpful in this area, university level computer science courses provide the most preparation to quickly learn C# and other languages as the need arises.

DESIGN AND IMPLEMENTATION OF A DATABASE APPLICATION

Like many other activities in life, by far the most important step in designing and implementing a database application is the initial design phase. Although flexibility is often required in the work world, the amount of time and effort spent in design will often pay dividends for years to come. A carefully planned conceptual model with a thorough requirements document will prevent rework and often reduce programming requirements. Without careful planning, objects such as tables and stored procedures might require many changes which unknowingly affect other portions of the application. Sometimes these last minute changes remain undocumented, which could cause future problems. Sometimes unnecessarily long and complex source code is written in an attempt to fix poor design. With a thorough, well thought design, much of the implementation and code work becomes fairly straight forward. This is also the only step that is not specific to any particular implementation or DBMS. Prematurely making these types of design decisions can cloud proper judgment.

On a personal note, I have worked with Microsoft SQL Server for a few years. However, I have never created a conceptual model prior to this project. I have personally seen many examples of the hours of rework and numerous compromises that could have been avoided with a proper design.

Another important step is the implementation of the conceptual model and requirements document. Although it is very difficult to fix a poor design, it is fairly easy to ruin a good design. Proper thought should be spent particularly with a relational implementation, because quite often the conceptual model will have to be approximated. A technique that works well with one conceptual model may overly complicate the implementation in another case, while the technique that works well in the second case may create an unreasonably high amount of nulls in the first scenario. Simply relying on a single conversion technique or a limited set of conversion techniques may simplify the knowledge necessary to work with database applications, but at the cost of poor and inefficient implementations.

I expect the methods described in this document will aid anyone that takes the time to learn them. With a very clear view of what is possible, more time and thought can be spent on which methods have the greatest benefits in terms of efficiency, future maintenance and other areas. Rather than just implementing what works, careful thought can be spent on what works *best*.

After an implementation is decided upon, the next step is to create it, which can be divided into two sub-steps. In the first sub-step, scripts can be written to create the relations and constraints that have been decided upon in previous steps. Additionally, predefined data can be loaded into the relations. An important aspect of this step is to properly store the created scripts, because with them the database may be rebuilt very quickly.

Sometime just before or during the next sub-step, the capabilities of the specific DBMS need to be taken into account. Although many of the first SQL creation scripts are standard and straight forward, more design decisions are necessary especially when considering stored subprograms. Schema objects that are very powerful in one DBMS may not even exist in another. Approaches that are learnt in one DBMS may help a developer discover a hidden opportunity in another.

On a personal note, working with Oracle has certainly expanded my knowledge of RDBMS and SQL. At times I find it a little frustrating that something that can be easily accomplished in T-SQL takes more code in PL/SQL. However, that minor frustration pales in comparison to the enthusiasm of the new possibilities present in Oracle. In particular, I am very impressed with the possibilities that nested tables and packages present, the reduced maintenance that column data types and row data types in PL/SQL can provide, and the elegance of the structured exception handling.

Although more steps can exist in larger applications that consider middle layers, such as a business logic layer, the final step considered in this project is a GUI application for the database. Although this is often the final step, ideally the GUI design may be discussed as early as during the conceptual phase. In real world scenarios the application team may consists of different members than the database team, so proper coordination is a necessity. Without proper foresight, last minute database changes or additions may be needed before the application team can proceed with their work. In other cases it isn't always apparent which team should handle a certain tasks. In some cases, certain tasks can be implemented mainly through stored subprograms in a database or through code in a front end, especially when considering LINQ. Without a proper plan and proper guidelines in place, the front end and possibly the database may become difficult to manage due to inconsistency and unnecessary coupling.

On one final personal note, this is my first real experience implementing a complete C# front-end application for a database. I have had some very minor experience with VB.Net windows applications and C# web applications, which provided some help. Just prior to this project I learnt about LINQ to Entities, so I decided to devote a portion of my study time to it. However, at a later date I discovered that no first party support exists for it, so I am only able to use LINQ syntax through datasets and other related objects. Additionally, while datasets provide strong typing and other convenient features, they also generate a lot of additional code when compared to

defining data connections, adapters and other objects by hand. This creates an application that is very difficult to troubleshoot when and if the generated code contains an error.

At the time of this writing, there are arguably no single, correct answers for anything we have considered. What is most important is to learn what is available and continue to learn, and continue to actively seek new knowledge. Only by this process can we hope to progress.