

# **Instructor Assistant Grading System (IAGS)**

database overview and design

Joseph E. Sutton  
November 28, 2006  
California State University, Bakersfield  
CMPS 342

## Table of Contents

<b>Section 1:</b> Fact-Finding Techniques and Information Gathering	<b>1-3</b>
<b>1.1 Description Fact-Finding Techniques</b>	<b>1-2</b>
1.1.1 Examine Documents	1
1.1.2 Interview	1
1.1.3 Research	2
1.1.4 Questioning	2
1.1.5 Observations	2
<b>1.2 Techniques used</b>	<b>2</b>
1.2.1 Examine Documents	2
1.2.2 Interview	2
1.2.3 Research	2
<b>1.3 Introduction to Enterprise/Organization</b>	<b>3</b>
<b>1.4 Structure of the Enterprise</b>	<b>3</b>
<b>1.5 Data views and operations for user groups</b>	<b>3</b>
<b>Section 2:</b> Conceptual Database Design	<b>4-24</b>
<b>2.1 Entity Set Description</b>	<b>4-19</b>
2.1.1 Section	4-6
2.1.2 CourseInfo	7-8
2.1.3 Person	9-10
2.1.4 Instructor	11
2.1.5 Student	13
2.1.6 Assignment	14-16
2.1.7 Grade	17-18
2.1.8 Student sign-up for a Section Map	19
<b>2.2 Relationship Set Description</b>	<b>20-21</b>
2.2.1 for	20
2.2.2 teachers	20
2.2.3 signup	20
2.2.4 gets	21
2.2.5 has	21
2.2.6 attached	21
<b>2.3 Related Entity Set</b>	<b>22-23</b>
<b>2.4 E-R Diagram</b>	<b>24</b>

---

**Section 3: E-R Model and Relational Model** **25-28**

<b>3.1 Description of E-R and Relational Model</b>	<b>25</b>
3.1.1 History of the E-R Model and the Relational Model	25
3.1.2 What is the E-R Model and the Relational Model	25-26
3.1.3 What are the major features of the E-R and Relational Model	26
3.1.4 What is the purpose of the E-R and Relational Model	27
<b>3.2 Comparison of the E-R Model and Relational Model</b>	<b>27</b>
<b>3.3 Conversion for the E-R Model to the Relational Model</b>	<b>27-29</b>
3.3.1 Why the conversion is needed?	27
3.3.2 The method for converting from E-R to Relational Model	27
3.3.3 Strong and weak entities	28
3.3.4 Entity types simple and composite attribute	28
3.3.5 Entity types single and multi-valued attributes	28
3.3.6 Relationship types with one-to-one, one-to-many, many-to-one, many-to-many	28-29
3.3.7 'IsA' (superclass and subclasses) relationship	29
3.3.8 'HasA' relationship	29
3.3.9 Relationship type involving other relationship type	29
3.3.10 Recursive relationship(involving one entity types)	29
3.3.11 Relationships involving more then 2 entity types	29
<b>3.4 Define Constraints and how does DBMS enforce constrains</b>	<b>30</b>
3.4.1 Entity constraint	30
3.4.2 Referential constraints	30
3.4.3 Primary key and unique key constraints	30
3.4.4 Check constraints and business rules	30

---

**Section 4: Convert the E-R database into relational database schema** **31-49**

<b>4.1 Entity to relation</b>	<b>31-34</b>
4.1.1 Section	31
4.1.2 CourseInfo	31
4.1.3 Person	32
4.1.4 Instructor	32
4.1.5 Student	32
4.1.6 Assignment	33
4.1.7 Grade	33
4.1.8 Signup	34

<b>4.2 Relationship Set Description</b>	<b>35-43</b>
4.2.1 Section	35
4.2.2 CourseInfo	35-37
4.2.3 Person	38-39
4.2.4 Instructor	39
4.2.5 Student	40
4.2.6 Assignment	40-41
4.2.7 Grade	42-43
4.2.8 Signup	43
<b>4.3 Non-trivial queries in relational algebra, tuple relational calculus, and domain relational calculus.</b>	<b>44-49</b>
4.3.1 List all student id's who has taken "cs215".	44
4.3.2 List all instructors who taught "cs221".	44
4.3.3 List all course short names, section crn, start date, end date, and worth for Fall(season_num=0) 2005	45
4.3.4 List all grade letters, grade points for student 165299473 in Winter(season_num=1) 2004.	45
4.3.5 List student's id, first & last name, average, min, max final grade points between 2004 and 2005.	46
4.3.6 List all students signed up for "cs221" in Fall 2005, taught by "Chung"(last name).	46
4.3.7 List all assignments for "cs223" in Spring(season_num=2) 2006, taught by instructor 346970481.	47
4.3.8 List assignment name, grade, max points, and is extra for student 369219713 in "cs350" Fall(season_num=0) 2005 taught by instructor 346970481.	48
4.3.9 List the min, max, and average points for all students in "cs457" Fall(season num=0) 2005, taught by instructor 320362253, on assignment "test"	49
4.3.10 Students that have taken all the courses.	49

<b>Section 5: SQL*PLUS</b>	<b>50-100</b>
<b>5.1 SQL*PLUS</b>	<b>50</b>
<b>5.2 Oracle Schema Objects</b>	<b>51-59</b>
5.2.1 Tables	51
5.2.2 Views	52
5.2.3 Objects	52-53
5.2.4 Sequences	54
5.2.5 Stored Procedures & Functions	54-55
5.2.6 Indexes	56
5.2.7 Synonyms	57
5.2.8 Clusters	57
5.2.9 Packages	58
5.2.10 Triggers	58-59
<b>5.3 Schema objects that were created in this project</b>	<b>60-63</b>
5.3.1 Course Information: Table & Sequence	60
5.3.2 Person: Table & Sequence	60
5.3.3 Instructor: Table & Sequence	61
5.3.4 Section: Table & Sequence	61
5.3.5 Sign-up: Table & Sequence	62
5.3.6 Assignment: Table & Sequence	62
5.3.7 Grade: Table & Sequence	63
<b>5.4 Relation Schemes</b>	<b>64-77</b>
5.4.1 js_course_info	64-67
5.4.2 js_person	68-69
5.4.3 js_instructor	70
5.4.4 js_section	71
5.4.5 js_assign	72-73
5.4.6 js_grade	74-75
5.4.7 js_signup	76-77
<b>5.5 Sample SQL Queries</b>	<b>78-93</b>
5.5.1 List all students id's who has taken "cs215".	78
5.5.2 List all instructors who taught "cs221".	79
5.5.3 List all course short names, section crn, start date, end date, and worth for Fall(season_num=0) 2005	80
5.5.4 List all grade letters, grade points and course short name, for student 165299473 in Fall(season_num=0) 2005.	81
5.5.5 List student's id, first & last name, average, min, max final grade points between 2004 and 2005.	82
5.5.6 List all students signed up for "cs342" in Fall(season_num=0) 2005, taught by "Wang"(last name).	83

## **Section 5: SQL\*PLUS**

**50-100**

5.5.7 List all assignments for “cs223” in Fall(season_num=0) 2005, taught by instructor 346970481. (Sort by assignment name)	84
5.5.8 List assignment name, grade, max points, and is extra for student 369219713 in “cs223” Fall(season_num=0) 2005 taught by instructor 346970481. (Sort by assignment name)	85
5.5.9 List the min, max, and average points for each student in “cs223” Fall 2005, taught by instructor 346970481, on all assignment.	86
5.5.10 Students that have taken all the courses.	87
5.5.11 List all student id's and course short name that have not gotten a grade for a course that they signup for. (Sorted by course name)	88
5.5.12 List all the latest grades.	89
5.5.13 List all assignments ID, grade ID, and grade points for all assignments. If a grade does not exist for that assignment then include it but leave the grade blank. (using a outer join)	90-91
5.5.14 Create new table and insert all grades older then 60 days.	92-93
<b>5.6 Loading Data Methods</b>	<b>94-100</b>
5.6.1 Description of INSERT Statement	94
5.6.2 Automatic Data Insertion	94-96
5.6.3 Modification of the DataLoader Program	96
5.6.4 DataLoader.java	97-100

## **Section 6: Oracle PL/SQL**

**101-122**

<b>6.1 Common Features in Oracle PL/SQL and MS Trans-SQL</b>	<b>101-102</b>
6.1.1 Components which consist of PL/SQL and T-SQL	101-102
6.1.2 Purposes of stored subprogram	102
6.1.3 Benefits of calling a stored subprogram	102
<b>6.2 Oracle PL/SQL</b>	<b>103-112</b>
6.2.1 Program Structure	103-104
6.2.2 Control Statement	105-107
6.2.3 Cursors	107
6.2.4 Stored Function	108
6.2.5 Stored Procedure	109
6.2.6 Package	110-111
6.2.7 Trigger	112

## **Section 6: Oracle PL/SQL**

**101-122**

<b>6.3 Oracle PL/SQL Subprogram</b>	<b>113-122</b>
6.3.1 View(s)	113-119
6.3.2 Stored Procedure(s)	120-121
6.3.3 Trigger(s)	122

## **Section 7: Users & the GUI Client Application**

**123-154**

<b>7.1 User Groups</b>	<b>123-124</b>
7.1.1 Database Administrator(s)	123
7.1.2 Instructor(s)	124
7.1.3 Student(s)	124
<b>7.2 Relations, Views and Subprograms related to Group Activities</b>	<b>125-130</b>
7.2.1 IAGS Tables	125
7.2.2 Instructor Views	126-129
7.2.3 Instructor Stored Procedures	130
7.2.4 Other Views	130
<b>7.3 Using the GUI Client Application</b>	<b>131-137</b>
7.3.1 Logging In/Out	131
7.3.2 DB Administrator User	132-133
7.3.3 Instructor User	134-136
7.3.4 Help & About	136
7.3.5 EULA	137
<b>7.4 Implementing the GUI Client Application</b>	<b>138-153</b>
7.4.1 Designing the User Interface	138
7.4.2 Major Class/Interface Description	139-152
1. cIAGS class	139-141
2. cDBItemList class	141-142
3. cDB class	143
4. cDB_Base interface	144-145
5. cBaseItem class	146
6. cSqlValueList class	147-148
7. Classes for User Groups	149-151
8. Class Diagram	152
7.4.3 Major Features	153
7.4.4 The Learning Process	153
<b>7.5 The Path to Success</b>	<b>154</b>

## **Section 1: Fact-Finding Techniques and Information Gathering**

In this section I will be discussing the fact-finding techniques and information gathering. Which includes description of fact-finding techniques, techniques used for this particular database design, an introduction to enterprise/organization, the structure of the enterprise, and data views & operations for user groups.

### **1.1 Description Fact-Finding Techniques**

Fact-Finding techniques are a very important part of the database design, because if the all the facts are not accounted for and the database is designed incorrectly. Changing the database design after implementation can lead to lot of problems, include a complete rewrite of all the software that uses the databased designed.

#### **1.1.1 Examine Documents**

Examining the documents is a useful method of fact finding. The documents give you a basis for all the information that will need to be entered. Several examples of documents are forms, reports, memos, raw data, charts, diagrams... All documents related the operation of the companies database should be examined in detail and review through the development process. Another useful part of examining the documents, is to look at what type of information that is entered in the to field and how the data is related the other document or entries.

It is a good idea to start by looking at the documents before interviewing, because you can ask the interviewer why the document were designed in the current fashion and how the information is entered on to the document(s).

#### **1.1.2 Interview**

Interviewing is one of the most common and most useful methods of fact finding. It allows you to interacted with people who should know how the data is currently structured. It is also useful because the people you interview will most likely be using the database you design. An example of some of the questions you may ask the interviewee: "Would it be a good idea to have a history of...?", "Is there anything that is not currently documented and needs/should be recorded?".

One drawback of just interviewing a person or persons, is that they may not know all aspects of the company and/or forget a crucial aspect of the company. This why it is not good to rely only one type of fact finding.



### 1.1.3 Research

Researching the companies field, is a good way to expand into other areas that the company has not thought of. Making it easy to expand the database at a later time. Furthermore, researching how other companies in the same field, design their database and/or documents can be a good way to learn from others mistakes.

Examples of ways to research would be to search online news forums, online bulletin boards, reference material, magazine and journal articles.

### 1.1.4 Questioning

A questioner can ask a large number of people how to improve the current system. This can be useful when there large number of people who will be interacting with the database. The questioner should be designed in such a way that the questions are not ambiguous, they should be clear and concise. So the database designer can derive an accurate conclusion.

### 1.1.5 Observations

Observations are when you passively record how a person(s) manages/enters the current set of documentation. This gives the database designer a birds eye view of how they currently record information, generate reports, charts, diagrams and so forth.

## 1.2 **Techniques used**

For the project in particular, the techniques used were as follows:

Examining Documents

Interviewing

Research

I also based some of the database designed on my own personal experiences. Keeping track of my own grades for the course that I have taken and taught.

### 1.2.1 Examine Documents

I examined grading books and course syllabuses from several different instructors.

### 1.2.2 Interview

I spoke with several instructors getting their opinion, on how they keep track of the grades currently and what their ideal way of keeping track would be.

### 1.2.3 Research

I searched online for grading systems such as WebCT and Banner.

### **1.3 Introduction to Enterprise/Organization**

The project is not for an enterprise, business or any particular organization. It is a grade tracking systems to assist one or more instructor in keeping track of the grade for the course(s) they are teaching.

### **1.4 Structure of the Enterprise**

The structure of the enterprise is such that it aids the instructor(s) in keeping track of the grades. It will keep a historical record of past courses, sections, students, instructors and grades in compliance with most of institutional rule(s).

The instructor or a trusted teacher assistant will be the only person(s) adding/updating grades, assignments and the instructor's information who logged in. The course information, students, sections and instructors will be managed by the database manager until another superiors level can be added at a later date. The student will only be able to view their own information once logged in. (see 1.5 for more details)

### **1.5 Data views and operations for user groups**

The data views should be split into two major types; instructor views and student views. The instructor and any trusting aid will be able to update/add to any of the records in the database. The login for the instructor should be based on a user ID and password system. Students on the other hand will only be able to view there own particular records in the database. The students login should be based on there full ID number only.

The instructor will be able to view/edit any of the following:

- A list of all section for any particular year and quarter. (view only)
- A list of all students for any particular section, year and quarter. (view only)
- A list of all assignments for any particular section, year and quarter. (view/add/edit only)
- A list of all grades with student info. for a particular assignment and section. Including all student's signed up for this section, even if they don't have a grade. (view/edit only)
- Generate a report of all students final grade info., for any particular section, year and quarter. Only showing the students last 4 numbers from there ID, final grade points, and final grade letter.

The students should be able to only view any of the following:

- A list of course sections and grades for this particular students.
- A list of assignment grades from the a particular student, section, year and quarter.

## Section 2: Conceptual Database Design

In this section I will be discussing the conceptual database design. The conceptual database design consists of the entity set description, relationship set description, related entity set and entity relationship diagram(also known as a ER-model).

### 2.1 Entity Set Description

An entity set is defined as a group of objects(like a class in c++) with the same attributes are identified as having independent existence.

The following is the entity set description for the Instructor Assistant Grading System database design:

#### 2.1.1 Section

##### a) **Description**

The section entity connects the instructor, course information, and students together. It contains the following: CRN, quarter/semester number, start date, end date, max number of students.

##### b) **Attributes**

---

Name:	<b>section_id</b>
Description:	<b>This is the automatically generated section ID.</b> Example: 123456789
Domain Type:	<b>integer</b>
Value Range:	<b>0 – 999999999</b>
Default Value:	<b>automatic</b>
Null Allowed:	<b>no</b>
Unique:	<b>yes</b>
Type, Value:	<b>simple, single</b>

---

---

Name:	<b>crn</b>
Description:	<b>This is the course number.</b> Example: 123456789
Domain Type:	<b>integer</b>
Value Range:	<b>0 – 999999999</b>
Default Value:	<b>0</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

---

Name:	<b>season_num</b>
Description:	<b>The season(quarter/semester) number of the section taught.</b> Example: 0 (0 = Fall, 1 = Winter,...)
Domain Type:	<b>integer</b>
Value Range:	<b>0 – 9</b>
Default Value:	<b>0</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---



---

Name:	<b>start_date</b>
Description:	<b>The start date of the section taught.</b> Example: 2006-25-09
Domain Type:	<b>date</b>
Value Range:	<b>1900-01-01 - 9999-31-12</b>
Default Value:	<b>today</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---



---

Name:	<b>end_date</b>
Description:	<b>The end date of the section taught.</b> Example: 2006-25-09
Domain Type:	<b>date</b>
Value Range:	<b>1900-01-01 - 9999-31-12</b>
Default Value:	<b>today</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

---

Name:	<b>max_num_students</b>
Description:	<b>The maximum number of students allowed to sign-up for a section(the maximum classroom size).</b> Example: 20
Domain Type:	<b>integer</b>
Value Range:	<b>1 - 999</b>
Default Value:	<b>1</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Candidate Key:	<b>section_id</b>
Primary Key:	<b>section_id</b>
Entity Type:	<b>weak</b>
Indexed Field:	<b>section_id</b>

### 2.1.2 **CourseInfo**

#### a) **Description**

The course information entity stores a course's short name, long name, a long description and worth(number of credits/units).

#### b) **Attributes**

---

Name:	<b>course_id</b>
Description:	<b>This is the automatically generated course ID.</b> Example: 123456789
Domain Type:	<b>integer</b>
Value Range:	<b>0 – 999999999</b>
Default Value:	<b>automatic</b>
Null Allowed:	<b>no</b>
Unique:	<b>yes</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>short_name</b>
Description:	<b>This is short name of the course.</b> Example: cs342
Domain Type:	<b>character string</b>
Value Range:	<b>32 characters long</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>long_name</b>
Description:	<b>This is long name(short description) of the course.</b> Example: Database System
Domain Type:	<b>character string</b>
Value Range:	<b>128 characters long</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

---

Name:	<b>description</b>
Description:	<b>This is description of the course.</b> Example: The database system course is the best course in history.
Domain Type:	<b>character string</b>
Value Range:	<b>1024 characters long</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

---

Name:	<b>worth</b>
Description:	<b>This is the worth(units/credits) of the course.</b> Example: 5
Domain Type:	<b>integer</b>
Value Range:	<b>0 - 99</b>
Default Value:	<b>0</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Candidate Key:	<b>short_name</b>
Primary Key:	<b>course_id</b>
Entity Type:	<b>strong</b>
Indexed Field:	<b>course_id</b>

### 2.1.3 **Person**

#### a) **Description**

The person entity is a super class of the student and instructor entity. It contains the following: id, name(first, middle and last), email, and phone.

#### b) **Attributes**

---

Name:	<b>id</b>
Description:	<b>This is a unique person ID.</b> Example: 123456789
Domain Type:	<b>integer</b>
Value Range:	<b>0 – 999999999</b>
Default Value:	<b>0</b>
Null Allowed:	<b>no</b>
Unique:	<b>yes</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>name</b>
Description:	<b>This is a the person's first, middle and last name.</b> Example: Joseph Edward Sutton
Domain Type:	<b>characters string</b>
Value Range:	<b>96</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>composite, single</b>

---

Name:	<b>p_email</b>
Description:	<b>This is the person's primary email address.</b> Example: jsutton@helios.cs.csubak.edu
Domain Type:	<b>characters string</b>
Value Range:	<b>128</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---



---

Name:	<b>p_phone</b>
Description:	<b>This is the person's primary phone number.</b> Example: 1-123-123-1234
Domain Type:	<b>characters string</b>
Value Range:	<b>16 characters</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Candidate Key: **id**  
 Primary Key: **id**  
 Entity Type: **strong**  
 Indexed Field: **id**

#### 2.1.4 **Instructor**

##### a) **Description**

The instructor entity contains information about a particular type of person an instructor. This is a sub class of **Person**.

##### b) **Attributes**

---

Name:	<b>password</b>
Description:	<b>This is the password that the instructor will use to log into and manage the students, assignments, and grades.</b> Example: snoopy5
Domain Type:	<b>character string</b>
Value Range:	<b>32 characters</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>website</b>
Description:	<b>This is the website for the instructor.</b> Example: <a href="http://www.cs.csubak.edu/~jsutton/">http://www.cs.csubak.edu/~jsutton/</a>
Domain Type:	<b>character string</b>
Value Range:	<b>64 characters</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>s_email</b>
Description:	<b>This is the secondary email address.</b> Example: JS@PolyFaust.com
Domain Type:	<b>characters string</b>
Value Range:	<b>128</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

---

Name:	<b>s_phone</b>
Description:	<b>This is the secondary phone number.</b> Example: 1-123-123-1234
Domain Type:	<b>characters string</b>
Value Range:	<b>16 characters</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Candidate Key:	<b>none</b>
Primary Key:	<b>none (derived from Person entity)</b>
Entity Type:	<b>weak</b>
Indexed Field:	<b>none</b>

### 2.1.5 Student

#### a) **Description**

The student entity contains information about a particular type of person a student.  
This is a sub class of **Person**.

#### b) **No Attributes**

All attributes derived from super class **Person**.

Candidate Key: **none**

Primary Key: **none (derived from Person entity)**

Entity Type: **weak**

Indexed Field: **none**

### 2.1.6 Assignment

#### a) **Description**

The assignment entity contains information about a particular assignment. An assignment can be a test, quiz, mid term, final, lab, homework... so on and so forth. It is a generic type of assignment given to students which will be taken into account then the grade points are added up. It contains the following information: assignment id, name of assignment, is this assignment extra credit, and details about this assignment.

#### b) **Attributes**

---

Name:	<b>assign_id</b>
Description:	<b>This is the automatically generated assignment ID.</b> Example: 123456789
Domain Type:	<b>integer</b>
Value Range:	<b>0 – 999999999</b>
Default Value:	<b>automatic</b>
Null Allowed:	<b>no</b>
Unique:	<b>yes</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>name</b>
Description:	<b>This is the name of this assignment.</b> Example: Mid Term Test
Domain Type:	<b>character string</b>
Value Range:	<b>64 characters</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>type_num</b>
Description:	<b>This is the type of assignment. Each type is given a unique number, for instance a test would be 0, quiz is 1, lab are 2...</b> Example: 0
Domain Type:	<b>integer</b>
Value Range:	<b>0 – 99</b>
Default Value:	<b>0</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>max_points</b>
Description:	<b>This is the maximum points for this assignment.</b> Example: 200
Domain Type:	<b>integer</b>
Value Range:	<b>0 – 999999999</b>
Default Value:	<b>0</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>is_extra_credit</b>
Description:	<b>This is this assignment extra credit, true/false.</b> Example: false
Domain Type:	<b>boolean</b>
Value Range:	<b>true/false</b>
Default Value:	<b>false</b>
Null Allowed:	<b>no</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

---

Name:	<b>details</b>
Description:	<b>This is the assignment details. It could be used to store the test key or just notes about the test.</b> Example: Through out several questions because of ambiguity.
Domain Type:	<b>variable character string</b>
Value Range:	<b>up to 1024 characters</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Candidate Key:	<b>assign_id</b>
Primary Key:	<b>assign_id</b>
Entity Type:	<b>weak</b>
Indexed Field:	<b>assign_id</b>

### 2.1.7 Grade

#### a) **Description**

The grade entity is a super class of current grade. The grade entity also keeps track of all grades from all assignments for all sections for all students. When a grade is entered/updated the current grade of an assignment is not removed but simply archived. This will keep a written history of all grade changes for any one student and assignment.

#### b) **Attributes**

---

Name:	<b>grade_id</b>
-------	-----------------

Description:	<b>This is the automatically generated grade ID.</b> Example: 123456789
--------------	--

Domain Type:	<b>integer</b>
--------------	----------------

Value Range:	<b>0 – 999999999</b>
--------------	----------------------

Default Value:	<b>automatic</b>
----------------	------------------

Null Allowed:	<b>no</b>
---------------	-----------

Unique:	<b>yes</b>
---------	------------

Type, Value:	<b>simple, single</b>
--------------	-----------------------

---

Name:	<b>points</b>
-------	---------------

Description:	<b>This is the number of points for an assignment.</b> Example: 185
--------------	--

Domain Type:	<b>integer</b>
--------------	----------------

Value Range:	<b>0 – 999999999</b>
--------------	----------------------

Default Value:	<b>0</b>
----------------	----------

Null Allowed:	<b>no</b>
---------------	-----------

Unique:	<b>no</b>
---------	-----------

Type, Value:	<b>simple, single</b>
--------------	-----------------------

---

Name:	<b>date_entered</b>
-------	---------------------

Description:	<b>This is the date and time at with this grade was entered.</b> Example: 2006-25-09 11:55 p.m.
--------------	--

Domain Type:	<b>date time</b>
--------------	------------------

Value Range:	<b>1900-01-01 12:00 a.m. - 9999-31-12 11:59 p.m.</b>
--------------	--

Default Value:	<b>now</b>
----------------	------------

Null Allowed:	<b>no</b>
---------------	-----------

Unique:	<b>no</b>
---------	-----------

Type, Value:	<b>simple, single</b>
--------------	-----------------------

---



---

Name:	<b>details</b>
Description:	<b>This is the grades details. It could be used to store the student's test key or just some notes.</b> Example: Student turned in assignment late, reduced by 25%
Domain Type:	<b>variable character string</b>
Value Range:	<b>up to 1024 characters</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Candidate Key:	<b>grade_id</b>
Primary Key:	<b>grade_id</b>
Entity Type:	<b>weak</b>
Indexed Field:	<b>grade_id</b>

### 2.1.8 Student sign-up for a Section

#### a) **Description**

The section student map entity establishes a relationship between students and sections. The entity will store the student final grade, date entered and details.

#### b) **Attributes**

---

Name:	<b>grade_letter</b>
Description:	<b>This is the student's final grade for a particular section.</b> Example: A
Domain Type:	<b>characters string</b>
Value Range:	<b>0 – 16</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>grade_points</b>
Description:	<b>This is the student's final grade points for a particular section. I can be used to store the percentage or any final calculation.</b> Example: 0.95
Domain Type:	<b>float</b>
Value Range:	<b>0.00 – 3.40E + 38</b>
Default Value:	<b>0</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Name:	<b>details</b>
Description:	<b>This is the student section details. It could be used to store comments about the final grade.</b> Example: The student was in the top 5%.
Domain Type:	<b>variable character string</b>
Value Range:	<b>up to 1024 characters</b>
Default Value:	<b>“”</b>
Null Allowed:	<b>yes</b>
Unique:	<b>no</b>
Type, Value:	<b>simple, single</b>

---

Candidate Key:	<b>none</b>
Primary Key:	<b>none</b>
Entity Type:	<b>weak</b>
Indexed Field:	<b>none</b>

---

## 2.2 Relationship Set Description

The relationship set description, describes how two entities are related to each other.

The following is the relationship set description for the Instructor Assistant Grading System database design:

### 2.2.1 **for**

Description: For is the relationship between the course information and the section. There is only be one course information for any one section.

Entities involved: **Course Info, Section**

Cardinality: 1 to Many

Descriptive Field: none

Constraint: Optional

### 2.2.2 **teach**

Description: Teach is the relationship between the instructor and section. There is only one teacher per section.

Entities involved: **Instructor, Section**

Cardinality: 1 to Many

Descriptive Field: none

Constraint: Optional

### 2.2.3 **signup**

Description: Signup is the relationship between the student and section. A student signs-up for zero or more sections and a section can have zero or more students.

Entities involved: **Student, Section**

Cardinality: Many to Many

Descriptive Field: The Student sign-up for a Section is the descriptive field. See **2.1.8**

Constraint: Optional

#### 2.2.4 gets

Description: Gets is the relationship between the student and the grade. A student gets a grade for an assignment. The grades will be a historical record of the grade updates for a student and an assignment. So when selecting the records for a student and assignment one or more records could be return. The current grade for a students assignment will be the one with the most current date.

Entities involved: **Student, Grade**

Cardinality: 1 to Many

Constraint: Optional

#### 2.2.5 has

Description: Has is the relationship between the section and assignment. Each section will have one or more assignments.

Entities involved: **Section, Assignment**

Cardinality: 1 to Many

Constraint: Optional

#### 2.2.6 attached

Description: Attached is the relationship between the assignment and grade. Each assignment can have one or grades as a historical record of all the changes to the grade.

Entities involved: **Assignment, Grade**

Cardinality: 1 to Many

Constraint: Optional

## 2.3 Related Entity Set

### 2.3.1 Specialization/Generalization Relationships(is-A)

Specialization/Generalization is a term assigned to special types of entities called superclass, subclass and how they relate to each other. The subclass is a child of the superclass. It inherits the attributes from the superclass similar to the way a child class inherits a parent class member(s) in an object oriented language. The specialization and generalization relationship is called the “is-A” hierarchy.

Specialization is the process of maximizing the differences between member of an entity by identifying their distinguishing characteristics. It is a top-down approach to define a set of superclasses and their related subclasses. When we identify a set of subclasses of an entity type we then associate attributes specific to each subclass (where necessary), and also identify any relationship between each subclass and other types of subclasses.<sup>1</sup>

Generalization is the process of minimizing the differences between entities by identifying their common characteristics. It is a bottom-up approach, which result in the identification of a generalization superclass from the original entity types. If we apply the process of generalization on an entity set, we attempt identify the similarities between them such as common attributes and relationships.<sup>1</sup>

There are two constraints applied to specialization and generalization. One is participant and the other is disjoint.

#### a) Participation constraint

Determines whether a member of the superclass can be a member of a subclass. The participation constraint can be either mandatory or optional. So when it's optional, a member of a superclass does not need to be a member of the subclass. When it's mandatory a member of a superclass has to be a member of the subclass. The mandatory or optional is show between the relationship to signify the super and sub class's participant.

#### b) Disjoint constraint

Is only applied when a superclass has more then one subclasses. Subclasses are considered disjoint when an entity occurrence can be a member only one of the subclasses. If this is true, then an “OR” is placed next to the participant constraint. If not then it's considered disjoint and “AND” is placed next to the participant constraint.

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition, pg. 374-375.

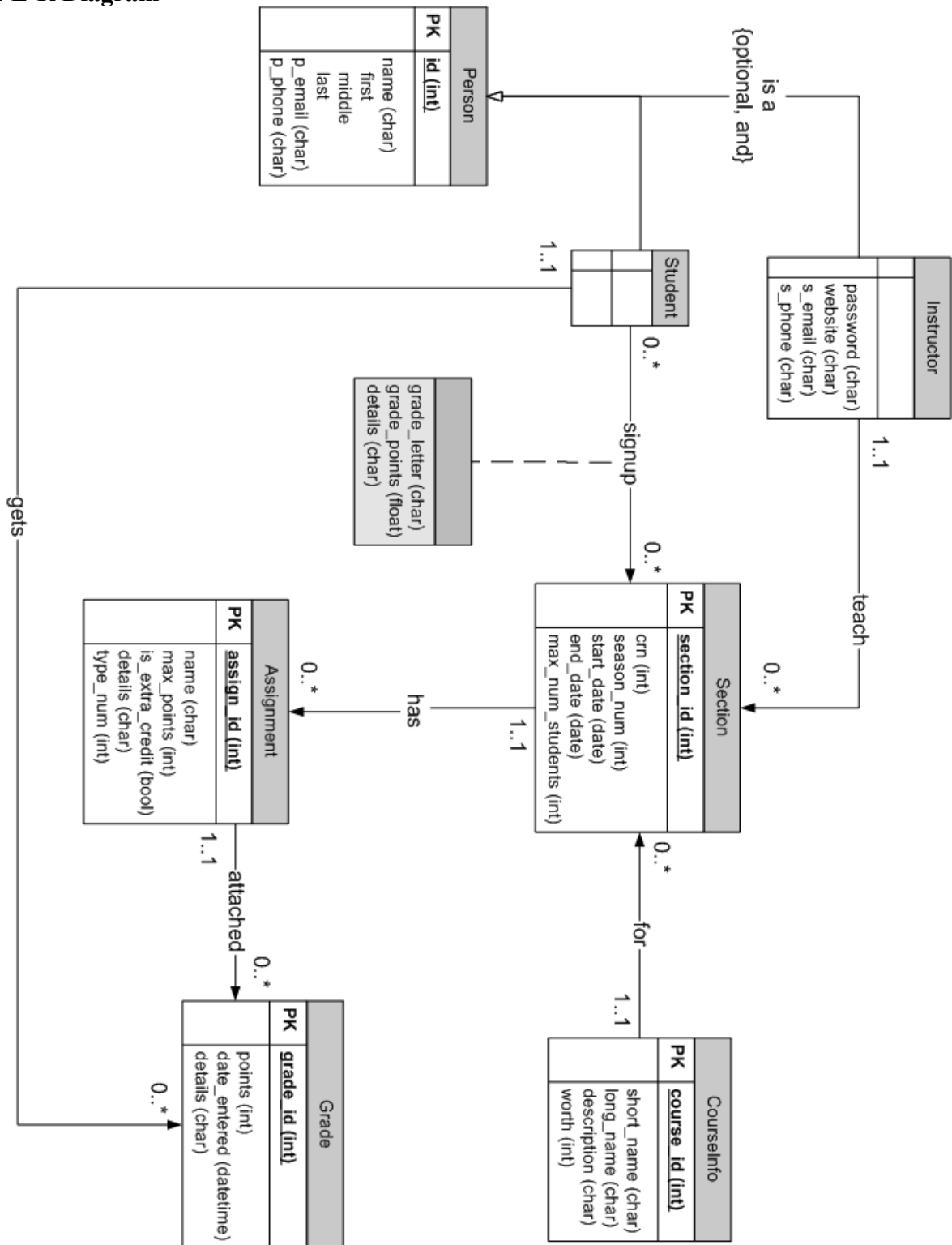
### 2.3.2 Aggregation/has-relationship

Aggregation is a relationship associated with two entity types, where one is the whole and the other is the part. It's commonly referred to as a "has-a" or "is-part-of" relationship, because one is part of the other. It's used to relate one entity to another. For example, an item is part of a sale. There is a specific form of aggregation called composition. Composition represents an association between entities, where there is a strong ownership & coincidental lifetime between the whole and the part. In composition the whole is responsible for the disposition of the parts, which means that the composition must manage the creation and destruction of its parts.<sup>1</sup> For example, test entity is a composite of assignment.

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition, pg. 374-375.

## 2.4 E-R Diagram



## Section 3: E-R Model and Relational Model

In this section I will be describing the E-R and Relational Model. The history, what is the E-R model/Relational model, the major features and purpose, comparison, conversion, and defined constraints.

### 3.1 Description of E-R and Relational Model

#### 3.1.1 History of the E-R Model and Relational Model

The Entity Relationship model was proposed by Peter Chen in a paper published in 1976. The paper was titled "The Entity Relationship Model - Toward A Unified View of Data". The purpose of the new model was to be a more of a natural view that the real world consists of entities and relationships.

The E-R model is perfect for conceptualizing databases, but does not help with the implementing a design. So, about seven years prior to the development of the Entity Relation model, Edgar F. Codd (Oxford trained mathematician), proposed the relational database model in his paper "A relational model of data for large shared data banks."

In 1976 IBM developed the most influential relational database prototype called System R. Its features included query optimization, transaction management, concurrency control, recovery techniques and data security. The System R initial design was the forerunner of the development of the SQL (Structured Query Language).

#### 3.1.2 What is the E-R Model and the Relational Model

The Entity-relationship model (E-R model) is a model providing a high-level description of a conceptual data model. Data modeling provides a graphical notation for representing such data models in the form of entity-relationship diagrams (ERD). The first stage of information system design uses these models to describe information needs or the type of information that is to be stored in a database during the requirements analysis. The data modeling technique can be used to describe any ontology (i.e. an overview and classifications of used terms and their relationships) for a certain universe of discourse (i.e. area of interest). In the case of the design of an information system that is based on a database, the conceptual data model is, at a later stage (usually called logical design), mapped to a logical data model, such as the relational model; this in turn is mapped to a physical model during physical design. Note that sometimes, both of these phases are referred to as "physical design".<sup>1</sup>

---

<sup>1</sup> Wikipedia - [http://en.wikipedia.org/wiki/Entity-relationship\\_model](http://en.wikipedia.org/wiki/Entity-relationship_model)



The relational model is that all data are represented as mathematical n-ary relations, an n-ary relation being a subset of the Cartesian product of n domains. In the mathematical model, reasoning about such data is done in two-valued predicate logic, meaning there are two possible evaluations for each proposition: either true or false (and in particular no third value such as unknown, or not applicable, either of which are often associated with the concept of NULL). Some think that logic (which is inherently two-valued) is an important part of the relational model, where others think that a system that uses a form of three-valued logic can still be considered relational. Data are operated upon by means of a relational calculus or algebra, these being equivalent in expressive power. The relational model of data permits the database designer to create a consistent, logical representation of information. Consistency is achieved by including declared constraints in the database design, which is usually referred to as the logical schema. The theory includes a process of database normalization whereby a design with certain desirable properties can be selected from a set of logically equivalent alternatives.<sup>2</sup>

### 3.1.3 What are the major features of the E-R and Relational Model

The major features of an E-R Model includes **entities**, which are groups of objects that have the same properties. They are identified by having an independent existence and **relations**, which is a meaningful association between the entities.

**Entity** is at the heart of the model, which can be thought of as an object. Each entity is given a name and has one or more attribute. The entity is similar to a class in an OOP language. It has a name and the attributes are similar to the data member of that class.

**Relationships** show how one entity relates to another entity. It has a direction, name(descriptor) and the number range that is allowed for that relationship. One example would be an Instructor teach a Section. You would also say that only one Instructor is allowed to teach zero or more Sections.

The aggregate relationships also known as a “is-a” or “has-a” relationships. As an example of this is a store has items, and an orange is-a product that the store sells.

The relational model consists of relations(known as tables), containing columns and rows. Each table has tuples & attributes and has a respective degree. An attribute is the named column in a relation. A tuple is a single row of data in the relation. The degree of the relation is the number of attributes it possesses.

---

<sup>2</sup> Wikipedia - [http://en.wikipedia.org/wiki/Relational\\_model](http://en.wikipedia.org/wiki/Relational_model)

### 3.1.4 What is the purpose of the E-R and Relational Model

The Entity Relationship model is also used to display the cardinality of entity relationships. The cardinality of a relationship is the number of times a given entity type can relate to another entity type.

The objectives of a relational database are as follows<sup>1</sup>:

1. To allow a high degree of data independence.
2. To provide substantial grounds for dealing with data semantics, consistency, and redundancy problems.
3. To enable the expansion of set-oriented data manipulation languages.

## 3.2 Comparison of the E-R Model and Relational Model

Although the E-R and Relational models are similar, there are a few differences:

1. The E-R model may display extended attributes for a particular relationship. In a relational model the extended attributes will be its own table or merge to be part of another relation.
2. The E-R model supports inheritance though aggregate(is-a) relationship. The relational model does not support this.
3. The relational model does not separate entities and relationships into different types. Everything is a relation(also known as a table).
4. The E-R model has no real language associated with it. The relational model on the other hand does. Typically a relational database is designed using SQL (Structured Query Language).

## 3.3 Conversion for the E-R Model to the Relational Model

In this sub section I will describe the ways to convert an E-R model to a relational model. Why we convert to the relational model, the methods, strong & weak entities, simple & composite entity types, single & multi-value entity types, the types of relationship, is-a, has-a, types of relations, recursive relations, and more than two relations.

### 3.3.1 Why the conversion is needed?

The conversion is needed so the database design can be implemented. The database management system only provides a relational model design structure using the structured query language.

### 3.3.2 The method for converting from E-R to Relational Model

The quickest and simplest method is to use it by hand, it's not very difficult. Another method is to use a software application to do the conversion for you but this has mixed results.

---

1 Connolly & Begg, Database Systems 4<sup>th</sup> edition

### 3.3.3 Strong and weak entities

Strong – Entities are entities that have two or more relationships with other entities. Use the entity type name as relation name, attributes, primary key and domain attributes. Strong entities may create entire relations within the Relational Model by using its attributes

Weak – Entities that only have one relation with another entity in an E-R Model are called weak entities. By looking at a weak entity, one can not simply look at the entities and decipher what the primary key is. There is a possibility that the primary key maybe later found in the parent or strong entity of the associated weak entity.

### 3.3.4 Entity types simple and composite attributes

Simple – Has the same name and domain is in the E-R model.

Composite – Each part(component) of the composite attribute will be a separate attribute in a relation.

For example an entity has a name in the E-R and it's composite into first, middle and last. In the relational model they would all be separate attributes called fname, mname, and lname. Another approach is to just flatten the data to all one type but can limit sorting and searching and/or make it very difficult.

### 3.3.5 Entity types single and multi-valued attributes

Single – Each attribute of the entity type as one of the attribute of the relation.

Multi-valued – The multi-valued attribute in an E-R model holds multiple values. In the relational model each attribute can only hold one value. So each value in the multi-value would be a new attribute. The second approach would allow for the number of values to grow and shrink to any size by creating a new relation, however this would add more complexity to the relational model.

### 3.3.6 Relationship types with one-to-one, one-to-many, many-to-one, many-to-many

One-to-One(1:1) – The one-to-one relationship you make a relationship type into a separate relation. Then add the primary key of the first entity type into the second entity type. There are three different cases of the one-to-one relationship. Once case mandatory participation on both sides, another is mandatory participation on one side, and optional participation on both sides. When looking at mandatory participation on both sides, the designer can include both entity types in a single relation, and then choose a primary key between the entities. When working with mandatory participation on one side, the entity type that has the mandatory constraint becomes the child and the other entity type becomes the parent. The primary key would most likely come from an entity type with optional participation. In the optional participation the designer must look at the relationships and determine which primary key is needed for a particular relation.<sup>1</sup>

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

One-to-Many(1:\*) – The one-to-many relationship will be converted into two separate relations where the primary key from the each relation will be add to the other relation as a foreign key. Another method is to only add the primary key from the “many” to the “one” side relation as a foreign key. The method use depends on the design.

Many-to-One(\*:1) – Same a one-to-many but flipped.

Many-to-Many(\*:\*) – The many-to-many relationship will be converted to a new relation. With the primary keys from all the entity types connected to the relationship. The two primary keys from the entity types make the primary key in the new relation. Or an artificial key will be create to server as the primary key.

### 3.3.7 'IsA' (superclass and subclasses) relationship

A is-a relationship deals with the relation between two entities. The superclass is the parent of the subclass. To convert to a relation, the primary key of the superclass is stored in the subclass as a foreign key.

### 3.3.8 'HasA' relationship

The has-a relationship is when the entity needs to include another entity set as one of it's members. The relation is done by placing a foreign key in the entity set so it can point to a primary key in the other entity set.

### 3.3.9 Relationship type involving other relationship type

If two entities are linked though more then one relationship, you can the role names. For example Instructor and Section entity types are linked through teach. This name clarifies the purpose of the relationship.

### 3.3.10 Recursive relationship(involving one entity types)

The recursive relationship(in the E-R model) involves only one entity that is related to it's self. This relationship is also given a name describing it's relation. The relation will have a foreign key from the primary key in it's own relation.

### 3.3.11 Relationships involving more then 2 entity types

More complex relations in the E-R model involve more then two entities types. To convert the relation to the relational model, you can make one relation the primary relation that contains all the other relations primary keys as foreign keys.

### 3.4 Define Constraints and how DBMS's enforce constraints

Every attribute has an associated domain constraint. These form restrictions on the set of values allowed for the attributes of relations. There are also two important major integrity constraints that apply to all instances on the database. The two main rules of the relational model are known as the entity integrity and referential integrity.<sup>1</sup>

#### 3.4.1 Entity constraint

There are two entity constraints that applies to the primary keys of base relations. The first is the entity integrity rule which says' "in a base relation, no attribute of a primary key can be null." A primary key is minimal identifier that is used to identify tuples uniquely, and no subset of the primary key is enough to provide unique identification of the tuples.<sup>1</sup>

#### 3.4.2 Referential constraints

The second entity constraint is the referential integrity rule which says, "if a foreign key exists in a relation, either the foreign key must match a candidate key value of some tuple in it home relation or the foreign key value must be null". This rule is implemented because if one was to create a tuple in a relation while using the foreign key, and the value of that foreign key does not exist in the home relation, this would cause problems of accountability in the database.<sup>1</sup>

#### 3.4.3 Primary key and unique key constraints

The constraint states that all of the attributes that form a primary key of a relation need to have a value assigned to them that is not null. The primary key is used as a link between two entities and must provide a proper unique value. Also, in the relational model a relation made with a weak entity is given the primary key of the strong entity that is related to.<sup>1</sup>

#### 3.4.4 Check constraints and business rules

The constraints and business rules can be enforced at many levels of the process. It could be handled by the client side application connecting to the database. This is not always a good idea because of performance, unless the rule can not be enforced any other way. Another method would be to place a go between application running on the database server or application server that maintains the rules then replays back to the client application.

Furthermore another method is to store a function to check the business rules in the database management system, these functions are called stored procedures. Finally another method would be, during the insertion additional constraints can be added as a "check" to see if there will be a problem, but this is only good for simple logic. There is not one cure all so you have to asses the rules and apply them at the lowest level to the database management system for best performance.

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

## Section 4: Convert the E-R database into relational database schema

In this section the E-R model design will be converted to a relational model suitable for entry into the database management system using the SQL.

Note: Some of the names may have changed to better suit the naming scheme.

### 4.1 Entity to relation

#### 4.1.1 Entity: **Section** to Relation: **Section**

Note: max\_num\_students was changed to max\_students

Section(section\_id, course\_id, id, crn, season\_num, start\_date, end\_date, max\_students)

Candidate key: section\_id

Field	Domain	Constraint
section_id	integer { 0 – 999999999 }	not null, auto, primary key
course_id	integer { 0 – 999999999 }	not null, foreign key(CourseInfo)
id	integer { 0 – 999999999 }	not null, foreign key(Person)
crn	integer { 0 – 999999999 }	not null
season_num	integer { 0 – 9 }	not null
start_date	date { 1900-01-01 - 9999-31-12 }	not null
end_date	date { 1900-01-01 – 9999-31-12 }	not null, >= start_date
max_students	integer { 1 – 999 }	not null

#### 4.1.2 Entity: **CourseInfo** to Relation: **CourseInfo**

CourseInfo(course\_id, short\_name, long\_name, description, worth )

Candidate key: short\_name or course\_id

Field	Domain	Constraint
course_id	integer { 0 – 999999999 }	not null, auto, primary key
short_name	varchar(32)	not null, unique
long_name	varchar(128)	not null
description	varchar(1024)	null allowed
worth	integer { 0 – 99 }	not null

#### 4.1.3 Entity: **Person** to Relation: **Person**

Person(id, fname, mname, lname, p\_email, p\_phone)

Candidate key: id

Field	Domain	Constraint
id	integer { 0 – 999999999 }	not null, auto, primary key
fname	varchar(32)	not null
mname	varchar(32)	null allowed
lname	varchar(32)	not null
p_email	varchar(128)	null allowed
p_phone	varchar(16)	null allowed

#### 4.1.4 Entity: **Instructor** to Relation: **Instructor**

Instructor(id, password, website, s\_email, s\_phone)

Candidate key: id

Field	Domain	Constraint
id	integer { 0 – 999999999 }	not null, foreign key(Person)
password	varchar(32)	not null
website	varchar(64)	null allowed
s_email	varchar(32)	null allowed
s_phone	varchar(128)	null allowed

#### 4.1.5 Entity: **Student**

This relation was dropped because of redundancy and can be derived. The person relation will be used instead. So by default all persons are students, thus all instructors are students too. This does not violate institutional rules, but an instructor can not teach and signup for the same course. This would need to be checked when inserting in to the database as one of the business rules.

#### 4.1.6 Entity: **Assignment** to Relation: **Assignment**

Note: is\_extra\_credit was changed to is\_extra

Assignment(assign\_id, section\_id, name, type\_num, max\_points, is\_extra, details)

Candidate key: assign\_id

Field	Domain	Constraint
assign_id	integer { 0 – 999999999 }	not null, auto, primary key
section_id	integer { 0 – 999999999 }	not null, foreign key(Section)
name	varchar(64)	not null
type_num	integer { 0 – 99 }	not null
max_points	integer { 0 – 999999999 }	not null
is_extra	boolean	not null
details	varchar(1024)	null allowed

#### 4.1.7 Entity: **Grade** to Relation: **Grade**

Grade(grade\_id, assign\_id, points, date\_entered, details )

Candidate key: grade\_id

Field	Domain	Constraint
grade_id	integer { 0 – 999999999 }	not null, auto, primary key
assign_id	integer { 0 – 999999999 }	not null, foreign key(Assignment)
id	integer { 0 – 999999999 }	not null, foreign key(Person)
points	integer { 0 – 999999999 }	not null
date_entered	date { 1900-01-01 - 9999-31-12 }	not null
details	varchar(1024)	null allowed



#### 4.1.8 Mapping from Student to Section called Signup to Relation: **Signup**

Signup( id, section\_id, grade\_letter, grade\_points, details)

Candidate key: id and section\_id

Field	Domain	Constraint
id	integer { 0 – 999999999 }	not null, foreign key(Person)
section_id	integer { 0 – 999999999 }	not null, foreign key(Section)
grade_letter	varchar(4)	null allowed
grade_points	float	null allowed
details	varchar(1024)	null allowed

## 4.2 Design relation instances

This section has examples(dummy data) of relational instances.

### 4.2.1 Section

section_id	course_id	id	crn	season_num	start_date	end_date	max_students
0	0	318205259	41503	0	09/12/2005	12/02/2005	50
1	1	188687932	41506	0	09/12/2005	12/02/2005	22
2	3	139865559	41505	0	09/12/2005	12/02/2005	26
3	3	167495791	41507	0	09/12/2005	12/02/2005	28
4	4	463882377	41508	0	09/12/2005	12/02/2005	22
5	5	346970481	41509	0	09/12/2005	12/02/2005	22
6	7	167495791	41510	0	09/12/2005	12/02/2005	32
7	8	419425447	41511	0	09/12/2005	12/02/2005	22
8	11	318205259	41512	0	09/12/2005	12/02/2005	22
9	15	139865559	41513	0	09/12/2005	12/02/2005	22
10	17	320362253	41515	0	09/12/2005	12/02/2005	22

### 4.2.2 CourseInfo

course_id	short_name	long_name	description	worth
0	ca150	Introduction to Unix	Basic Unix commands and programming utilities will be introduced. Students will learn how to use email, a text editor, and manage files and directories. This course is designed for students who have no experience with Unix. [F, W, S]	1
1	cs215	Unix Programming Environment	This course covers common Unix commands, shell scripting, regular expressions, tools and the applications used in a Unix programming environment. The tools to be introduced include make utility, a debugger, advanced text editing and text processing (vi, sed, tr). Prerequisite: CMPS 150 or permission of instructor.	5
2	cs216	Unix System Administration	This course covers the knowledge and skills critical to administering a multi-user, networked Unix system. The course assumes a basic knowledge of Unix commands and an editor (vi or Emacs). Topics include: kernel and network configuration, managing daemons, devices, and critical processes, controlling startup and shutdown events, account management, installing software, security issues, shell scripting. Many concepts will be demonstrated during hands-on labs. Prerequisite: CMPS 215.	5
3	cs221	Programming Fundamentals	Introduces the fundamentals of procedural programming. Topics include: data types, control structures, functions, arrays, and standard and file I/O. The mechanics of compiling, linking, running, debugging and testing within a particular programming environment are covered. Ethical issues and a historical perspective of programming within the context of computer science as a discipline is given. [F, W, S]	5

course_id	short_name	long_name	description	worth
4	cs222	Object-Oriented Programming	Builds on foundation provided by CMPS 221 to introduce the concepts of object-oriented programming. The course focuses on the definition and use of classes and the fundamentals of object-oriented design. Other topics include: an overview of programming language principles, basic searching and sorting techniques, and an introduction to software engineering issues. Prerequisite: CMPS 221. [F, W, S]	5
5	cs223	Data Structures and Algorithms	Builds on the foundation provided by the CMPS 221-222 sequence to introduce the fundamental concepts of data structures and the algorithms that proceed from them within the framework of object-oriented programming methodology. Topics include: recursion, fundamental data structures (including stacks, queues, linked lists, hash tables, trees, and graphs), and the basics of algorithmic analysis. Prerequisite: CMPS 222. [F, W, S]	5
6	cs300	Discrete Structures	Elementary logic and set theory, functions and relations, induction and recursion, elementary algorithm analysis, counting techniques, and introduction to computability. Prerequisite: CMPS 223.	5
7	cs312	Algorithm Analysis	Algorithm analysis, asymptotic notation, hashing, hash tables, scatter tables, and AVL and B-trees, brute-force and greedy algorithms, divide-and-conquer algorithms, dynamic programming, randomized algorithms, graphs and graph algorithms, and distributed algorithms. Prerequisite: CMPS 223.	5
8	cs320	Digital Circuits	An introduction to the logical design of digital computers including the analysis and synthesis of combinatorial and sequential circuits, and the use of such circuits in building processor components and memory. The course will apply the circuit theory to the design of an elementary processor with a small instruction set with absolute addressing and a hard-wired control unit. An assembly language for this processor will also be developed. This course includes a laboratory which will cover a mix of actual circuit work together with circuit synthesis and testing using software. Prerequisite: One course in programming or permission of the instructor.	5
9	cs321	Computer Architecture	This course follows the Digital Logic Design course and focuses on the design of the CPU and computer system at the architectural (or functional) level: CPU instruction sets and functional units, data types, control unit design, interrupt handling and DMA, I/O support, memory hierarchy, virtual memory, and buses and bus timing. In contrast, the Digital logic Design course is primarily concerned with implementation; that is, the combinatorial and sequential circuits which are the building blocks of the functional units. Prerequisite: CMPS 223 and 320.	5
10	cs335	Software Engineering	Introduction to the software life cycle. Methods	5

course_id	short_name	long_name	description	worth
			and tools for the analysis, design, and specification of large, complex software systems. Project documentation, organization and control, communication, and time and cost estimates. Group laboratory project. Graphical User Interface Design. Technical presentation methods and practice. Software design case studies and practices. Prerequisite: CMPS 223.	
11	cs342	Database Systems	Basic issues in data modeling, database application software design and implementation. File organizations, relational model, relational database management systems, and query languages are addressed in detail. Two-tier architecture, three-tier architecture and development tools are covered. Prerequisite: CMPS 223.	5
12	cs350	Programming Lang.	An examination of underlying concepts in high level programming languages and techniques for the implementation of a representative sample of such languages with regard to considerations such as typing, block structure, scope, recursion, procedures invocation, context, binding, and modularity. Prerequisite: CMPS 223.	5
13	cs356	Artificial Intelligence	This course is intended to teach the fundamentals of artificial intelligence which include topics such as expert systems, artificial neural networks, fuzzy logic, inductive learning and evolutionary algorithms. Prerequisite: CMPS 223.	5
14	cs360	Operating Systems	A study of the introductory concepts in operating systems: historical development of batch, multiprogrammed, and interactive systems; file, memory, device, process, and thread management; interrupt and trap handlers, abstraction layer, message passing; kernel tasks and kernel design issues; signals and interprocess communication; synchronization, concurrency, and deadlock problems. Prerequisite: CMPS 223.	5
15	cs371	Computer Graphics	Introduction to computer graphics hardware, animation, two-dimensional transformations, basic concepts of computer graphics, theory and implementation. Use of graphics API's such as DirectX or OpenGL. Prerequisite: CMPS 223.	5
16	cs376	Networking	A study of computer networks focusing on the TCP/IP Internet protocols and covering in detail the four layers: physical, data link, network, and transport. This course includes a laboratory in which students will cover important network utilities, debugging tools, process and thread control as it relates to network programming, and the coding of programs which do interprocess communication over sockets. Prerequisite: CMPS 223.	5
17	cs457	Robotics	The course will provide an opportunity for students to understand intelligent robot system architecture and to design algorithms ...	5

#### 4.2.3 Person

id	fname	mname	lname	p_email	p_phone
003651553	Mark	L.	Getty	mgetty@cs.csubak.edu	(661) 945-0331
013766087	Conan	S.	Warashky	cwarashk@cs.csubak.edu	(661) 393-7972
021316065	Marc		Black	mblack@cs.csubak.edu	(661) 492-4758
042840922	William	A.	Sutton	wsutton@cs.csubak.edu	(661) 326-4168
066403651	Chris	E.	Gonzales	cgonzale@cs.csubak.edu	(661) 392-0595
085776546	Eugene	C.	Black	eblack@cs.csubak.edu	(661) 289-2452
089625452	Joseph	B.	Zappa	jzappa@cs.csubak.edu	(661) 348-5098
090141755	Sam		Smith	ssmith@cs.csubak.edu	(661) 860-0016
106575040	Casey		Jobs	cjobs@cs.csubak.edu	(661) 663-5469
125109089	Byran	C.	Warashky	bwarashk@cs.csubak.edu	(661) 409-3357
131493310	Sam		Black	sblack@cs.csubak.edu	(661) 785-1508
133768257	Pat	B.	Kerry	pkerry@cs.csubak.edu	(661) 627-5037
139865559	Casey	E.	Chung	cchung@cs.csubak.edu	(661) 107-4858
141198552	Donna		Johnson	djohnson@cs.csubak.edu	(661) 630-6610
146209587	Jon	A.	Langen	jlangen@cs.csubak.edu	(661) 581-4637
147650213	Maureen	L.	Tran	mtran@cs.csubak.edu	(661) 391-4379
150120557	Sam	L.	Johnson	sjohnson@cs.csubak.edu	(661) 333-1365
165299473	Mark	J.	Johnson	mjohnson@cs.csubak.edu	(661) 368-3724
167326160	Conan	C.	Danforth	cdanfort@cs.csubak.edu	(661) 396-8122
167495791	Chris		Sutton	csutton@cs.csubak.edu	(661) 182-2200
169640153	Brian	A.	Piven	bpiven@cs.csubak.edu	(661) 460-7443
172785946	Maureen	P.	Nevsky	mnevsky@cs.csubak.edu	(661) 620-2720
174940733	Bob	F.	Basheta	bbasheta@cs.csubak.edu	(661) 767-5866
185445705	Brenda	E.	Johnson	bjohnson@cs.csubak.edu	(661) 265-2151
186648582	Mark	S.	Thomas	mthomas@cs.csubak.edu	(661) 391-1997
188687932	Sam	L.	Gates	sgates@cs.csubak.edu	(661) 188-8970
197400921	Melissa	A.	Gates	mgates@cs.csubak.edu	(661) 408-6812
199768032	Scott	C.	Chung	schung@cs.csubak.edu	(661) 804-9483
219225017	Donna	P.	Gates	dgates@cs.csubak.edu	(661) 576-6795
225712527	Nick	E.	Zappa	nzappa@cs.csubak.edu	(661) 883-2867
234813752	Jennifer	F.	Thomas	jthomas@cs.csubak.edu	(661) 340-4698
242530867	Bob	C.	Gates	bgates@cs.csubak.edu	(661) 958-2143
242579947	Bob	P.	Martinez	bmartine@cs.csubak.edu	(661) 994-6330
245815123	Jon	B.	Warashky	jwarashk@cs.csubak.edu	(661) 852-5939
253221481	Lauren	F.	Rodregez	lrodrege@cs.csubak.edu	(661) 426-6137
260955005	George		Meyers	gmeyers@cs.csubak.edu	(661) 965-0212
262329876	Eugene		Tran	etran@cs.csubak.edu	(661) 388-5155
263996508	Linda	F.	Martinez	lmartine@cs.csubak.edu	(661) 867-7947
272741205	Sam	S.	Sutton	ssutton@cs.csubak.edu	(661) 904-2508
289462976	Pat	E.	Rodregez	prodrege@cs.csubak.edu	(661) 612-8684
300328538	Donna		Piven	dpiven@cs.csubak.edu	(661) 815-5125
300867195	Scott		Wang	swang@cs.csubak.edu	(661) 538-4559
313252727	Scott		Piven	spiven@cs.csubak.edu	(661) 852-0862
318205259	Bob	L.	Wang	bwang@cs.csubak.edu	(661) 131-0177
320362253	Bill	C.	Gonzales	bgonzale@cs.csubak.edu	(661) 214-5916
321141053	Casey	P.	Piven	cpiven@cs.csubak.edu	(661) 307-2742
330200714	Jack	L.	Kerry	jkerry@cs.csubak.edu	(661) 398-5131
338566364	Marc	J.	Danforth	mdanfort@cs.csubak.edu	(661) 312-7029
346970481	Luis	B.	Warashky	lwarashk@cs.csubak.edu	(661) 119-8442

id	fname	mname	lname	p_email	p_phone
354641836	Mark	A.	Brian	mbrian@cs.csubak.edu	(661) 263-2773
369219713	Linda	E.	Nevsky	lnevsky@cs.csubak.edu	(661) 903-2843
381837284	Scott		Nevsky	snevsky@cs.csubak.edu	(661) 997-1796
395162265	Jack	F.	Piven	jpiven@cs.csubak.edu	(661) 895-3509
398778602	Eugene	A.	Guteriuz	eguteriu@cs.csubak.edu	(661) 809-6064
412511608	George	S.	Jobs	gjobs@cs.csubak.edu	(661) 157-6179
417420769	Bob	F.	Getty	bgetty@cs.csubak.edu	(661) 580-3951
418911842	Lauren	E.	Black	lblack@cs.csubak.edu	(661) 860-1926
419425447	Sam	F.	Rivers	srivers@cs.csubak.edu	(661) 208-1187
440965598	Melissa	F.	Gonzales	mgonzale@cs.csubak.edu	(661) 714-6265
454422757	Jennifer	S.	Brian	jbrian@cs.csubak.edu	(661) 581-2413
457555131	George	J.	Kerry	gkerry@cs.csubak.edu	(661) 891-8394
459692930	Kris	P.	Warashky	kwarashk@cs.csubak.edu	(661) 538-6041
463882377	Pat	J.	Rivers	privers@cs.csubak.edu	(661) 140-4662
463992699	Casey	L.	Langen	clangen@cs.csubak.edu	(661) 494-0091
465754579	Mark	J.	Chung	mchung@cs.csubak.edu	(661) 556-2538
470389394	Kris	P.	Rivers	krivers@cs.csubak.edu	(661) 881-4836
475493900	Marc		Smith	msmith@cs.csubak.edu	(661) 445-7466
498943316	Chris		Kerry	ckerry@cs.csubak.edu	(661) 388-7371
510627845	Marc	F.	Guteriuz	mguteriu@cs.csubak.edu	(661) 528-6521
517173452	Steve	L.	Meyers	smeyers@cs.csubak.edu	(661) 175-8858

#### 4.2.4 Instructor

id	password	website	s_email	s_phone
139865559	2K79ZwYS	http://www.cs.csubak.edu/~cchung	csutton.csub@gmail.com	(661) 401-6090
167495791	4Zs_L	http://www.cs.csubak.edu/~csutton		(661) 959-4341
188687932	?t1VkytN	http://www.cs.csubak.edu/~sgates		(661) 750-1671
318205259	DXxlmvr	http://www.cs.csubak.edu/~bwang	bill.gonz@hotmail.com	(661) 559-2550
320362253	v9W5q2pD	http://www.cs.csubak.edu/~bgonzale		(661) 572-3429
346970481	O8FRsYdH	http://www.cs.csubak.edu/~lwarashk		
412511608	M/wc=	http://www.cs.csubak.edu/~gjobs	rivercake@aol.com	
419425447	Gv/QJGdI	http://www.cs.csubak.edu/~srivers		
463882377	EsAsjy7	http://www.cs.csubak.edu/~privers		(661) 245-6027
517173452	p~cfbavt	http://www.cs.csubak.edu/~smeyers		(661) 125-5780

#### 4.2.5 Assignment

assign_id	section_id	name	type_num	max_points	is_extra	details
0	3	Lab	0	80	FALSE	question 20 thrown out
1	9	Lab	0	170	FALSE	
2	8	Quiz	1	30	TRUE	
3	8	Final	4	10	FALSE	
4	0	Quiz	1	80	FALSE	
5	5	Test	2	150	FALSE	
6	0	Test	2	60	FALSE	
7	10	Quiz	1	160	FALSE	
8	1	Lab	0	20	TRUE	
9	1	Final	4	110	FALSE	
10	3	Quiz	1	130	FALSE	
11	3	Final	4	50	FALSE	
12	6	Quiz	1	120	FALSE	
13	3	Lab	0	120	FALSE	
14	5	Test	2	70	FALSE	
15	3	Final	4	40	FALSE	
16	10	Final	4	50	FALSE	
17	7	Lab	0	160	FALSE	
18	4	Quiz	1	100	FALSE	
19	5	Lab	0	120	FALSE	
20	10	Mid Term	3	160	FALSE	has bonus problem
21	5	Lab	0	80	TRUE	
22	6	Final	4	100	FALSE	
23	7	Lab	0	70	TRUE	
24	1	Lab	0	80	FALSE	
25	5	Lab	0	100	TRUE	
26	7	Lab	0	100	FALSE	
27	8	Quiz	1	130	FALSE	
28	3	Test	2	30	FALSE	
29	2	Lab	0	80	FALSE	
30	9	Quiz	1	80	TRUE	
31	2	Quiz	1	170	FALSE	
32	2	Quiz	1	100	TRUE	
33	9	Lab	0	140	FALSE	
34	1	Mid Term	3	50	FALSE	
35	9	Lab	0	170	FALSE	
36	5	Quiz	1	20	FALSE	
37	2	Quiz	1	50	FALSE	
38	3	Mid Term	3	20	FALSE	question 12 thrown out
39	1	Mid Term	3	40	FALSE	
40	6	Mid Term	3	130	FALSE	
41	7	Quiz	1	20	FALSE	
42	6	Final	4	120	FALSE	
43	10	Test	2	100	FALSE	
44	1	Test	2	10	FALSE	
45	0	Mid Term	3	140	FALSE	
46	10	Final	4	140	FALSE	
47	2	Quiz	1	60	TRUE	
48	9	Test	2	80	FALSE	

assign_id	section_id	name	type_num	max_points	is_extra	details
49	1	Quiz	1	100	FALSE	question 5 thrown out
50	8	Lab	0	10	FALSE	
51	6	Final	4	80	FALSE	
52	10	Mid Term	3	20	FALSE	
53	5	Lab	0	190	FALSE	
54	2	Lab	0	100	FALSE	
55	6	Quiz	1	120	FALSE	
56	9	Lab	0	110	FALSE	
57	1	Quiz	1	90	FALSE	
58	2	Quiz	1	70	FALSE	
59	0	Quiz	1	110	FALSE	has bonus problem
60	1	Quiz	1	30	FALSE	
61	10	Test	2	130	FALSE	
62	0	Test	2	120	FALSE	
63	8	Lab	0	70	FALSE	
64	1	Lab	0	160	FALSE	
65	4	Lab	0	130	FALSE	
66	1	Quiz	1	140	FALSE	
67	4	Test	2	200	FALSE	



#### 4.2.6 Grade

grade_id	assign_id	id	points	date_entered	details
0	27	253221481	117	10/13/2005	turned in late
1	30	172785946	8	11/3/2005	
2	2	459692930	5	11/2/2005	
3	35	440965598	117	10/15/2005	
4	0	262329876	45	11/2/2005	
5	62	234813752	30	10/1/2005	
6	64	330200714	121	10/3/2005	
7	17	242530867	70	11/2/2005	
8	53	272741205	111	10/17/2005	
9	52	150120557	7	11/30/2005	
10	30	172785946	18	11/3/2005	turned in late
11	0	262329876	65	11/5/2005	
12	22	165299473	82	10/26/2005	
13	12	165299473	13	10/11/2005	
14	5	369219713	119	10/19/2005	
15	29	330200714	42	11/21/2005	
16	40	470389394	93	11/11/2005	
17	26	418911842	40	10/18/2005	
18	12	165299473	42	10/13/2005	
19	17	242530867	30	10/21/2005	
20	2	150120557	20	10/17/2005	
21	25	369219713	85	11/11/2005	
22	45	234813752	56	10/14/2005	
23	66	459692930	67	11/26/2005	
24	5	272741205	48	10/21/2005	
25	12	165299473	41	11/14/2005	
26	27	150120557	15	10/30/2005	
27	11	262329876	0	10/23/2005	
28	27	150120557	47	11/1/2005	
29	40	131493310	59	11/27/2005	
30	24	185445705	75	10/13/2005	
31	41	242530867	12	10/17/2005	
32	45	234813752	104	11/24/2005	
33	40	131493310	85	10/16/2005	
34	38	199768032	10	11/20/2005	
35	27	459692930	125	11/19/2005	
36	59	174940733	24	11/26/2005	
37	51	165299473	5	10/4/2005	
38	5	272741205	103	11/6/2005	
39	10	262329876	109	11/4/2005	
40	5	369219713	84	10/7/2005	turned in late
41	41	418911842	10	11/13/2005	
42	37	330200714	25	11/4/2005	
43	40	470389394	69	10/11/2005	
44	45	234813752	138	10/5/2005	
45	51	165299473	46	11/14/2005	
46	28	262329876	3	11/25/2005	
47	17	459692930	63	10/26/2005	
48	64	459692930	55	10/6/2005	

grade_id	assign_id	id	points	date_entered	details
49	27	459692930	32	11/6/2005	turned in late
50	34	185445705	30	11/8/2005	
51	17	459692930	50	10/19/2005	
52	36	272741205	17	11/10/2005	
53	53	369219713	69	10/6/2005	
54	67	165299473	54	11/11/2005	
55	56	242530867	6	10/11/2005	
56	27	459692930	93	10/28/2005	
57	4	234813752	63	11/22/2005	
58	17	459692930	115	11/16/2005	
59	25	272741205	23	10/29/2005	
60	50	150120557	4	11/20/2005	
61	62	234813752	9	11/15/2005	
62	22	272741205	72	10/3/2005	
63	17	242530867	160	10/15/2005	

#### 4.2.7 Signup

id	section_id	grade_letter	grade_points	details
150120557	8	D	63.4	Incomplete
089625452	0	B-	80.8	
459692930	8	C-	71	
131493310	6	D-	61.5	
459692930	7	B	85.7	
234813752	0			
253221481	8			
242530867	7	C	74.1	
172785946	9			
242530867	9	A	99.3	
369219713	5	C+	78.3	
459692930	1			
418911842	7			
185445705	1			
165299473	6	A-	90.3	Incomplete
440965598	9	C	73.7	
272741205	5			
174940733	0	D-	61.8	
150120557	10			
272741205	6	C	74.7	
165299473	4	C	75.9	
125109089	0	C-	72.7	
262329876	3	F	33.9	
330200714	2			
199768032	3	D	66.1	
470389394	6	B-	81.7	
330200714	1			

---

### 4.3 Non-trivial queries in relational algebra, tuple relational calculus, and domain relational calculus.

**R.A.** - Relational Algebra  
**T.R.C.** - Tuple Relational Calculus  
**D.R.C.** - Domain Relational Calculus

#### 4.3.1 List all student id's who has taken 'cs215'.

a) R.A.

$$\Pi_{id} \left( \text{Person} \bowtie \left[ \Pi_{id} \left( \Pi_{\text{section\_id}} \left( \text{Section} \bowtie \sigma_{\text{short\_name} = \text{'cs215'}}^{\text{(CourseInfo)}} \right) \bowtie \text{Signup} \right) \right] \right)$$

b) T.R.C.

$$\{t^1 \mid (\exists_s)(\text{Person}(s) \wedge (\exists_c)(\text{CourseInfo}(c) \wedge c.\text{short\_name} = \text{'cs215'} \wedge (\exists_{se})(\text{Section}(se) \wedge se.\text{course\_id} = c.\text{course\_id} \wedge (\exists_{si})(\text{Signup}(si) \wedge s.\text{id} = si.\text{id} \wedge t.1 = s.\text{id} \wedge se.\text{section\_id} = si.\text{section\_id} ))))\}$$

c) D.R.C.

$$\{ \langle id \rangle \mid \text{Person}(id, -, -, -, -, -) \wedge (\exists_c)(\text{CourseInfo}(c, \text{'cs215'}, -, -, -) \wedge (\exists_{se})(\text{Section}(se, c, -, -, -, -, -) \wedge \text{Signup}(id, se, -, -, -)) \}$$

#### 4.3.2 List all instructors who taught 'cs221'.

a) R.A.

$$\text{Person} \bowtie \left[ \Pi_{id} \left( \text{Section} \bowtie \sigma_{\text{short\_name} = \text{'cs221'}}^{\text{(CourseInfo)}} \right) \bowtie \text{Instructor} \right]$$

b) T.R.C.

$$\{t^{10} \mid (\exists_i)(\exists_p)(\text{Instructor}(i) \wedge \text{Person}(p) \wedge t.1 = p.\text{id} \wedge t.2 = p.\text{fname} \wedge t.3 = p.\text{mname} \wedge t.4 = p.\text{lname} \wedge t.5 = p.\text{p\_email} \wedge t.6 = p.\text{p\_phone} \wedge t.7 = i.\text{password} \wedge t.8 = i.\text{website} \wedge t.9 = i.\text{s\_email} \wedge t.10 = i.\text{s\_phone} \wedge (\exists_c)(\text{CourseInfo}(c) \wedge c.\text{short\_name} = \text{'cs221'} \wedge (\exists_{se})(\text{Section}(se) \wedge se.\text{course\_id} = c.\text{course\_id} \wedge se.\text{id} = i.\text{id} \wedge ))))\}$$

c) D.R.C.

$$\{ \langle id, f, m, l, pe, pp, p, w, se, sp \rangle \mid \text{Person}(id, f, m, l, pe, pp) \wedge \text{Instructor}(id, p, w, se, sp) \wedge (\exists_c)(\text{CourseInfo}(c, \text{'cs221'}, -, -, -) \wedge (\exists_{se})(\text{Section}(se, c, id, -, -, -, -)) \}$$

---

4.3.3 List all course short names, section crn, start date, end date, and worth for  
Fall(season\_num=0) 2005.

a) R.A.

$$\prod_{\text{short\_name, course\_id}} (\text{CourseInfo}) \bowtie \left( \prod_{\text{id, crn, start\_date, end\_date, worth}} \left( \sigma_{\text{season\_num} = 0 \wedge \text{start\_date} \geq '1/1/2005' \wedge \text{end\_date} \leq '12/31/2005'} \right) (\text{Section}) \right)$$

b) T.R.C.

$$\{t^5 \mid (\exists c)(\exists s)(\text{CourseInfo}(c) \wedge \text{Section}(s) \wedge \\ t.1 = c.\text{short\_name} \wedge t.2 = s.\text{crn} \wedge t.3 = s.\text{start\_date} \wedge \\ t.4 = s.\text{end\_date} \wedge t.5 = c.\text{worth} \wedge \\ s.\text{course\_id} = c.\text{course\_id} \wedge s.\text{season\_num} = 0 \wedge \\ s.\text{start\_date} \geq '1/1/2005' \wedge s.\text{end\_date} \leq '12/31/2005')\}$$

c) D.R.C.

$$\{ \langle n, \text{crn}, \text{sd}, \text{ed}, w \rangle \mid (\exists c)(\text{CourseInfo}(c, n, -, -, w) \wedge \\ (\exists se)(\text{Section}(se, c, 0, \text{crn}, \text{sd}, \text{ed}, -, -) \wedge \\ \text{sd} \geq '1/1/2005' \wedge \text{ed} \leq '12/31/2005')) \}$$

4.3.4 List all grade letters, grade points for student 165299473 in  
Winter(season\_num=1) 2004.

a) R.A.

$$\prod_{\text{section\_id, grade\_letter, grade\_points}} \left( \text{Signup} \bowtie \sigma_{\text{id} = 165299473}^{(\text{Person})} \right) \bowtie \left( \prod_{\text{section\_id}} \left( \sigma_{\text{season\_num} = 1 \wedge \text{start\_date} \geq '1/1/2004' \wedge \text{end\_date} \leq '12/31/2004'} \right) (\text{Section}) \right)$$

b) T.R.C.

$$\{t^2 \mid (\exists st)(\exists si)(\exists se)(\text{Person}(st) \wedge \text{Signup}(si) \wedge \text{Section}(se) \wedge \\ t.1 = si.\text{grade\_letter} \wedge t.2 = si.\text{grade\_point} \wedge \\ st.\text{id} = si.\text{id} \wedge si.\text{id} = 165299473 \wedge si.\text{section\_id} = se.\text{section\_id} \wedge \\ se.\text{season\_num} = 1 \wedge se.\text{start\_date} \geq '1/1/2004' \wedge \\ se.\text{end\_date} \leq '12/31/2004')\}$$

c) D.R.C.

$$\{ \langle gl, gp \rangle \mid (\exists st)(\exists se)(\text{Person}(st, -, -, -, -, -) \wedge \\ \text{Section}(se, -, -, 1, \geq '1/1/2004', \leq '12/31/2004', -) \wedge \\ \text{Signup}(st, se, gl, gp, -)) \}$$

---

4.3.5 List student's id, first & last name, average, min, max final grade points between 2004 and 2005.

a) R.A.

$$\begin{aligned} & \text{id} \begin{smallmatrix} \mathfrak{S} \\ \text{avg(grade_points), min(grade_points), max(grade_points)} \end{smallmatrix} \\ & ( \\ & \quad \left( \prod_{\text{id, fname, lname}} (\text{Person}) \right) \bowtie \\ & \quad \left( \text{Signup} \bowtie \left( \sigma_{\text{start\_date} \geq '1/1/2004' \wedge \text{end\_date} \leq '12/31/2005'} (\text{Section}) \right) \right) \\ & ) \end{aligned}$$

b) T.R.C.

Not possible because of aggregate function.

c) D.R.C.

Not possible because of aggregate function.

4.3.6 List all students signed up for 'cs221' in Fall(season\_num=0) 2005, taught by 'Chung'(last name).

a) R.A.

$$\begin{aligned} & ( \\ & \quad \prod_{\text{section\_id, course\_id}} ( \\ & \quad \quad \sigma_{\text{lname} = 'Chung'} (\text{Person}) \\ & \quad \quad \sigma_{\text{season\_num} = 0 \wedge \text{start\_date} \geq '1/1/2005' \wedge \text{end\_date} \leq '12/31/2005'} (\text{Section}) \\ & \quad ) \bowtie \\ & \quad \sigma_{\text{short\_name} = 'cs221'} (\text{CourseInfo}) \\ & ) \bowtie \text{Signup} \end{aligned}$$

b) T.R.C.

$$\begin{aligned} & \{t^1 \mid (\exists p)(\exists_{si})(\exists_{se})(\exists c)( \text{Person}(p) \wedge \text{Signup}(si) \wedge \text{Section}(se) \wedge \\ & \quad \text{CourseInfo}(c) \wedge t.1 = si.id \wedge si.section\_id = se.section\_id \wedge \\ & \quad p.id = se.id \wedge p.lname = 'Chung' \wedge se.course\_id = c.course\_id \wedge \\ & \quad c.short\_name = 'cs221' \wedge se.season\_num = 0 \wedge \\ & \quad se.start\_date \geq '1/1/2005' \wedge se.end\_date \leq '12/31/2005' \wedge ) \} \end{aligned}$$

c) D.R.C.

$$\begin{aligned} & \{ \langle s \rangle \mid (\exists p)(\exists_{se})(\exists c)( \text{CourseInfo}(c, 'cs221', -, -, -, -) \wedge \\ & \quad \text{Person}(p, -, -, 'Chung', -, -) \wedge \\ & \quad \text{Section}(se, c, p, 0, \geq '1/1/2005', \leq '12/31/2005', -) \wedge \\ & \quad \text{Signup}(s, se, -, -, -) ) \} \end{aligned}$$

---

4.3.7 List all assignments for 'cs223' in Spring(season\_num=2) 2006,  
taught by instructor 346970481.

a) R.A.

$$C \leftarrow \sigma_{\text{short\_name} = \text{'cs223'}}(\text{CourseInfo})$$

$$S \leftarrow \sigma_{\text{id} = 346970481 \wedge \text{season\_num} = 2 \wedge \text{start\_date} \geq \text{'1/1/2006'} \wedge \text{end\_date} \leq \text{'12/31/2006'}}(\text{Section})$$

$$\text{Assignment} \bowtie \prod_{\text{section\_id}} (C \bowtie S)$$

b) T.R.C.

$$\{t^5 \mid (\exists a)(\exists se)(\exists c)(\text{Assignment}(a) \wedge \text{Section}(se) \wedge \text{CourseInfo}(c) \wedge \\ t.1 = a.name \wedge t.2 = a.max\_points \wedge t.3 = a.is\_extra \wedge \\ t.4 = a.details \wedge t.5 = a.type\_num \wedge se.id = 346970481 \wedge \\ se.course\_id = c.course\_id \wedge c.short\_name = \text{'cs223'} \wedge \\ a.section\_id = se.section\_id \wedge se.season\_num = 2 \wedge \\ se.start\_date \geq \text{'1/1/2005'} \wedge se.end\_date \leq \text{'12/31/2005'} )\}$$

c) D.R.C.

$$\{ \langle an, amp, ae, ad, at \rangle \mid (\exists se)(\exists c)(\text{CourseInfo}(c, \text{'cs223'}, -, -, -, -) \wedge \\ \text{Section}(se, c, 346970481, 2, \geq \text{'1/1/2005'}, \leq \text{'12/31/2005'}, -) \wedge \\ \text{Assignment}(-, se, an, at, amp, ae, ad)) \}$$

---

4.3.8 List assignment name, points student got, max points, and is extra for student 369219713 in 'cs350' Fall(season\_num=0) 2005, taught by instructor 346970481.

a) R.A.

$$C \leftarrow \sigma_{\text{short\_name} = \text{'cs350'}}(\text{CourseInfo})$$

$$S \leftarrow \sigma_{\text{id} = 346970481 \wedge \text{season\_num} = 0 \wedge \text{start\_date} \geq \text{'1/1/2005'} \wedge \text{end\_date} \leq \text{'12/31/2005'}}(\text{Section})$$

$$A \leftarrow \prod_{\text{assign\_id}, \text{name}, \text{max\_points}, \text{is\_extra}} (\text{Assignment} \bowtie (C \bowtie S))$$

$$\sigma_{\text{id} = 369219713}(\text{Grade}) \bowtie A$$

b) T.R.C.

$$\{t^4 \mid (\exists a)(\exists se)(\exists c)(\exists g)(\text{Grade}(g) \wedge \text{Assignment}(a) \wedge \text{Section}(se) \wedge \text{CourseInfo}(c) \wedge \\ t.1 = a.\text{name} \wedge t.2 = g.\text{points} \wedge t.3 = a.\text{max\_points} \wedge \\ t.4 = a.\text{is\_extra} \wedge c.\text{short\_name} = \text{'cs350'} \wedge se.\text{season\_num} = 0 \wedge \\ se.\text{start\_date} \geq \text{'1/1/2005'} \wedge se.\text{end\_date} \leq \text{'12/31/2005'} \wedge \\ se.\text{id} = 346970481 \wedge se.\text{course\_id} = c.\text{course\_id} \wedge \\ g.\text{assign\_id} = a.\text{assign\_id} \wedge a.\text{section\_id} = se.\text{section\_id} \wedge \\ g.\text{id} = 369219713 )\}$$

c) D.R.C.

$$\{ \langle an, p, mp, e \rangle \mid (\exists se)(\exists c)(\exists a)(\text{CourseInfo}(c, \text{'cs350'}, -, -, -, -) \wedge \\ \text{Section}(se, c, 346970481, 0, \geq \text{'1/1/2005'}, \leq \text{'12/31/2005'}, -) \wedge \\ \text{Assignment}(a, se, an, -, mp, e, -) \wedge \\ \text{Grade}(-, a, 369219713, p, -, -) ) \}$$

---

4.3.9 List the min, max, and average points for all students in  
'cs457' Fall(season\_num=0) 2005, taught by  
instructor 320362253, on assignment 'test'

a) R.A.

$$C \leftarrow \sigma_{\text{short\_name} = \text{'cs457'}}(\text{CourseInfo})$$

$$S \leftarrow \sigma_{\text{id} = 320362253 \wedge \text{season\_num} = 0 \wedge \text{start\_date} \geq \text{'1/1/2005'} \wedge \text{end\_date} \leq \text{'12/31/2005'}}(\text{Section})$$

$$A \leftarrow \prod_{\text{assign\_id}} \left( \sigma_{\text{name} = \text{'test'}}(\text{Assignment}) \bowtie (C \bowtie S) \right)$$

$$G \leftarrow \prod_{\text{id}, \text{points}} (\text{Person} \bowtie (\text{Grade} \bowtie A))$$

$$\rho(\text{MinPoints}, \text{MaxPoints}, \text{AvgPoints}) \left( \text{id}^{\mathfrak{S}^{(G)}_{\text{min}(\text{points}), \text{max}(\text{points}), \text{avg}(\text{points})}} \right)$$

b) T.R.C.

Not possible because of aggregate function.

c) D.R.C.

Not possible because of aggregate function.

4.3.10 Students that have taken all the courses.

a) R.A.

$$S \leftarrow \prod_{\text{id}, \text{course\_id}} (\text{Signup} \bowtie \text{Section})$$

$$S \div \prod_{\text{course\_id}} (\text{CourseInfo})$$

b) T.R.C.

$$\{t^1 \mid (\forall c)(\text{CourseInfo}(c) \rightarrow (\exists se)(\exists si)(\text{Section}(se) \wedge \text{Signup}(si) \wedge t.1 = si.id \wedge se.section\_id = si.section\_id \wedge se.course\_id = c.course\_id))\}$$

c) D.R.C.

$$\{< id > \mid (\forall c)(\text{CourseInfo}(c, -, -, -, -, -, -) \rightarrow (\exists se)(\text{Section}(se, c, -, -, -, -, -) \wedge \text{Signup}(id, se, -, -, -))\}$$



## Section 5: SQL\*PLUS

In this section I will be describing the SQL, SQL\*Plus and its functionality. This includes, SQL\*Plus, The Oracle schema objects, schema objects that were created in this project, relation schemes, sample SQL queries, and finally loading data methods.

### 5.1 SQL\*PLUS

SQL\*PLUS is a client interface providing a command line interface. Which allows users to connect to a remotely or locally to the Oracle server. Once connected and authenticated, you can execute SQL statements(more details below), plus extra function specific to Oracle.

The main purpose of the SQL\*PLUS is to manage your set of databases and set or tables. With the set of SQL commands, like CREATE, ALTER, INSERT, UPDATE, SELECT, DROP.... Each command has a set of allowable options to add more flexibility. SQL\*PLUS added additional advanced functions(the plus part). Allowing you to load SQL statements from a file and save ran SQL statements to a file. Another useful feature, is the commit and rollback which you can use to undo and update database data changes. Most database management system's allow for a developer to create a client application, in an array of languages(C++, C#, Java,...), to send SQL\*PLUS statements to the server remotely(on a particular port) and/or locally. The client application uses a driver or links in a library proved by the DBMS manufacture, which implements connection and transition to the DBMS. Client applications are useful because it can hide the DBMS from the user completely, you can also make the application very easy to use and further enforce the business rules.

A typical SQL\*PLUS session look like:

```
SQL*Plus: Release 8.1.5.0.0 - Production on Thu Nov 2 14:35:50 2006

(c) Copyright 1999 Oracle Corporation. All rights reserved.

Connected to:
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

SQL> @update_all
SQL> @q4

Grade Percent Course
-----
A-          90 cs312
C           76 cs222

SQL>
```

## 5.2 Oracle Schema Objects

Oracle allows the following schema objects: tables, views, objects, sequences, stored procedures & functions, indexes, synonyms, database links, clusters, packages, and triggers.

### 5.2.1 Tables

Tables(in Oracle) are generic data storage objects with columns & rows defined by a user. Each column(attribute) is defined with a data type and all data in that column must be of that type. Oracle has only a few basic types that all other types are derived from, they are a number, string, date and binary object. Each row(tuple) is an inserted record. Tables are used to store the actual data, in most cases it related to another table through a foreign key, but this is not required. It could just be a table of unrelated data.

In SQL you use the CREATE command to create/define a table with a given name, and ALTER to change an existing table's definition. The ALTER command can be complicated and cause further problems, so it's advised to use a view(see view below) and create a new table instead.

Creating a table syntax:

```
CREATE TABLE tableName
  {(columnName dataType [NOT NULL] [UNIQUE]
    [DEFAULT defaultOption]
    [CHECK (condition)] [,...]}
  [PRIMARY KEY (listOfColumns),]
  {[UNIQUE (listOfColumns)] [,...]}
  {[FOREIGN KEY (listOfColumns)
  REFERENCES ParentTableName[(listOfForeignKeys)]
    [MATCH {PARTIAL|FULL}
    [ON UPDATE action]
    [ON DELETE action]] [,...]}
  {[CHECK (condition)] [,...]}))
```

Example of creating a table:

```
CREATE TABLE js_section (
  section_id    INTEGER      NOT NULL PRIMARY KEY,
  course_id     INTEGER      NOT NULL,
  id            INTEGER      NOT NULL,
  crn           INTEGER      NOT NULL,
  season_num    INTEGER      NOT NULL,
  start_date    DATE          NOT NULL,
  end_date      DATE          NOT NULL,
  max_students  INTEGER      NOT NULL,
  FOREIGN KEY(course_id) REFERENCES js_course_info(course_id),
  FOREIGN KEY(id) REFERENCES js_person(id)
);
```

### 5.2.2 Views

A view can be thought of as a virtual table, where the user is not aware that it's not a real table. In actuality a combination of one or more attributes from one or more tables. Views are defined as a query, they can be as simple or complex as they need to be. It can provide a level of security and/or selective view of other tables in the database.

Creating a view syntax:

```
CREATE VIEW viewName [(columnName [,...])] AS
    SELECT ...
[WITH [CASCADED|LOCAL] CHECK OPTION]
```

Example of creating a view:

```
CREATE VIEW js_student_signup AS
    SELECT p.*, se.season_num, se.start_date, si.grade_letter
    FROM js_person p, js_signup si, js_section se
    WHERE si.id = p.id AND
           si.section_id = se.section_id;
```

### 5.2.3 Objects

An Object allows you to create a new custom data type that can be used as a column in a table. Oracle only supports four basic types: strings(fixed length and variable length), numbers(integer and float), dates(include the time), and binary blobs. There are lots of reasons you would like to create a new custom data type(aka object). If you wanted to store an IP address for instance, or an address. These can easily be implemented other ways but an object make it much easier and more controllable.

Creating an object syntax:

```
CREATE [OR REPLACE] TYPE typeName AS OBJECT
    attributeName dataType, ...,
    MEMBER PROCEDURE | FUNCTION
        <procedure or function spec>, ...,
    [MAP|ORDER MEMBER FUNCTION
        <comparison function spec>, ... ]
    [PRAGMA RESTRICT_REFERENCES (
        <what to restrict>, restrictions)];
```

### Example of creating and using an object:

```
-- Create a new typed called "Pet_t"
CREATE TYPE Pet_t AS OBJECT (
    tag_no INTEGER,
    name VARCHAR2(60),
    MEMBER FUNCTION set_tag_no (new_tag_no IN INTEGER)
        RETURN Pet_t
);
/

-- Create a new body for type called "Pet_t"
CREATE TYPE BODY Pet_t
AS
    MEMBER FUNCTION set_tag_no (new_tag_no IN INTEGER)
        RETURN Pet_t
    IS
        the_pet Pet_t := SELF; -- initialize to "current" object
    BEGIN
        the_pet.tag_no := new_tag_no;
        RETURN the_pet;
    END;
END;
/

-- Create table with the new type "Pet_t"
CREATE TABLE pets OF Pet_t;

-- Now we can create an object (object instance) of type "Pet_t"
INSERT INTO pets VALUES (Pet_t(23052, 'Mambo'));
```

([http://www.unix.org.ua/orelly/oracle/prog2/ch18\\_01.htm](http://www.unix.org.ua/orelly/oracle/prog2/ch18_01.htm))

#### 5.2.4 Sequences

A sequence is a automatically generated sequence of unique numbers based on an increment value, start and end range.

Creating an sequence syntax:

```
CREATE SEQUENCE seqName
  [INCREMENT increment]
  [MINVALUE minVal]
  [MAXVALUE maxVal]
  [START WITH start]
  [CACHE cache]
  [CYCLE]
```

Example of creating and using a sequence:

```
-- Create my sequence
CREATE SEQUENCE my_seq
  MINVALUE 0
  MAXVALUE 999999999
  INCREMENT BY 1
  CACHE 20;

-- Create my table
CREATE TABLE my_table(
  mid INTEGER
);

-- Insert into my table
INSERT INTO my_table VALUES(my_seq.nextval);
```

#### 5.2.5 Stored Procedures & Functions

Stored Procedures & Functions are defined in Oracle by a language called PL/SQL. It's a mixture of a simple programming language and SQL statements. Procedures are identical in all ways except one very important fact. Procedures can't return a value but functions can.

Creating a function syntax:

```
CREATE [OR REPLACE] FUNCTION
  name ([[argMode] [argName] argType [,...]])
  [RETURNS retType]
  BEGIN
    PL/SQL CODE...
  END
```

### Creating a stored procedure syntax:

```
CREATE [OR REPLACE] PROCEDURE
  name ([[argMode] [argName] argType [,...]])
BEGIN
  PL/SQL CODE...
END
```

### Example of creating a function:

```
CREATE OR REPLACE FUNCTION isLeapYear(i_year NUMBER) RETURN
boolean AS
BEGIN
  -- A year is a leap year if it is evenly divisible by 4
  -- but not if it's evenly divisible by 100
  -- unless it's also evenly divisible by 400

  IF mod(i_year, 400) = 0 OR
    ( mod(i_year, 4) = 0 AND
      mod(i_year, 100) != 0) THEN
    return TRUE;
  ELSE
    return FALSE;
  END IF;
END
/
```

(<http://www.orafaq.com/scripts/plsql/leapyear.txt>)

### Example of a stored procedure:

```
CREATE PROCEDURE remove_emp (employee_id NUMBER) AS
tot_emps NUMBER;
BEGIN
  DELETE FROM employees
  WHERE employees.employee_id = remove_emp.employee_id;
  tot_emps := tot_emps - 1;
END
/
```

([http://lc.leidenuniv.nl/awcourse/oracle/server.920/a96540/statements\\_610a.htm](http://lc.leidenuniv.nl/awcourse/oracle/server.920/a96540/statements_610a.htm))

### 5.2.6 Indexes

An index is a structure that provides accelerated access to the rows of a table based on the values in one or more columns. Oracle supports index only tables, where the data and index are stored together. An index in the database is similar to an index in a book. Index are used in a book, to quickly find all the locates(pages) a key word can be found. Similarly in a database an index does something very similar, it can significantly improve the performance of a query. But there is a downside, an index may be updated by the system ever time the underlying tables are updated, thus an additional overhead may be incurred. Indexes are usually created to satisfy particular search criteria after the table has been us use for some time and has grown in size. The creation of indexes is not standard SQL. However, most dialects at least support it<sup>1</sup>

The specific columns constitute the index key and should be listed in major to minor order. Indexes can be created only on base tables not on views. If the UNIQUE clause is used, uniqueness of the indexed column or combination of columns will be enforced by as well (for example, alternate keys). Although indexes can be created at any time, we may have a problem if we try to create a unique index on a table with records in it, because the value stored for the indexed column(s) may already contain duplicates. Therefore, it is good practice to create unique indexes, at least for primary key columns, when the base table is created and the DBMS does not automatically enforce primary key uniqueness.<sup>1</sup>

Creating a index syntax:

```
CREATE [UNIQUE] INDEX indexName  
ON tableName (columnName [ASC|DESC] [, ...])
```

Example of creating a index:

```
CREATE INDEX js_person_lname_idx  
ON js_person (lname ASC);
```

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

### 5.2.7 Synonyms

A synonym is alternative name for a object in the database. This is similar to an alias in the sh language.

Creating a synonym syntax:

```
CREATE SYNONYM fakeName
FOR realName
```

Example of creating and using a synonym:

```
-- Create table
CREATE TABLE my_really_long_table_name(
    data NUMBER;
);

-- Create Synonym "myt"
CREATE SYNONYM myt FOR my_really_long_table_name;

-- Example of Use Synonym "myt" instead of
-- "my_really_long_table_name"
-- Inserting some data
insert into myt values(4);
insert into myt values(5);

-- Updating data in "my_really_long_table_name"
update myt set data = 40 where data = 4;

-- Deleting record from "my_really_long_table_name"
delete from myt where a = 5;
```

(<http://www.adp-gmbh.ch/ora/sql/synonyms.html>)

### 5.2.8 Clusters

A cluster is a set of tables physically stored together as one table that shares common columns. If data in two or more tables are frequently retrieved together based on data in the common column, using a cluster can be quite efficient. Tables can be accessed separately even though they are part of a cluster. Because of the structure of the cluster, related data requires much less input/output (I/O) overhead if accessed simultaneously. Oracle supports two types of cluster: indexed clusters and hash clusters. An index cluster is a grouping of tables that share the same cluster key. Hash clusters also cluster table data in a manner similar to index cluster. However record is stored in a hash cluster based on the result of applying a hash function to the record's cluster key value. All records with the same hash key value are stored together on disk.<sup>1</sup>

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition



### 5.2.9 Packages

A package is a collection of procedures, functions, variables, and SQL statements that are grouped together and stored as a single program unit. A package has two parts: a specification and a body. A package specification declares all public constructs of the package, and the body defines all constructs of the package, and so implements the specification. By doing this, packages provide a form of encapsulation.<sup>1</sup>

Example of creating a package:

```
-- Create a package
CREATE OR REPLACE PACKAGE studentClassPackage AS
    PROCEDURE classesForStudents (vStudentNo VARCHAR2);
END studentClassPackage;

-- Creating body for package
CREATE OR REPLACE PACKAGE BODY studentClassPackage AS
    ...
END studentClassPackage;
```

### 5.2.10 Triggers

Triggers are code stored in the database and invoked (triggered) by events that occur in the database. A trigger may be used to enforce some referential integrity constraints, to enforce complex enterprise constraints, or to audit changes to data. The code within a trigger, called the trigger body, is made up of a PL/SQL block, Java program, or 'C' callout.<sup>1</sup> Triggers are based on the Event-Condition-Action (ECA) model:

1. The event (or events) that trigger the rule. In Oracle, they are:
  - a) INSERT, UPDATE, or DELETE statement on a specified table.
  - b) CREATE, ALTER, or DROP statement on any schema object.
  - c) Database startup or instance shutdown, or a user logon or logoff.
  - d) Specific error message or any error message.<sup>1</sup>
2. The condition that determines whether the actions should be executed. The condition is optional but, if specified, the action will be executed only if the condition is true.<sup>1</sup>
3. The action to be taken. The block contains the SQL statements and code to be executed when a triggering statement is issued and the trigger condition turns out to be true.<sup>1</sup>

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

### Creating a trigger syntax:

```
CREATE TRIGGER name {BEFORE|AFTER} {event [OR ...]}  
  ON table [ FOR [EACH] {ROW|STATEMENT} ]  
  EXECUTE PROCEDURE funcName ( arguments )
```

### Example of creating a trigger:

```
-- Creates a trigger that would enforce the constraint that  
-- a student only be allowed to take 4 classes in one semester.  
-- AKA, Students not taking too many classes.  
CREATE OR REPLACE TRIGGER js_Student_Limit  
  BEFORE INSERT OR UPDATE ON js_signup  
  REFERENCING NEW AS newRow  
  FOR EACH ROW  
  DECLARE  
    vpCount NUMBER;  
  BEGIN  
    SELECT COUNT(*) INTO vpCount  
    FROM js_signup si, js_section se  
    WHERE  
      si.section_id = se.section_id AND  
      se.start_date >= SYSDATE AND  
      se.end_date <= SYSDATE AND  
      si.id = :newRow.id;  
  
    IF (vpCount > 4) THEN  
      RAISE_APPLICATION_ERROR(-20000,  
        ('Student ' || :newRow.id ||  
         ' can only take 4 classes')  
      );  
    END IF;  
  END js_Student_Limit;  
/
```

### 5.3 Schema objects that were created in this project

This section describes the method for creating(in SQL), all the tables used in this project.

#### 5.3.1 Course Information: Table & Sequence

This tables holds general information about the courses.

Including: the course ID, a short name, a long name, description, and credits/units worth.

```
-- Course Info. Sequence
CREATE SEQUENCE js_ci_seq
MINVALUE 0
MAXVALUE 999999999
START WITH 100;

-- Course Info. Table
CREATE TABLE js_course_info (
  course_id    INTEGER          NOT NULL PRIMARY KEY,
  short_name   VARCHAR2(32)     NOT NULL,
  long_name    VARCHAR2(128),
  description  VARCHAR2(1024),
  worth        NUMBER(2)        NOT NULL
);
```

#### 5.3.2 Person: Table & Sequence

This tables holds general information about the persons(students or instructors).

Including: the student/instructor ID, first name, middle name, last name, primary email, and primary phone.

```
-- Person Sequence
CREATE SEQUENCE js_p_seq
MINVALUE 0
MAXVALUE 999999999
START WITH 100;

-- Person Table
CREATE TABLE js_person (
  id           INTEGER          NOT NULL PRIMARY KEY,
  fname       VARCHAR2(64)     NOT NULL,
  mname       VARCHAR2(64)     ,
  lname       VARCHAR2(64)     NOT NULL,
  p_email     VARCHAR2(128)    ,
  p_phone     VARCHAR2(16)
);
```

### 5.3.3 Instructor: Table & Sequence

This tables holds more information about the instructors.

Including: the login password, website, secondary email, and secondary phone.

```
-- Instructor Sequence
CREATE SEQUENCE js_i_seq
MINVALUE 0
MAXVALUE 999999999
START WITH 100;

-- Instructor Table
CREATE TABLE js_instructor (
  id          INTEGER      NOT NULL,
  password    VARCHAR2(32) NOT NULL,
  website     VARCHAR2(256) ,
  s_email     VARCHAR2(128) ,
  s_phone     VARCHAR2(16)  ,
  FOREIGN KEY(id) REFERENCES js_person(id)
);
```

### 5.3.4 Section: Table & Sequence

This tables holds information about the sections.

Including: the CRN, season number, starting date, ending date, and maximum allowed students.

```
-- Section Sequence
CREATE SEQUENCE js_se_seq
MINVALUE 0
MAXVALUE 999999999
START WITH 100;

-- Section Table
CREATE TABLE js_section (
  section_id  INTEGER      NOT NULL PRIMARY KEY,
  course_id   INTEGER      NOT NULL,
  id          INTEGER      NOT NULL,
  crn         INTEGER      NOT NULL,
  season_num  NUMBER(1)    NOT NULL,
  start_date  DATE         NOT NULL,
  end_date    DATE         NOT NULL,
  max_students NUMBER(3)   NOT NULL,
  FOREIGN KEY(course_id) REFERENCES js_course_info(course_id),
  FOREIGN KEY(id) REFERENCES js_person(id)
);
```

### 5.3.5 Sign-up: Table & Sequence

This tables holds information about the sign-up.

Including: the final letter grade, the final grade points(could be percentage), and details.

```
-- Signup Sequence
CREATE SEQUENCE js_si_seq
MINVALUE 0
MAXVALUE 999999999
START WITH 100;

-- Signup Table
CREATE TABLE js_signup (
  id            INTEGER      NOT NULL,
  section_id    INTEGER      NOT NULL,
  grade_letter  VARCHAR2(4)  ,
  grade_points  FLOAT        ,
  details       VARCHAR2(1024),
  FOREIGN KEY(id) REFERENCES js_person(id),
  FOREIGN KEY(section_id) REFERENCES js_section(section_id)
);
```

### 5.3.6 Assignment: Table & Sequence

This tables holds information about the assignments.

Including: the name(title), type number(for grouping), max points, whether it's extra credit or not, and details.

```
-- Assignment Sequence
CREATE SEQUENCE js_a_seq
MINVALUE 0
MAXVALUE 999999999
START WITH 100;

-- Assignment Table
CREATE TABLE js_assign (
  assign_id    INTEGER      NOT NULL PRIMARY KEY,
  section_id    INTEGER      NOT NULL,
  name         VARCHAR2(64)  NOT NULL,
  type_num     NUMBER(2)     NOT NULL,
  max_points    INTEGER      NOT NULL,
  is_extra     NUMBER(1)     NOT NULL,
  details       VARCHAR2(1024) ,
  FOREIGN KEY(section_id) REFERENCES js_section(section_id)
);
```

### 5.3.7 Grade: Table & Sequence

This tables holds information about the grades.

Including: the points given to an assignment for a student, the date entered, and details.

```
-- Grade Sequence
CREATE SEQUENCE js_g_seq
MINVALUE 0
MAXVALUE 999999999
START WITH 100;

-- Grade Table
CREATE TABLE js_grade (
  grade_id      INTEGER      NOT NULL PRIMARY KEY,
  assign_id     INTEGER      NOT NULL,
  id            INTEGER      NOT NULL,
  points        INTEGER      NOT NULL,
  date_entered  DATE         NOT NULL,
  details       VARCHAR2(1024),
  FOREIGN KEY(assign_id) REFERENCES js_assign(assign_id),
  FOREIGN KEY(id)      REFERENCES js_person(id)
);
```

## 5.4 Relation Schemes

This section is a list of relations designed for the Instructor Assistant Grading System(IAGS). For each relation, I have listed its relation schema and its contents (relation instance), using DESC and SELECT commands to show the relation schema and the SELECT statement to show the contents of the relation.

### 5.4.1 js\_course\_info

This relation holds general information about the courses.

Including: the course ID, a short name, a long name, description, and credits/units worth.

Relation Schema:

```
SQL> DESC js_course_info;
```

Name	Null?	Type
COURSE_ID	NOT NULL	NUMBER(38)
SHORT_NAME	NOT NULL	VARCHAR2(32)
LONG_NAME		VARCHAR2(128)
DESCRIPTION		VARCHAR2(1024)
WORTH	NOT NULL	NUMBER(2)

Relation Instances:

```
SQL> SELECT * FROM js_course_info;
```

COURSE_ID	SHORT_NAME	LONG_NAME	DESCRIPTION	WORTH
0	cs150	Introduction to Unix	Basic Unix commands and programming utilities will be introduced. Students will learn how to use email, a text editor, and manage files and directories. This course is designed for students who have no experience with Unix. [F, W, S]	1
1	cs215	Unix Programming Environment	This course covers common Unix commands, shell scripting, regular expressions, tools and the applications used in a Unix programming environment. The tools to be introduced include make utility, a debugger, advanced text editing and text processing (vi, sed, tr). Prerequisite: CMPS 150 or permission of instructor.	5
2	cs216	Unix System Administration	This course covers the knowledge and skills critical to administering a multi-user, networked Unix system. The course assumes a basic knowledge of Unix commands and an editor (vi or Emacs). Topics include: kernel and network configuration, managing daemons, devices, and critical processes, controlling startup and shutdown events, account management, installing software, security issues, shell scripting. Many concepts will be demonstrated during hands-on labs. Prerequisite: CMPS 215.	5

3	cs221	Programming Fundamentals	Introduces the fundamentals of procedural programming. Topics include: data types, control structures, functions, arrays, and standard and file I/O. The mechanics of compiling, linking, running, debugging and testing within a particular programming environment are covered. Ethical issues and a historical perspective of programming within the context of computer science as a discipline is given. [F, W, S]	5
4	cs222	Object-Oriented Programming	Builds on foundation provided by CMPS 221 to introduce the concepts of object-oriented programming. The course focuses on the definition and use of classes and the fundamentals of object-oriented design. Other topics include: an overview of programming language principles, basic searching and sorting techniques, and an introduction to software engineering issues. Prerequisite: CMPS 221. [F, W, S]	5
5	cs223	Data Structures and Algorithms	Builds on the foundation provided by the CMPS 221-222 sequence to introduce the fundamental concepts of data structures and the algorithms that proceed from them within the framework of object-oriented programming methodology. Topics include: recursion, fundamental data structures (including stacks, queues, linked lists, hash tables, trees, and graphs), and the basics of algorithmic analysis. Prerequisite: CMPS 222. [F, W, S]	5
6	cs300	Discrete Structures	Elementary logic and set theory, functions and relations, induction and recursion, elementary algorithm analysis, counting techniques, and introduction to computability. Prerequisite: CMPS 223.	5
7	cs312	Algorithm Analysis	Algorithm analysis, asymptotic notation, hashing, hash tables, scatter tables, and AVL and B-trees, brute-force and greedy algorithms, divide-and-conquer algorithms, dynamic programming, randomized algorithms, graphs and graph algorithms, and distributed algorithms. Prerequisite: CMPS 223.	5
8	cs320	Digital Circuits	An introduction to the logical design of digital computers including the analysis and synthesis of combinatorial and sequential circuits, and the use of such circuits in building processor components and memory. The course will apply the circuit theory to the design of an elementary processor with a small instruction set with absolute addressing and a hard-wired control unit. An assembly language for this processor will also be developed. This course includes a laboratory which will cover a mix of actual circuit work together with circuit synthesis and testing using software. Prerequisite: One course in programming or permission of	5



		the instructor.		
9	cs321	Computer Architecture	<p>This course follows the Digital Logic Design course and focuses on the design of the CPU and computer system at the architectural (or functional) level: CPU instruction sets and functional units, data types, control unit design, interrupt handling and DMA, I/O support, memory hierarchy, virtual memory, and buses and bus timing. In contrast, the Digital logic Design course is primarily concerned with implementation that is, the combinatorial and sequential circuits which are the building blocks of the functional units. Prerequisite: CMPS 223 and 320.</p>	5
10	cs335	Software Engineering	<p>Introduction to the software life cycle. Methods and tools for the analysis, design, and specification of large, complex software systems. Project documentation, organization and control, communication, and time and cost estimates. Group laboratory project. Graphical User Interface Design. Technical presentation methods and practice. Software design case studies and practices. Prerequisite: CMPS 223.</p>	5
11	cs342	Database Systems	<p>Basic issues in data modeling, database application software design and implementation. File organizations, relational model, relational database management systems, and query languages are addressed in detail. Two-tier architecture, three-tier architecture and development tools are covered. Prerequisite: CMPS 223.</p>	5
12	cs350	Programming Language	<p>An examination of underlying concepts in high level programming languages and techniques for the implementation of a representative sample of such languages with regard to considerations such as typing, block structure, scope, recursion, procedures invocation, context, binding, and modularity. Prerequisite: CMPS 223.</p>	5
13	cs356	Artificial Intelligence	<p>This course is intended to teach the fundamentals of artificial intelligence which include topics such as expert systems, artificial neural networks, fuzzy logic, inductive learning and evolutionary algorithms. Prerequisite: CMPS 223.</p>	5
14	cs360	Operating Systems	<p>A study of the introductory concepts in operating systems: historical development of batch, multiprogrammed, and interactive systems file, memory, device, process, and thread management interrupt and trap handlers, abstraction layer, message passing kernel tasks and kernel design issues signals and interprocess communication synchronization, concurrency, and deadlock problems. Prerequisite: CMPS 223.</p>	5
15	cs371	Computer Graphics	<p>Introduction to computer graphics hardware</p>	5

	s	re, animation, two-dimensional transformations, basic concepts of computer graphics, theory and implementation. Use of graphics APIs such as DirectX or OpenGL. Developing 2D graphics applications software. Prerequisite: CMPS 223.	
16	cs376 Networking	A study of computer networks focusing on the TCP/IP Internet protocols and covering in detail the four layers: physical, data link, network, and transport. This course includes a laboratory in which students will cover important network utilities, debugging tools, process and thread control as it relates to network programming, and the coding of programs which do interprocess communication over sockets. The typical Internet client program which accesses a TCP network server daemon will be covered in detail. Prerequisite: CMPS 223.	5
17	cs457 Robotics	The course will provide an opportunity for students to understand intelligent robot system architecture and to design algorithms and programs for control and planning of intelligent robot systems based on analytical modeling and behavior modeling. Students will use simulation software (Webots) and hardware test-bed (Kheper II) to verify their algorithm and program performance during their project work. Prerequisites: CMPS 223.	5

### 5.4.2 js\_person

This relation holds general information about the persons(students or instructors). Including: the student/instructor ID, first name, middle name, last name, primary email, and primary phone.

#### Relation Schema:

```
SQL> DESC js_person;
```

Name	Null?	Type
ID	NOT NULL	NUMBER(38)
FNAME	NOT NULL	VARCHAR2(64)
MNAME		VARCHAR2(64)
LNAME	NOT NULL	VARCHAR2(64)
P_EMAIL		VARCHAR2(128)
P_PHONE		VARCHAR2(16)

#### Relation Instances:

```
SQL> SELECT * FROM js_person;
```

ID	FNAME	MNAME	LNAME	P_EMAIL	P_PHONE
3651553	Mark	L.	Getty	mgetty@cs.csubak.edu	(661) 945-0331
13766087	Conan	S.	Warashky	cwarashk@cs.csubak.edu	(661) 393-7972
21316065	Marc		Black	mblack@cs.csubak.edu	(661) 492-4758
42840922	William	A.	Sutton	wsutton@cs.csubak.edu	(661) 326-4168
66403651	Chris	E.	Gonzales	cgonzale@cs.csubak.edu	(661) 392-0595
85776546	Eugene	C.	Black	eblack@cs.csubak.edu	(661) 289-2452
89625452	Joseph	B.	Zappa	jzappa@cs.csubak.edu	(661) 348-5098
90141755	Sam		Smith	ssmith@cs.csubak.edu	(661) 860-0016
106575040	Casey		Jobs	cjobs@cs.csubak.edu	(661) 663-5469
125109089	Byran	C.	Warashky	bwarashk@cs.csubak.edu	(661) 409-3357
131493310	Sam		Black	sblack@cs.csubak.edu	(661) 785-1508
133768257	Pat	B.	Kerry	pkerry@cs.csubak.edu	(661) 627-5037
139865559	Casey	E.	Chung	cchung@cs.csubak.edu	(661) 107-4858
141198552	Donna		Johnson	djohnson@cs.csubak.edu	(661) 630-6610
146209587	Jon	A.	Langen	jlangen@cs.csubak.edu	(661) 581-4637
147650213	Maureen	L.	Tran	mtran@cs.csubak.edu	(661) 391-4379
150120557	Sam	L.	Johnson	sjohnson@cs.csubak.edu	(661) 333-1365
165299473	Mark	J.	Johnson	mjohnson@cs.csubak.edu	(661) 368-3724
167326160	Conan	C.	Danforth	cdanfort@cs.csubak.edu	(661) 396-8122
167495791	Chris		Sutton	csutton@cs.csubak.edu	(661) 182-2200
169640153	Brian	A.	Piven	bpiven@cs.csubak.edu	(661) 460-7443
172785946	Maureen	P.	Nevsky	mnevsky@cs.csubak.edu	(661) 620-2720
174940733	Bob	F.	Basheta	bbasheta@cs.csubak.edu	(661) 767-5866
185445705	Brenda	E.	Johnson	bjohnson@cs.csubak.edu	(661) 265-2151
186648582	Mark	S.	Thomas	mthomas@cs.csubak.edu	(661) 391-1997
188687932	Sam	L.	Gates	sgates@cs.csubak.edu	(661) 188-8970
197400921	Melissa	A.	Gates	mgates@cs.csubak.edu	(661) 408-6812
199768032	Scott	C.	Chung	schung@cs.csubak.edu	(661) 804-9483
219225017	Donna	P.	Gates	dgates@cs.csubak.edu	(661) 576-6795
225712527	Nick	E.	Zappa	nzappa@cs.csubak.edu	(661) 883-2867
234813752	Jennifer	F.	Thomas	jthomas@cs.csubak.edu	(661) 340-4698
242530867	Bob	C.	Gates	bgates@cs.csubak.edu	(661) 958-2143
242579947	Bob	P.	Martinez	bmartinez@cs.csubak.edu	(661) 994-6330
245815123	Jon	B.	Warashky	jwarashk@cs.csubak.edu	(661) 852-5939
253221481	Lauren	F.	Rodregez	lrodrige@cs.csubak.edu	(661) 426-6137
260955005	George		Meyers	gmeyers@cs.csubak.edu	(661) 965-0212
262329876	Eugene		Tran	etran@cs.csubak.edu	(661) 388-5155

263996508	Linda	F.	Martinez	lmartine@cs.csubak.edu	(661)	867-7947
272741205	Sam	S.	Sutton	ssutton@cs.csubak.edu	(661)	904-2508
289462976	Pat	E.	Rodregez	prodrege@cs.csubak.edu	(661)	612-8684
300328538	Donna		Piven	dpiven@cs.csubak.edu	(661)	815-5125
300867195	Scott		Wang	swang@cs.csubak.edu	(661)	538-4559
313252727	Scott		Piven	spiven@cs.csubak.edu	(661)	852-0862
318205259	Bob	L.	Wang	bwang@cs.csubak.edu	(661)	131-0177
320362253	Bill	C.	Gonzales	bgonzale@cs.csubak.edu	(661)	214-5916
321141053	Casey	P.	Piven	cpiven@cs.csubak.edu	(661)	307-2742
330200714	Jack	L.	Kerry	jkerry@cs.csubak.edu	(661)	398-5131
338566364	Marc	J.	Danforth	mdanfort@cs.csubak.edu	(661)	312-7029
346970481	Luis	B.	Warashky	lwarashk@cs.csubak.edu	(661)	119-8442
354641836	Mark	A.	Brian	mbrian@cs.csubak.edu	(661)	263-2773
369219713	Linda	E.	Nevsky	lnevsky@cs.csubak.edu	(661)	903-2843
381837284	Scott		Nevsky	snevsky@cs.csubak.edu	(661)	997-1796
395162265	Jack	F.	Piven	jpiven@cs.csubak.edu	(661)	895-3509
398778602	Eugene	A.	Guteriuz	eguteriu@cs.csubak.edu	(661)	809-6064
412511608	George	S.	Jobs	gjobs@cs.csubak.edu	(661)	157-6179
417420769	Bob	F.	Getty	bgetty@cs.csubak.edu	(661)	580-3951
418911842	Lauren	E.	Black	lblack@cs.csubak.edu	(661)	860-1926
419425447	Sam	F.	Rivers	srivers@cs.csubak.edu	(661)	208-1187
440965598	Melissa	F.	Gonzales	mgonzale@cs.csubak.edu	(661)	714-6265
454422757	Jennifer	S.	Brian	jbrian@cs.csubak.edu	(661)	581-2413
457555131	George	J.	Kerry	gkerry@cs.csubak.edu	(661)	891-8394
459692930	Kris	P.	Warashky	kwarashk@cs.csubak.edu	(661)	538-6041
463882377	Pat	J.	Rivers	privers@cs.csubak.edu	(661)	140-4662
463992699	Casey	L.	Langen	clangen@cs.csubak.edu	(661)	494-0091
465754579	Mark	J.	Chung	mchung@cs.csubak.edu	(661)	556-2538
470389394	Kris	P.	Rivers	krivers@cs.csubak.edu	(661)	881-4836
475493900	Marc		Smith	msmith@cs.csubak.edu	(661)	445-7466
498943316	Chris		Kerry	ckerry@cs.csubak.edu	(661)	388-7371
510627845	Marc	F.	Guteriuz	mguteriu@cs.csubak.edu	(661)	528-6521
517173452	Steve	L.	Meyers	smeyers@cs.csubak.edu	(661)	175-8858

### 5.4.3 js\_instructor

This relation holds more information about the instructors.

Including: the login password, website, secondary email, and secondary phone.

#### Relation Schema:

```
SQL> DESC js_instructor;
```

Name	Null?	Type
-----	-----	-----
ID	NOT NULL	NUMBER (38)
PASSWORD	NOT NULL	VARCHAR2 (32)
WEBSITE		VARCHAR2 (256)
S_EMAIL		VARCHAR2 (128)
S_PHONE		VARCHAR2 (16)

#### Relation Instances:

```
SQL> SELECT * FROM js_instructor;
```

ID	PASSWORD	WEBSITE	S_EMAIL	S_PHONE
-----	-----	-----	-----	-----
139865559	2K79ZwYS	http://www.cs.csubak.edu/~cchung		(661) 401-6090
167495791	4Zs_L	http://www.cs.csubak.edu/~csutton	csutton.csub@gmail.com	(661) 959-4341
188687932	?t1VkytN	http://www.cs.csubak.edu/~sgates		(661) 750-1671
318205259	DXxlmvr	http://www.cs.csubak.edu/~bwang		
320362253	v9W5q2pD	http://www.cs.csubak.edu/~bgonzalez	bill.gonz@hotmail.com	(661) 559-2550
346970481	O8FRsYdH	http://www.cs.csubak.edu/~lwarashk		(661) 572-3429
412511608	M/wc=	http://www.cs.csubak.edu/~gjjobs		
419425447	Gv/QJGdI	http://www.cs.csubak.edu/~srivers	rivercake@aol.com	
463882377	EsAsjy7	http://www.cs.csubak.edu/~privers		(661) 245-6027
517173452	p~cfbavt	http://www.cs.csubak.edu/~smeyers		(661) 125-5780

#### 5.4.4 js\_section

This relation holds information about the sections.

Including: the CRN, season number, starting date, ending date, and maximum allowed students.

Relation Schema:

```
SQL> DESC js_section;
```

Name	Null?	Type
SECTION_ID	NOT NULL	NUMBER(38)
COURSE_ID	NOT NULL	NUMBER(38)
ID	NOT NULL	NUMBER(38)
CRN	NOT NULL	NUMBER(38)
SEASON_NUM	NOT NULL	NUMBER(1)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
MAX_STUDENTS	NOT NULL	NUMBER(3)

Relation Instances:

```
SQL> SELECT * FROM js_section;
```

SECTION_ID	COURSE_ID	ID	CRN	SEASON_NUM	START_DAT	END_DATE	MAX_STUDENTS
0	0	318205259	41503	0	12-SEP-05	02-DEC-05	50
1	1	188687932	41506	0	12-SEP-05	02-DEC-05	22
2	3	139865559	41505	0	12-SEP-05	02-DEC-05	26
3	3	167495791	41507	0	12-SEP-05	02-DEC-05	28
4	4	463882377	41508	0	12-SEP-05	02-DEC-05	22
5	5	346970481	41509	0	12-SEP-05	02-DEC-05	22
6	7	167495791	41510	0	12-SEP-05	02-DEC-05	32
7	8	419425447	41511	0	12-SEP-05	02-DEC-05	22
8	11	318205259	41512	0	12-SEP-05	02-DEC-05	22
9	15	139865559	41513	0	12-SEP-05	02-DEC-05	22
10	17	320362253	41515	0	12-SEP-05	02-DEC-05	22
11	6	320362253	41515	0	12-SEP-05	02-DEC-05	22
12	9	320362253	41515	0	12-SEP-05	02-DEC-05	22
13	10	320362253	41515	0	12-SEP-05	02-DEC-05	22
14	11	320362253	41515	0	12-SEP-05	02-DEC-05	22
15	12	320362253	41515	0	12-SEP-05	02-DEC-05	22
16	13	320362253	41515	0	12-SEP-05	02-DEC-05	22
17	14	320362253	41515	0	12-SEP-05	02-DEC-05	22
18	16	320362253	41515	0	12-SEP-05	02-DEC-05	22
19	2	320362253	41515	0	12-SEP-05	02-DEC-05	22

#### 5.4.5 js\_signup

This relation holds information about the sign-up.

Including: the final letter grade, the final grade points (could be percentage), and details.

##### Relation Schema:

```
SQL> DESC js_signup;
```

Name	Null?	Type
ID	NOT NULL	NUMBER(38)
SECTION_ID	NOT NULL	NUMBER(38)
GRADE_LETTER		VARCHAR2(4)
GRADE_POINTS		FLOAT(126)
DETAILS		VARCHAR2(1024)

##### Relation Instances:

```
SQL> SELECT * FROM js_signup;
```

ID	SECTION_ID	GRAD	GRADE_POINTS	DETAILS
150120557	8	D	63.4	
89625452	0	B-	80.8	
459692930	8	C-	71	
131493310	6	D-	61.5	Incomplete
459692930	7	B	85.7	
234813752	0			
253221481	8			
242530867	7	C	74.1	
172785946	9			
242530867	9	A	99.3	
369219713	5	C+	78.3	
459692930	1			
418911842	7			
185445705	1			
165299473	6	A-	90.3	Incomplete
440965598	9	C	73.7	
272741205	5			
174940733	0	D-	61.8	
150120557	10			Incomplete
272741205	6	C	74.7	
165299473	4	C	75.9	
125109089	0	C-	72.7	
262329876	3	F	33.9	
330200714	2			
199768032	3	D	66.1	
470389394	6	B-	81.7	
330200714	1			
330200714	0			
330200714	4			
330200714	5			

330200714	6
330200714	7
330200714	8
330200714	9
330200714	10
330200714	11
330200714	12
330200714	13
330200714	14
330200714	15
330200714	16
330200714	17
330200714	18
330200714	19



#### 5.4.6 js\_assign

This relation holds information about the assignments.

Including: the name(title), type number(for grouping), max points, whether it's extra credit or not, and details.

##### Relation Schema:

```
SQL> DESC js_assign;
```

Name	Null?	Type
ASSIGN_ID	NOT NULL	NUMBER(38)
SECTION_ID	NOT NULL	NUMBER(38)
NAME	NOT NULL	VARCHAR2(64)
TYPE_NUM	NOT NULL	NUMBER(2)
MAX_POINTS	NOT NULL	NUMBER(38)
IS_EXTRA	NOT NULL	NUMBER(1)
DETAILS		VARCHAR2(1024)

##### Relation Instances:

```
SQL> SELECT * FROM js_assign;
```

ASSIGN_ID	SECTION_ID	NAME	TYPE_NUM	MAX_POINTS	IS_EXTRA	DETAILS
0	3	Lab	0	80	0	
1	9	Lab	0	170	0	
2	8	Quiz	1	30	1	
3	8	Final	4	10	0	question 20 thrown out
4	0	Quiz	1	80	0	
5	5	Test	2	150	0	
6	0	Test	2	60	0	
7	10	Quiz	1	160	0	
8	1	Lab	0	20	1	
9	1	Final	4	110	0	
10	3	Quiz	1	130	0	
11	3	Final	4	50	0	
12	6	Quiz	1	120	0	
13	3	Lab	0	120	0	
14	5	Test	2	70	0	
15	3	Final	4	40	0	
16	10	Final	4	50	0	
17	7	Lab	0	160	0	
18	4	Quiz	1	100	0	
19	5	Lab	0	120	0	
20	10	Mid Term	3	160	0	
21	5	Lab	0	80	1	
22	6	Final	4	100	0	
23	7	Lab	0	70	1	
24	1	Lab	0	80	0	
25	5	Lab	0	100	1	
26	7	Lab	0	100	0	
27	8	Quiz	1	130	0	
28	3	Test	2	30	0	
29	2	Lab	0	80	0	
30	9	Quiz	1	80	1	
31	2	Quiz	1	170	0	
32	2	Quiz	1	100	1	
33	9	Lab	0	140	0	
34	1	Mid Term	3	50	0	
35	9	Lab	0	170	0	

36	5 Quiz	1	20	0
37	2 Quiz	1	50	0
38	3 Mid Term	3	20	0 has bonus problem
39	1 Mid Term	3	40	0
40	6 Mid Term	3	130	0
41	7 Quiz	1	20	0
42	6 Final	4	120	0
43	10 Test	2	100	0
44	1 Test	2	10	0
45	0 Mid Term	3	140	0 question 12 thrown out
46	10 Final	4	140	0
47	2 Quiz	1	60	1
48	9 Test	2	80	0
49	1 Quiz	1	100	0
50	8 Lab	0	10	0
51	6 Final	4	80	0 question 5 thrown out
52	10 Mid Term	3	20	0
53	5 Lab	0	190	0
54	2 Lab	0	100	0
55	6 Quiz	1	120	0
56	9 Lab	0	110	0
57	1 Quiz	1	90	0
58	2 Quiz	1	70	0
59	0 Quiz	1	110	0
60	1 Quiz	1	30	0
61	10 Test	2	130	0 has bonus problem
62	0 Test	2	120	0
63	8 Lab	0	70	0
64	1 Lab	0	160	0
65	4 Lab	0	130	0
66	1 Quiz	1	140	0
67	4 Test	2	200	0

#### 5.4.7 js\_grade

This table holds information about the grades.

Including: the points given to an assignment for a student, the date entered, and details.

##### Relation Schema:

```
SQL> DESC js_grade;
```

Name	Null?	Type
GRADE_ID	NOT NULL	NUMBER(38)
ASSIGN_ID	NOT NULL	NUMBER(38)
ID	NOT NULL	NUMBER(38)
POINTS	NOT NULL	NUMBER(38)
DATE_ENTERED	NOT NULL	DATE
DETAILS		VARCHAR2(1024)

##### Relation Instances:

```
SQL> SELECT * FROM js_grade;
```

GRADE_ID	ASSIGN_ID	ID	POINTS	DATE_ENTERED	DETAILS
0	27	253221481	117	13-OCT-05	
1	30	172785946	8	03-NOV-05	
2	2	459692930	5	02-NOV-05	
3	35	440965598	117	15-OCT-05	turned in late
4	0	262329876	45	02-NOV-05	
5	62	234813752	30	01-OCT-05	
6	64	330200714	121	03-OCT-05	
7	17	242530867	70	02-NOV-05	
8	53	272741205	111	17-OCT-05	
9	52	150120557	7	30-NOV-05	
10	30	172785946	18	03-NOV-05	
11	0	262329876	65	05-NOV-05	
12	22	165299473	82	26-OCT-05	
13	12	165299473	13	11-OCT-05	
14	5	369219713	119	19-OCT-05	turned in late
15	29	330200714	42	21-NOV-05	
16	40	470389394	93	11-NOV-05	
17	26	418911842	40	18-OCT-05	
18	12	165299473	42	13-OCT-05	
19	17	242530867	30	21-OCT-05	
20	2	150120557	20	17-OCT-05	
21	25	369219713	85	11-NOV-05	
22	45	234813752	56	14-OCT-05	
23	66	459692930	67	26-NOV-05	
24	5	272741205	48	21-OCT-05	
25	12	165299473	41	14-NOV-05	
26	27	150120557	15	30-OCT-05	
27	11	262329876	0	23-OCT-05	
28	27	150120557	47	01-NOV-05	
29	40	131493310	59	27-NOV-05	

30	24	185445705	75	13-OCT-05	
31	41	242530867	12	17-OCT-05	
32	45	234813752	104	24-NOV-05	
33	40	131493310	85	16-OCT-05	
34	38	199768032	10	20-NOV-05	
35	27	459692930	125	19-NOV-05	
36	59	174940733	24	26-NOV-05	
37	51	165299473	5	04-OCT-05	
38	5	272741205	103	06-NOV-05	
39	10	262329876	109	04-NOV-05	
40	5	369219713	84	07-OCT-05	
41	41	418911842	10	13-NOV-05	
42	37	330200714	25	04-NOV-05	turned in late
43	40	470389394	69	11-OCT-05	
44	45	234813752	138	05-OCT-05	
45	51	165299473	46	14-NOV-05	
46	28	262329876	3	25-NOV-05	
47	17	459692930	63	26-OCT-05	
48	64	459692930	55	06-OCT-05	turned in late
49	27	459692930	32	06-NOV-05	
50	34	185445705	30	08-NOV-05	
51	17	459692930	50	19-OCT-05	
52	36	272741205	17	10-NOV-05	turned in late
53	53	369219713	69	06-OCT-05	
54	67	165299473	54	11-NOV-05	
55	56	242530867	6	11-OCT-05	
56	27	459692930	93	28-OCT-05	
57	4	234813752	63	22-NOV-05	
58	17	459692930	115	16-NOV-05	
59	25	272741205	23	29-OCT-05	
60	50	150120557	4	20-NOV-05	
61	62	234813752	9	15-NOV-05	
62	22	272741205	72	03-OCT-05	
63	17	242530867	160	15-OCT-05	

## 5.5 Sample SQL Queries

In section 4.3 a set of non-trivial queries were proposed. The following section uses Structured Query Language (SQL) to answer them, and displays sample results based on the data described above.

Note: Some of the queries were changed slightly(what value looking for) in order to display a result, and additional queries were created to demonstrate other possible queries.

### 5.5.1 List all students id's who has taken “cs215”.

SQL:

```
SELECT p.id "Student ID's who took cs215"
FROM js_person p, js_course_info c, js_section se, js_signup si
WHERE
    lower(c.short_name) = 'cs215' AND
    se.course_id = c.course_id AND
    si.id = p.id AND
    se.section_id = si.section_id
```

Result:

Student ID's who took cs215
-----
459692930
185445705
330200714

### 5.5.2 List all instructors who taught “cs221”.

SQL:

```
-- List all instructors who taught "cs221".
COLUMN "ID" FORMAT 9999999999
COLUMN "FName" FORMAT A5
COLUMN "MName" FORMAT A5
COLUMN "LName" FORMAT A6
COLUMN "Email 1" FORMAT A22
COLUMN "Email 2" FORMAT A22
COLUMN "Phone 1" FORMAT A14
COLUMN "Phone 2" FORMAT A14
COLUMN "Password" FORMAT A8
COLUMN "Website" FORMAT A33
SELECT p.id "ID",
       p.fname "FName",
       p.mname "MName",
       p.lname "LName",
       p.p_email "Email 1",
       i.s_email "Email 2",
       p.p_phone "Phone 1",
       i.s_phone "Phone 2",
       i.password "Password",
       i.website "Website"
FROM js_person p, js_instructor i, js_course_info c, js_section
se
WHERE
    lower(c.short_name) = 'cs221' AND
    se.course_id = c.course_id AND
    se.id = p.id AND
    p.id = i.id
;
```

Result:

ID	FName	MName	LName	Email 1
139865559	Casey	E.	Chung	cchung@cs.csubak.edu
167495791	Chris		Sutton	csutton@cs.csubak.edu
...	Email 2		Phone 1	Phone 2
...	-----			
...			(661) 107-4858	(661) 401-6090
...	csutton.csub@gmail.com		(661) 182-2200	(661) 959-4341
...	Password	Website		
...	-----			
...	2K79ZwYS	http://www.cs.csubak.edu/~cchung		
...	4Zs_L	http://www.cs.csubak.edu/~csutton		

5.5.3 List all course short names, section crn, start date, end date, and worth for Fall(season\_num=0) 2005.

SQL:

```
COLUMN "Short Name" FORMAT A10
COLUMN "CRN" FORMAT 999999
COLUMN "Worth" FORMAT 99999
SELECT c.short_name "Short Name",
       s.crn "CRN",
       s.start_date "Start Date",
       s.end_date "End Date",
       c.worth "Worth"
FROM js_section s, js_course_info c
WHERE
    s.course_id = c.course_id AND
    s.season_num = 0 AND
    s.start_date >= to_date('1/1/2005', 'mm/dd/yyyy') AND
    s.end_date <= to_date('12/31/2005', 'mm/dd/yyyy')
```

Result:

Short Name	CRN	Start Dat	End Date	Worth
cs150	41503	12-SEP-05	02-DEC-05	1
cs215	41506	12-SEP-05	02-DEC-05	5
cs221	41505	12-SEP-05	02-DEC-05	5
cs221	41507	12-SEP-05	02-DEC-05	5
cs222	41508	12-SEP-05	02-DEC-05	5
cs223	41509	12-SEP-05	02-DEC-05	5
cs312	41510	12-SEP-05	02-DEC-05	5
cs320	41511	12-SEP-05	02-DEC-05	5
cs342	41512	12-SEP-05	02-DEC-05	5
cs371	41513	12-SEP-05	02-DEC-05	5
cs457	41515	12-SEP-05	02-DEC-05	5

5.5.4 List all grade letters, grade points and course short name, for student 165299473 in Fall(season\_num=0) 2005.

SQL:

```
COLUMN "Grade" FORMAT A5
COLUMN "Percent" FORMAT 999
COLUMN "Course" FORMAT A10
SELECT si.grade_letter "Grade",
       si.grade_points "Percent",
       c.short_name "Course"
FROM js_person p, js_signup si, js_section se, js_course_info c
WHERE
  p.id = si.id AND
  si.id = 165299473 AND
  si.section_id = se.section_id AND
  se.course_id = c.course_id AND
  se.season_num = 0 AND
  se.start_date >= to_date('1/1/2005', 'mm/dd/yyyy') AND
  se.end_date <= to_date('12/31/2005', 'mm/dd/yyyy')
```

Result:

Grade	Percent	Course
A-	90	cs312
C	76	cs222



### 5.5.5 List student's id, first & last name, average, min, max final grade points between 2004 and 2005.

SQL:

```
COLUMN "ID" FORMAT 999999999
COLUMN "Name" FORMAT A16
COLUMN "Avg Grade" FORMAT 999999
COLUMN "Min Grade" FORMAT 999999
COLUMN "Max Grade" FORMAT 999999
SELECT id "ID",
       name "Name",
       AVG(pts) "Avg Grade",
       MIN(pts) "Min Grade",
       MAX(pts) "Max Grade"
FROM (
  SELECT p.id id,
         (p.fname || ' ' || p.lname) name,
         si.grade_points pts
  FROM
    js_person p, js_signup si, js_section se
  WHERE
    si.section_id = se.section_id AND
    si.id = p.id AND
    se.start_date >= to_date('1/1/2004', 'mm/dd/yyyy') AND
    se.end_date <= to_date('12/31/2005', 'mm/dd/yyyy')
)
GROUP BY id, name
```

Result:

ID	Name	Avg Grade	Min Grade	Max Grade
89625452	Joseph Zappa	81	81	81
125109089	Byran Warashky	73	73	73
131493310	Sam Black	62	62	62
150120557	Sam Johnson	32	0	63
165299473	Mark Johnson	83	76	90
172785946	Maureen Nevsky	0	0	0
174940733	Bob Basheta	62	62	62
185445705	Brenda Johnson	0	0	0
199768032	Scott Chung	66	66	66
234813752	Jennifer Thomas	0	0	0
242530867	Bob Gates	87	74	99
253221481	Lauren Rodregez	0	0	0
262329876	Eugene Tran	34	34	34
272741205	Sam Sutton	37	0	75
330200714	Jack Kerry	0	0	0
369219713	Linda Nevsky	78	78	78
418911842	Lauren Black	0	0	0
440965598	Melissa Gonzales	74	74	74
459692930	Kris Warashky	52	0	86
470389394	Kris Rivers	82	82	82

5.5.6 List all students signed up for “cs342” in Fall(season\_num=0) 2005, taught by “Wang”(last name).

SQL:

```
COLUMN "ID" FORMAT 999999999
COLUMN "Name" FORMAT A32
SELECT su.id "ID",
       (su.fname || ' ' || su.lname) "Name"
FROM
  js_person su, js_signup si, js_person i,
  js_course_info c, js_section se
WHERE
  si.id = su.id AND
  si.section_id = se.section_id AND
  se.id = i.id AND
  lower(i.lname) = 'wang' AND
  se.course_id = c.course_id AND
  lower(c.short_name) = 'cs342' AND
  se.season_num = 0 AND
  se.start_date >= to_date('1/1/2005', 'mm/dd/yyyy') AND
  se.end_date <= to_date('12/31/2005', 'mm/dd/yyyy')
```

Result:

ID	Name
150120557	Sam Johnson
459692930	Kris Warashky
253221481	Lauren Rodregez

5.5.7 List all assignments for “cs223” in Fall(season\_num=0) 2005, taught by instructor 346970481. (Sort by assignment name)

SQL:

```
COLUMN "Name" FORMAT A10
COLUMN "Type" FORMAT 9
COLUMN "Max Points" FORMAT 9999
COLUMN "Is Extra" FORMAT 9
COLUMN "Details" FORMAT A10
SELECT a.name "Name",
       a.type_num "Type",
       a.max_points "Max Points",
       a.is_extra "Is Extra",
       a.details "Details"
FROM
  js_assign a, js_course_info c, js_section se
WHERE
  se.id = 346970481 AND
  lower(c.short_name) = 'cs223' AND
  se.course_id = c.course_id AND
  se.section_id = a.section_id AND
  se.season_num = 0 AND
  se.start_date >= to_date('1/1/2005', 'mm/dd/yyyy') AND
  se.end_date <= to_date('12/31/2005', 'mm/dd/yyyy')
ORDER BY a.name ASC
```

Result:

Name	Type	Max Points	Is Extra	Details
Lab	0	120	0	
Lab	0	80	1	
Lab	0	190	0	
Lab	0	100	1	
Quiz	1	20	0	
Test	2	150	0	
Test	2	70	0	

5.5.8 List assignment name, grade, max points, and is extra for student 369219713 in “cs223” Fall(season\_num=0) 2005 taught by instructor 346970481. (Sort by assignment name)

SQL:

```
COLUMN "Name" format A5
COLUMN "Points" format 999
COLUMN "Max Points" format 999
COLUMN "Is Extra" format 9
SELECT DISTINCT
    a.name "Name",
    g.points "Points",
    a.max_points "Max Points",
    a.is_extra "Is Extra"
FROM
    js_grade g, js_assign a, js_course_info c, js_section se
WHERE
    g.id = 369219713 AND
    lower(c.short_name) = 'cs223' AND
    se.season_num = 0 AND
    se.start_date >= to_date('1/1/2005', 'mm/dd/yyyy') AND
    se.end_date <= to_date('12/31/2005', 'mm/dd/yyyy') AND
    se.id = 346970481 AND
    se.course_id = c.course_id AND
    se.section_id = a.section_id AND
    g.assign_id = a.assign_id
ORDER BY a.name ASC
```

Result:

Name	Points	Max Points	Is Extra
Lab	69	190	0
Lab	85	100	1
Test	84	150	0
Test	119	150	0

5.5.9 List the min, max, and average points for each student in “cs223” Fall 2005, taught by instructor 346970481, on all assignment.

SQL:

```
SELECT id "ID",
       AVG(pts) "Avg Grade",
       MIN(pts) "Min Grade",
       MAX(pts) "Max Grade"
FROM (
  SELECT p.id id, g.points pts, a.name
  FROM
    js_course_info c, js_section se, js_person p, js_assign a,
    js_grade g
  WHERE
    se.id = 346970481 AND
    se.season_num = 0 AND
    se.start_date >= to_date('1/1/2004', 'mm/dd/yyyy') AND
    se.end_date <= to_date('12/31/2005', 'mm/dd/yyyy') AND
    lower(c.short_name) = 'cs223' AND
    se.course_id = c.course_id AND
    a.section_id = se.section_id AND
    g.assign_id = a.assign_id AND
    p.id = g.id
)
GROUP BY id
```

Result:

ID	Avg Grade	Min Grade	Max Grade
272741205	60	17	111
369219713	89	69	119

#### 5.5.10 Students that have taken all the courses.

SQL:

```
SELECT
  p.id "ID"
FROM
  js_person p
WHERE
  NOT EXISTS (
    SELECT c.course_id
    FROM js_course_info c
    WHERE
      NOT EXISTS (
        SELECT *
        FROM js_section se, js_signup si
        WHERE
          se.section_id = si.section_id AND
          se.course_id = c.course_id AND
          si.id = p.id
      )
  )
)
```

Result:

ID
330200714

5.5.11 List all student id's and course short name that have not gotten a grade for a course that they signup for. (Sorted by course name)

SQL:

```
COLUMN "ID" format 999999999
COLUMN "Course Name" format A12
SELECT p.id "ID", c.short_name "Course Name"
FROM js_person p, js_signup si, js_section se, js_course_info c
WHERE
  p.id = si.id AND
  si.section_id = se.section_id AND
  se.course_id = c.course_id AND
  si.grade_letter is NULL
ORDER BY
  c.short_name ASC
```

Result:

ID	Course Name
234813752	cs150
459692930	cs215
185445705	cs215
330200714	cs215
330200714	cs221
272741205	cs223
418911842	cs320
253221481	cs342
172785946	cs371
150120557	cs457

### 5.5.12 List all the latest grades.

SQL:

```
SELECT g1.grade_id, g1.id, g1.assign_id
FROM js_grade g1
WHERE
    NOT EXISTS (
        SELECT * from js_grade g2
        WHERE g1.date_entered < g2.date_entered AND
              g1.assign_id = g2.assign_id AND
              g1.id = g2.id
    )
ORDER BY g1.id ASC
;
```

Result:

GRADE_ID	ID	ASSIGN_ID
29	131493310	40
9	150120557	52
20	150120557	2
60	150120557	50
28	150120557	27
12	165299473	22
45	165299473	51
54	165299473	67
25	165299473	12
1	172785946	30
10	172785946	30
36	174940733	59
30	185445705	24
50	185445705	34
34	199768032	38
32	234813752	45
61	234813752	62
57	234813752	4
7	242530867	17
31	242530867	41
55	242530867	56
0	253221481	27
11	262329876	0
39	262329876	10
46	262329876	28
27	262329876	11
8	272741205	53
62	272741205	22
59	272741205	25
52	272741205	36
38	272741205	5
6	330200714	64
15	330200714	29
42	330200714	37
14	369219713	5
53	369219713	53
21	369219713	25
17	418911842	26
41	418911842	41
3	440965598	35
2	459692930	2
23	459692930	66
58	459692930	17
48	459692930	64
35	459692930	27
16	470389394	40



5.5.13 List all assignments ID, grade ID, and grade points for all assignments. If a grade does not exist for that assignment then include it but leave the grade blank. (using a outer join)

SQL:

```
SELECT
    a.assign_id "Assign ID",
    g.grade_id "Grade ID",
    g.points "Grade Points"
FROM
    js_assign a, js_grade g
WHERE
    a.assign_id = g.assign_id(+)
ORDER BY a.assign_id ASC
;
```

Result:

Assign ID	Grade ID	Grade Points
0	4	45
0	11	65
1		
2	2	5
2	20	20
3		
4	57	63
5	14	119
5	38	103
5	40	84
5	24	48
6		
7		
8		
9		
10	39	109
11	27	0
12	13	13
12	18	42
12	25	41
13		
14		
15		
16		
17	7	70
17	47	63
17	19	30
17	51	50
17	58	115
17	63	160
18		
19		
20		
21		
22	12	82
22	62	72
23		
24	30	75
25	21	85
25	59	23
26	17	40

27	0	117
27	28	47
27	35	125
27	56	93
27	49	32
27	26	15
28	46	3
29	15	42
30	1	8
30	10	18
31		
32		
33		
34	50	30
35	3	117
36	52	17
37	42	25
38	34	10
39		
40	16	93
40	29	59
40	43	69
40	33	85
41	31	12
41	41	10
42		
43		
44		
45	22	56
45	44	138
45	32	104
46		
47		
48		
49		
50	60	4
51	37	5
51	45	46
52	9	7
53	8	111
53	53	69
54		
55		
56	55	6
57		
58		
59	36	24
60		
61		
62	5	30
62	61	9
63		
64	6	121
64	48	55
65		
66	23	67
67	54	54

#### 5.5.14 Create new table and insert all grades older then 60 days.

SQL:

```
CREATE TABLE js_old_grade AS
  SELECT g.*
  FROM js_grade g
  WHERE
    g.date_entered < (SYSDATE-60)
  ORDER BY g.grade_id
;
```

Result:

Table created.

SQL> SELECT \* FROM js\_old\_grade;

GRADE_ID	ASSIGN_ID	ID	POINTS	DATE_ENTE	DETAILS
0	27	253221481	117	13-OCT-05	
1	30	172785946	8	03-NOV-05	
2	2	459692930	5	02-NOV-05	
3	35	440965598	117	15-OCT-05	turned in late
4	0	262329876	45	02-NOV-05	
5	62	234813752	30	01-OCT-05	
6	64	330200714	121	03-OCT-05	
7	17	242530867	70	02-NOV-05	
8	53	272741205	111	17-OCT-05	
9	52	150120557	7	30-NOV-05	
10	30	172785946	18	03-NOV-05	
11	0	262329876	65	05-NOV-05	
12	22	165299473	82	26-OCT-05	
13	12	165299473	13	11-OCT-05	
14	5	369219713	119	19-OCT-05	turned in late
15	29	330200714	42	21-NOV-05	
16	40	470389394	93	11-NOV-05	
17	26	418911842	40	18-OCT-05	
18	12	165299473	42	13-OCT-05	
19	17	242530867	30	21-OCT-05	
20	2	150120557	20	17-OCT-05	
21	25	369219713	85	11-NOV-05	
22	45	234813752	56	14-OCT-05	
23	66	459692930	67	26-NOV-05	
24	5	272741205	48	21-OCT-05	
25	12	165299473	41	14-NOV-05	
26	27	150120557	15	30-OCT-05	
27	11	262329876	0	23-OCT-05	
28	27	150120557	47	01-NOV-05	
29	40	131493310	59	27-NOV-05	
30	24	185445705	75	13-OCT-05	
31	41	242530867	12	17-OCT-05	
32	45	234813752	104	24-NOV-05	
33	40	131493310	85	16-OCT-05	
34	38	199768032	10	20-NOV-05	
35	27	459692930	125	19-NOV-05	
36	59	174940733	24	26-NOV-05	
37	51	165299473	5	04-OCT-05	
38	5	272741205	103	06-NOV-05	
39	10	262329876	109	04-NOV-05	
40	5	369219713	84	07-OCT-05	
41	41	418911842	10	13-NOV-05	
42	37	330200714	25	04-NOV-05	turned in late
43	40	470389394	69	11-OCT-05	

44	45	234813752	138	05-OCT-05
45	51	165299473	46	14-NOV-05
46	28	262329876	3	25-NOV-05
47	17	459692930	63	26-OCT-05
48	64	459692930	55	06-OCT-05 turned in late
49	27	459692930	32	06-NOV-05
50	34	185445705	30	08-NOV-05
51	17	459692930	50	19-OCT-05
52	36	272741205	17	10-NOV-05 turned in late
53	53	369219713	69	06-OCT-05
54	67	165299473	54	11-NOV-05
55	56	242530867	6	11-OCT-05
56	27	459692930	93	28-OCT-05
57	4	234813752	63	22-NOV-05
58	17	459692930	115	16-NOV-05
59	25	272741205	23	29-OCT-05
60	50	150120557	4	20-NOV-05
61	62	234813752	9	15-NOV-05
62	22	272741205	72	03-OCT-05
63	17	242530867	160	15-OCT-05

## 5.6 Loading Data Methods

This section describe the methods of loading the data to the database. These methods include the INSERT statement, and using a data loader.

### 5.6.1 Description of INSERT Statement

The INSERT statement in SQL is used to add data to an existing table. The data can be inserted two slightly different ways, both using the INSERT SQL command.

a) The first method is to insert one row, from a given set of values.

```
INSERT INTO tableName [(columnList)]  
VALUES (valueList)
```

b) The second method inserts from a query(SELECT), this can insert zero or more rows at one time into the table.

```
INSERT INTO tableName [(columnList)]  
SELECT ...
```

However this can only be done if the following conditions are met<sup>1</sup>:

1. The number of items in each list are be the same.
2. There must be a direct correspondence in the position of items in the two lists, so that the first item valueList applies to the first item in columnList, the second item in valueList applies to the second item in columnList, and so on.
3. The data type of each item in valueList must be compatible with the data type of the corresponding column.

### 5.6.2 Data loader Program

#### 5.6.2.1 Automatic Data Insertion and DataLoader

Although the interactive SQL\*Plus interface is very useful for designing and modifying queries, it makes insertion of larges of of data very cumbersome.

A wide variety of tools can be used to expedite the data entry process, however a software developer would find it more convenient to write a custom program to do it the insertion.

For this project an automatic data insertions application called “DataLoader” was provided, however is needed one additional feature to make it more user friendly (these are outlined in the next section).

---

1 Connolly & Begg, Database Systems 4<sup>th</sup> edition

The data loader application asks the user for DBMS login and basic information about the data file.

Here are the steps the DataLoader follows in order to insert the data from the file to the server:

- 1) Ask user for login information.
- 2) Login to Oracle Server, using user login information.
- 3) Ask user what delimiter they want to use for there datafile.
- 4) Ask user for filename of delimited ASCII file.
- 5) Parse file by table.
- 6) Construct prepared statement for the table.
- 7) Construct INSERT SQL statement from prepared statement.
- 8) Execute SQL statement.
- 9) Go to 5. until there are no more tables.

DataLoader uses a library called JDBC. To connect to, be authenticated, and send SQL statements to the DBMS. In Java this process is quite simple and strait forward (for more details see the code below).

The data file is a delimited ASCII file, which is read by the data loader, parsed(broken up into rows and columns). Then this parsed data is inserted in to the DB.

Here is an example of a delimited ASCII file:

Table Name  js_section  8									
0	0	318205259	41503	0	2005-09-12	00:00:00	2005-12-02	00:00:00	50
1	1	188687932	41506	0	2005-09-12	00:00:00	2005-12-02	00:00:00	22
2	3	139865559	41505	0	2005-09-12	00:00:00	2005-12-02	00:00:00	26
3	3	167495791	41507	0	2005-09-12	00:00:00	2005-12-02	00:00:00	28
4	4	463882377	41508	0	2005-09-12	00:00:00	2005-12-02	00:00:00	22
5	5	346970481	41509	0	2005-09-12	00:00:00	2005-12-02	00:00:00	22
6	7	167495791	41510	0	2005-09-12	00:00:00	2005-12-02	00:00:00	32
7	8	419425447	41511	0	2005-09-12	00:00:00	2005-12-02	00:00:00	22
8	11	318205259	41512	0	2005-09-12	00:00:00	2005-12-02	00:00:00	22
9	15	139865559	41513	0	2005-09-12	00:00:00	2005-12-02	00:00:00	22
10	17	320362253	41515	0	2005-09-12	00:00:00	2005-12-02	00:00:00	22
Table Name  js_person  6									
3651553	Mark	L.	Getty	mgetty@cs.csubak.edu	(661)	945-0331			
13766087	Conan	S.	Warashky	cwarashk@cs.csubak.edu	(661)	393-7972			
21316065	Marc		Black	mblack@cs.csubak.edu	(661)	492-4758			
42840922	William	A.	Sutton	wsutton@cs.csubak.edu	(661)	326-4168			
66403651	Chris	E.	Gonzales	cgonzale@cs.csubak.edu	(661)	392-0595			
85776546	Eugene	C.	Black	eblack@cs.csubak.edu	(661)	289-2452			
89625452	Joseph	B.	Zappa	jzappa@cs.csubak.edu	(661)	348-5098			
90141755	Sam		Smith	ssmith@cs.csubak.edu	(661)	860-0016			
106575040	Casey		Jobs	cjobs@cs.csubak.edu	(661)	663-5469			
125109089	Byran	C.	Warashky	bwarashk@cs.csubak.edu	(661)	409-3357			
131493310	Sam		Black	sblack@cs.csubak.edu	(661)	785-1508			
133768257	Pat	B.	Kerry	pkerry@cs.csubak.edu	(661)	627-5037			

Some of the parts delimited ASCII file formatting is fixed and some parts are customizable. Here are the basic formatting requirements:

1. "Table Name" is a keyword that defines what table that data is going to be inserted into. It is followed by the actual table name in the database. Then the number of columns. So "Table Name|js\_section| 6", is going to insert data into the "js\_section" table with 6 columns.
2. Each row is always separated by a return.
3. In this example the columns are separated by the pipe "|". This can be changed to any single character you wish. The DataLoader application will ask you what delimiter you want to use.
4. If you are inserting a date or time it must be in the following format: (Notice that the DATE type of Oracle 8.05 is actually TIMESTAMP type, and the format of date should be yyyy-mm-dd hh:mm:ss)
  - a) Date: yyyy-mm-dd
  - b) Time: hh:mm:ss
  - c) Timestamp: yyyy-mm-dd hh:mm:ss.ffffffffff  
where ffffffffff is nano seconds

As mentioned earlier, in order for DataLoader to insert the data into the database it must construct a SQL statement. DataLoader specifically uses a method called "prepareStatement" where you create a template for an SQL statement, using methods to replace "?" in the template with given values, the statement is then executed.

For the first row in the example file the final SQL statement would look like:

```
INSERT INTO js_section VALUES(0, 0, 318205259, 41503, 0,  
to_date('2005-09-12 00:00:00'), to_date('2005-12-02  
00:00:00'), 50);
```

DataLoader repeatedly inserts into the current table until it hits another table then that table becomes the current table and so on.

#### 5.6.2.2 Modification of the DataLoader Program

DataLoader is a very useful application but it's lacking one thing. Allowing for command line argument to be passed in, instead of user input.

For example, instead of entering "cs342", "student2", "|" and "section.txt" when DataLoader runs. You could just pass them in from the command line. It would look like:

```
helios > java DataLoader "cs342" "student2" "|" "iags.txt"
```

For details on what changes were made, see code and comments below.

### 5.6.2.3 DataLoader.java Source Code

```
import java.sql.*;
import java.io.*;
import java.util.*;

/** SP data loader :
-----
ADDED by Joseph E. Sutton
Allowing for command line argument to be passed in, instead of user input.
This includes a new function to check the command line arguments, minor
conditional check, and updated the code formating.

Command Line Usage: DataLoader [username [password [delimiter [filename]]]]
-----

The data for S, P, SP table are stored in sp_data.txt with the following format:
Table name| S | 4
s6| Wang|          99| Bakersfield
.....

TABLENAME| P | 5
p1| Nut|          Red|          12| London
....

TABLENAME| SP | 3
s6| p6| 6
....

The word "Table Name" or "TableName" can be in any cases, and must
not be preceded by any other non white space letter.

To insert Date, Time and Timestamp data, the following methods can be used:
1. The Date, Time and Timestamp string should have the following
format in your data file:
    Date:          yyyy-mm-dd
    Time:          hh:mm:ss
    Timestamp:     yyyy-mm-dd hh:mm:ss.fffffffffff
                  where ffffffffff is nano seconds
2. Use Date, Time and Timestamp's static valueOf():
    static Date valueOf( String dateString);
    static Time valueOf( String timeString);
    static Timestamp valueOf( String timestampString);
to convert a date/time/timestamp string to a date, time or
a timestamp object, and then
3. Use the PreparedStatement's set functions to set the parameter:
    preparedStmt.setDate( i, dateObject);
    preparedStmt.setTime( i, timeObject);
    preparedStmt.setTimestamp( i, timestampObject);
4. Notice that the DATE type of Oracle 8.05 is actually TIMESTAMP type,
and the format of date should be yyyy-mm-dd hh:mm:ss.fffffffffff.
*/

public class DataLoader {
    Connection      cnn = null;
    PreparedStatement pStmt = null;
    Statement       stmt = null;
    ResultSet       res = null ;
    ResultSetMetaData meta = null;
    String          tableName = null,
                  colSepChars;

    int fieldCount;
    int insertSucc = 0, insertFail = 0, lineNo = 0;;

    String url = "jdbc:oracle:thin:@prover.cs.csubak.edu:1521:PROVERDB";
    static String askUser = "          Oracle user name: ";
    static String askPass = "          Oracle user password: ";
```



```

static String askChar = "    Column Separating char: ";
static String askFile = " Enter the data file name: ";

public DataLoader(String user, String passwd) {
    try {
        try { Class.forName("oracle.jdbc.driver.OracleDriver"); }
        catch (ClassNotFoundException ee) { ee.printStackTrace(); System.exit(-1); }

        cnn = DriverManager.getConnection(url, user, passwd);

    } catch (SQLException e ) { e.printStackTrace(); System.exit(-1); }
}

void buildPreparedStatement(String line) {
    StringTokenizer tkz = new StringTokenizer(line, colSepChars);
    /**
     * Skip "TABLENAME", get table name and number of fields.
     * Build INSERT statement, and prepare the insert statement.
     */
    tableName = tkz.nextToken();
    tableName = tkz.nextToken().trim();
    fieldCount = Integer.parseInt(tkz.nextToken().trim());
    StringBuffer buf = new StringBuffer();
    buf.append("INSERT INTO " + tableName + " VALUES(?");
    for ( int i = 1; i < fieldCount; i++ ) buf.append(", ?");

    buf.append(")");
    try {
        pstmt = cnn.prepareStatement(buf.toString());
        // The Java document says that it is positive to access
        // ResultSetMeta data before the statement is executed. However,
        // Oracle 8.05 has not implemented that. The statement doesn't
        // work. meta = pstmt.getMetaData();

        System.out.println("SQL statement: " + buf.toString());
        // In order to know the types of columns, execute SELECT
        // statement. It is inefficient.
        if ( stmt == null ) stmt = cnn.createStatement();

        res = stmt.executeQuery("select * from " + tableName);
        meta = res.getMetaData();
        res.close();
    } catch ( SQLException e ) { e.printStackTrace(); return; }
}

boolean addRecordToCurrentTable(int lineNo, String line ) {
    StringTokenizer tkz = new StringTokenizer(line, colSepChars);
    String colStr = null;

    try {
        pstmt.clearParameters();

        for ( int i = 1; i <= fieldCount; i ++ ) {
            colStr = tkz.nextToken().trim();

            switch( meta.getColumnType(i) ) {
                case Types.DATE:
                    pstmt.setDate(i, java.sql.Date.valueOf(colStr));
                    break;

                case Types.TIME:
                    pstmt.setTime(i, Time.valueOf( colStr));
                    break;

                case Types.TIMESTAMP: // Oracle DATE type is TIMESTAMP type.
                    pstmt.setTimestamp(i, Timestamp.valueOf( colStr ));
            }
        }
    }
}

```

```

                                break;

                                default :
                                    pstmt.setString(i, colStr);
                                    break;
                                }
                            }

                            pstmt.execute();
                            return true;

                        } catch ( SQLException e ) {
                            if ( insertFail < 3 )
                                System.out.println( "\n\t" + e.getMessage() +
                                    "\t on line " + lineNo + ": [" + line +
                                    "]" Column String = '" + colStr + "'"
                                );

                            return false;
                        }
                    }
                }

                // -----
                // ADDED by Joseph E. Sutton
                private static void parseArgs(String argv[],
                    String[] opts) {
                    // simply loop though arguments and set variables accordingly
                    for(int i = 0; i < argv.length; i++)
                    {
                        opts[i] = new String(argv[i]);
                    }
                }
                // -----

                public static void main(String argv[]) throws IOException {
                    boolean succ;
                    String line = null;
                    String upperCaseLine = null;
                    String opts[] = new String[4];
                    // 0 - UserName
                    // 1 - Password
                    // 2 - SepChar
                    // 3 - FileName

                    // -----
                    // ADDED by Joseph E. Sutton
                    // Check command line arguments, if they exist then set values
                    parseArgs(argv, opts);
                    // -----

                    // only ask if not set
                    if(opts[0] == null) opts[0] = ScreenIO.promptForString(askUser);
                    if(opts[1] == null) opts[1] = ScreenIO.promptForString(askPass);

                    // get connected, and create stmt.
                    DataLoader ldr = new DataLoader(opts[0], opts[1]);

                    // only ask if not set
                    if(opts[2] == null)
                        ldr.colSepChars = ScreenIO.promptForString(askChar);
                    else
                        ldr.colSepChars = opts[2];

                    // only ask if not set
                    if(opts[3] == null) opts[3] = ScreenIO.promptForString(askFile);

                    BufferedReader spFile = new BufferedReader(new FileReader(opts[3]));

```

```

while ( (line = spFile.readLine()) != null )
{
    if ( line.trim().equals("") ) continue;        // skip blank line.

    upperCaseLine = line.trim().toUpperCase();
    if ( upperCaseLine.indexOf("TABLENAME") == 0 ||
        upperCaseLine.indexOf("TABLE NAME") == 0 ) {
        ldr.printInsertionInfo();
        ldr.buildPreparedStatement(line);
        ldr.insertSucc = 0;
        ldr.insertFail = 0;
    } else {
        succ = ldr.addRecordToCurrentTable(ldr.lineNo, line);
        if ( succ ) ldr.insertSucc ++;
        else ldr.insertFail ++;
    }

    ldr.lineNo ++;
}

ldr.printInsertionInfo();
ldr.quit();
}

protected void printInsertionInfo() {
    System.out.println();

    if ( insertSucc > 0 )
        System.out.print( insertSucc + " records inserted" );

    if ( insertFail > 0 )
        System.out.print(" " + insertFail + " records failed" );

    if ( insertSucc > 0 || insertFail > 0 )
        System.out.println(" in table " + tableName + "\n\n");
}

protected void quit() {
    try {
        pStmt.close();
        stmt.close();
        cnn.close();
        System.exit(0);
    } catch (SQLException e) {}
    try { super.finalize(); } catch (Throwable e) {}
}
}

```

## Section 6: Oracle PL/SQL

In this section I will discussing the common features in Oracle PL/SQL & MS Transact-SQL, how to use Oracle PL/SQL, and Oracle PL/SQL subprogram developed for IAGS.

### 6.1 Common Features in Oracle PL/SQL and MS Transact-SQL

PL/SQL and Transact-SQL languages are very similar in concept, however each are usually only supported by a particular DBMS. Microsoft SQL Server supports Transact-SQL or T-SQL but not PL/SQL. While Oracle supports PL/SQL but not T-SQL.

#### 6.1.1 The Components of PL/SQL and T-SQL

##### PL/SQL

PL/SQL supports variables, conditions, arrays, and exceptions. Implementations from version 8 of the Oracle RDBMS onwards have included features associated with object-orientation.<sup>1</sup>

The underlying SQL functions as a declarative language. Standard SQL—unlike some functional programming languages—does not require implementations to convert tail calls to jumps. SQL does not readily provide "first row" and "rest of table" accessors, and it cannot easily perform some constructs such as loops. PL/SQL, however, as a Turing-complete procedural language which fills in these gaps, allows Oracle database developers to interface with the underlying relational database in an imperative manner. SQL statements can make explicit in-line calls to PL/SQL functions, or can cause PL/SQL triggers to fire upon pre-defined Data Manipulation Language (DML) events.<sup>1</sup>

PL/SQL stored procedures (functions, procedures, packages, and triggers) which perform DML get compiled into an Oracle database: to this extent their SQL code can undergo syntax-checking. Programmers working in an Oracle database environment can construct PL/SQL blocks of such functionality to serve as procedures, functions; or they can write in-line segments of PL/SQL within SQL\*Plus scripts.<sup>1</sup>

While programmers can readily incorporate SQL DML statements into PL/SQL (as cursor definitions, for example, or using the SELECT ... INTO syntax), Data Definition Language (DDL) statements such as CREATE TABLE/DROP INDEX etc require the use of "Dynamic SQL". Earlier versions of Oracle required the use of a complex built-in DBMS\_SQL package for Dynamic SQL where the system needed to explicitly parse and execute an SQL statement. Later versions have included an EXECUTE IMMEDIATE syntax called "Native Dynamic SQL" which considerably simplifies matters. Any use of DDL in Oracle will result in an implicit commit. Programmers can also use Dynamic SQL to execute DML where they do not know the exact content of the statement in advance.<sup>1</sup>

---

<sup>1</sup> <http://en.wikipedia.org/wiki/PL/SQL>

## T-SQL

Sometimes abbreviated T-SQL, Transact-SQL is Microsoft's and Sybase's proprietary extension to the SQL language. Microsoft's implementation ships in the Microsoft SQL Server product. Sybase uses the language in its Adaptive Server Enterprise, the successor to Sybase SQL Server.<sup>1</sup>

In order to make it more powerful, SQL has been enhanced with additional features such as: <sup>1</sup>

- Control-of-flow language
- Local variables
- User authentication integrated with Microsoft Windows
- Various support functions for string processing, date processing, mathematics, etc.
- Improvements to DELETE and UPDATE statements

### 6.1.2 Purposes of stored subprogram

Subprograms are named PS/SQL blocks that can take parameters and be invoked. PL/SQL has two types of subprogram called (stored) procedures and functions. Procedures and functions can take a set of parameters given to them by the calling program and perform a set of actions. Both can modify and data passed to them as a parameter. Stored procedures and functions are identical except that functions always return a value, procedures do not. By processing the SQL code on the database server, the number of instructions sent across the network and returned from the SQL statements are reduced. Also, each stored procedure runs in its own address space, and typically its own thread. Having its own address space allows the stored procedures to be called by multiple users at the same time asynchronously.<sup>2</sup>

### 6.1.3 Benefits of calling a stored subprogram

A user would want to use a stored procedure over the use of dynamic SQL from a front end application to the database server for the reasons mentioned above. It is also much preferable to do all data manipulation within stored procedures because the database can access and process its own internal data much quicker than a front end application. An application must: setup and format the query, run the query, build the data set, manipulate the dataset, then commit the modified values back to the server. This is a very time consuming process, and puts unnecessary strain on both the DMBS and the front end computer's processor.<sup>2</sup>

---

<sup>1</sup> <http://en.wikipedia.org/wiki/T-SQL>

<sup>2</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

## 6.2 Oracle PL/SQL

In this section I will discussing PL/SQL program structure, control statements, cursors, stored procedure, stored function, package, and trigger.

### 6.2.1 Program Structure

The basic units that comprise a PL/SQL sub program are procedures, functions, and anonymous blocks. Each program has at most three sections, and is based upon the ADA programming language.<sup>1</sup>

1. A data declaration section, which contains variables, constants, cursors, and exceptions. Each declaration may be initialized or not. This part of the program is optional.
2. An executable section. The executable section is the block of code where the actual data manipulation is done. This part is mandatory.
3. An exception section, which is where any user defined exception handling occurs. This part of the program is also optional.

A PL/SQL Program will have the following syntax:

```
[DECLARE
  -- optional
  -- this is where you declare variable
]

BEGIN
  -- mandatory
  -- executable statements

[EXCEPTION
  -- optional
  -- this is were you handle exceptions thrown,
  -- usually on an error
]

END;
```

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

### Declarations and Assignments

A variable is a symbol denoting a quantity or symbolic representation. An assignment sets or resets the value assigned to a variable. All variables in the sub program uses **MUST** be declared within the DECLARE section of the sub program.<sup>1</sup> Additionally we can declare a variable to be the same type as a specified column, or even an entire row.

The variable declaration has the following syntax:

```
...  
DECLARE  
...  
variableName {variableType |  
               relationName.columnName%TYPE |  
               relationName%ROWTYPE} [:= value];  
...  
BEGIN  
...
```

An example of a variable declaration:

```
...  
DECLARE  
-- declare variable without a value  
myVar1 NUMBER;  
  
-- declare variable with a value  
myVar2 NUMBER := 42;  
  
-- declare variable with same type as js_person column fname  
name js_person.fname%TYPE;  
  
-- declare variable as a row type from the js_section table  
section_row js_section%ROWTYPE;  
  
BEGIN  
...
```

All assignment operations use the ADA ':=' operator, rather than the traditional '=' operator found in many languages, including C, C++, and Java.

An assignment has the following syntax:

```
variableName := value;
```

An example of an assignment:

```
size := 1234;  
name := 'Joe';
```

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

### 6.2.2 Control Statement

Control statements may exist inside the executable section(between the BEGIN and END block) of the sub program. PL/SQL supports all usual conditional, iterative, and sequential statements.

#### 1. Conditional Statements

A conditional statement is a set of commands that executes if a specified condition is true as shown below. All conditions can be any expression that evaluates to true or false.<sup>1</sup>

An conditional statement has the following syntax:

```
IF (condition) THEN
    statement(s) ...
END IF;

IF (condition) THEN
    statement(s)1...
ELSE IF (condition) THEN
    statement(s)2...
ELSE
    statement(s)3...
END IF;

EXIT
WHEN [condition]
```

An example of an conditional statement:

```
IF (salary > 50000) THEN
    Statement1();
ELSE IF (salary <= 50000 AND salary >= 10000 ) THEN
    Statement2();
ELSE
    Statement3();
END IF;
```

Thus, if salary greater then 50,000 then Statement1 will be executed. If salary between 50,000 and 10,000 then Statement2 will be executed. If salary less then 10,000 then Statement3 will be executed.

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition



## 2. Iterative Controls

Iterative statements (for loop generation) are used for programming of cyclically iterative code sections. These statements have control structures that delimit them and which determine how many times (zero or more) the delimited code is executed, based on some condition.<sup>1</sup>

An iterative control has the following syntax:

```
LOOP
    statement(s) ...
END LOOP;

WHILE condition LOOP
    statement(s) ...
END LOOP;

FOR cursorVariable IN cursorName LOOP
    statement(s) ...
END LOOP;
```

An example of an iterative control:

```
i := 0;
WHILE i < 100 LOOP
    UPDATE js_signup SET grade_points=0.00 WHERE signup_id=i;
    i := i + 1;
END LOOP;
```

## 3. Sequential Control

Certain PL/SQL control structures offer structured methods for processing executable statements in your program. PL/SQL offers two statements to handle out of the ordinary requirements for sequential processing: GOTO and NULL. The GOTO statement allows you to perform unconditional branching to another executable statement in the same execution section of a PL/SQL block. The NULL statement gives you a way to tell the compiler to do...absolutely nothing.<sup>1</sup>

A goto statement has the following syntax:

```
...
<<labelName>>
...
GOTO labelName;
```

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

An example of a goto statement:

```
<<start>>
-- do something here
...
IF (error > 0) THEN
    goto start;
END IF;
```

A null statement has the following syntax:

```
NULL;
```

An example of a null statement:

```
IF (salary > 100000) THEN
    Statement();
ELSE
    NULL;
END IF;
```

### 6.2.3 Cursors

When you execute a SQL statement from PL/SQL, the Oracle RDBMS assigns a private work area for that statement. This work area contains information about the SQL statement and the set of data returned or affected by that statement. The PL/SQL cursor is a mechanism by which you can name that work area and manipulate the information within it. A cursor is basically a “handle” or “pointer” to a dataset represented by a query. With a cursor, rows can be accessed one-by-one in a stored procedure.<sup>1</sup>

A cursor has the following syntax:

```
CURSOR cursorName IS
    SELECT...
```

An example of using a cursor:

```
DECLARE
    CURSOR allPersons IS SELECT * FROM jr_person;
BEGIN
    FOR p1 IN allPersons
    LOOP
        -- process current row (p1)
    END LOOP;
END;
```

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

#### 6.2.4 Stored Function

A stored function is a set of SQL or PL/SQL statements used together to do a particular task (or logic) and stored in the database. PL/SQL is Oracle's procedural extension to SQL.

A stored function has the following syntax:

```
CREATE [OR REPLACE] FUNCTION
  name ([[argMode] [argName] argType [,...]])
  RETURN retType AS
  [declareName dataType; [...]]
  BEGIN
    PL/SQL CODE...
  END
```

An example of a stored function:

```
-- creating stored function
CREATE OR REPLACE FUNCTION isLeapYear(i_year NUMBER) RETURN boolean
AS
BEGIN
  -- A year is a leap year if it is evenly divisible by 4
  -- but not if it's evenly divisible by 100
  -- unless it's also evenly divisible by 400

  IF mod(i_year, 400) = 0 OR
    ( mod(i_year, 4) = 0 AND
      mod(i_year, 100) != 0) THEN
    return TRUE;
  ELSE
    return FALSE;
  END IF;
END
/

-- testing stored function
SET SERVEROUTPUT ON;
BEGIN
  IF isLeapYear(2004) THEN
    dbms_output.put_line('Yes, it is a leap year');
  ELSE
    dbms_output.put_line('No, it is not a leap year');
  END IF;
END;
SET SERVEROUTPUT OFF;
```

(<http://www.orafaq.com/scripts/plsql/leapyear.txt>)

### 6.2.5 Stored Procedure

Functions and procedures are identical except that functions always return a value and procedures do not. By processing the SQL code on the database server, the number of instructions sent across the network and returned from the SQL statements are reduced.

A stored procedure has the following syntax:

```
CREATE [OR REPLACE] PROCEDURE
  name ([[argMode] [argName] argType [,...]]) AS
  [declareName dataType; [...]]
BEGIN
  PL/SQL CODE...
END
```

An example of a stored procedure:

```
-- creating stored procedure
CREATE OR REPLACE PROCEDURE js_remove_person (id NUMBER) AS
total NUMBER;
BEGIN
  DELETE FROM js_person p
  WHERE p.id = js_remove_person.id;
  total := total - 1;
END;
/

-- testing stored procedure
SET SERVEROUTPUT ON;
EXECUTE js_remove_person(123456789);
SET SERVEROUTPUT OFF;
```

### 6.2.6 Package

A package is a collection of procedures, functions, variables, and SQL statements that are grouped together and stored as a single program unit. A package has two parts: a specification and a body. A package's specification declares all public constructs of the package, and the body defines all constructs of the package, and so implements the specification. By doing this, packages provide a form of encapsulation.<sup>1</sup>

A package has the following syntax:

```
-- package prototype
CREATE [OR REPLACE] PACKAGE packageName AS
    [{procedure | function} ...]
END packageName;

...
-- package definition
CREATE [OR REPLACE] PACKAGE BODY packageName AS
    [{procedure | function} ...]
END packageName;
```

An example of a package:

```
-- package prototype
CREATE OR REPLACE PACKAGE js_iags AS
    -- not useful, but an example of a member variable
    total_students NUMBER;

PROCEDURE insert_student(
    id          INTEGER,          -- student id
    fname       VARCHAR2,         -- first name
    mname       VARCHAR2,         -- middle name
    lname       VARCHAR2,         -- last name
    p_email     VARCHAR2,         -- primary email
    p_phone     VARCHAR2         -- primary phone
);

PROCEDURE insert_grade(
    sid INTEGER,                  -- student id
    aid INTEGER,                  -- assignment id
    grade INTEGER,                -- grade points
    details VARCHAR2              -- details
);

END js_iags;
/
```

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

```

-- package definition
CREATE OR REPLACE PACKAGE BODY js_iags AS

PROCEDURE insert_student( id INTEGER,
    fname VARCHAR2, mname VARCHAR2, lname VARCHAR2,
    p_email VARCHAR2, p_phone VARCHAR2) AS
BEGIN
    INSERT INTO js_person VALUES(
        id, fname, mname, lname,
        p_email, p_phone);

    total_students := total_students + 1;
END;

PROCEDURE insert_grade( sid INTEGER,
    aid INTEGER, grade INTEGER, details VARCHAR2) AS
BEGIN
    INSERT INTO js_grade VALUES(
        js_grade_seq.nextval,
        aid,
        sid,
        grade,
        SYSDATE,
        details);
END;

END js_iags;
/

-- using package
EXECUTE js_iags.insert_student(123456789, 'Joe', '', 'Bob', '', '');

```

### 6.2.7 Trigger

Triggers are code stored in the database and invoked (triggered) by events that occur in the database. A trigger may be used to enforce some referential integrity constraints, to enforce complex enterprise constraints, or to audit changes to data. The code within a trigger, called the trigger body, is made up of a PL/SQL block, Java program, or 'C' block. Triggers are based on the Event-Condition-Action (ECA) model.<sup>1</sup>

1. The event(s) that trigger the rule. In Oracle, they are:
  - a) An INSERT, UPDATE, or DELETE statement on a specified table.
  - b) A CREATE, ALTER, or DROP statement on any schema object.
  - c) A database startup or instance shutdown, a user logon, or logoff.
  - d) A specific error message or any error message.
2. The condition that determines whether the actions should be executed. The condition is optional but, if specified, the action will be executed only if the condition is true.
3. The action to be taken. The block contains the SQL statements and code to be executed when a triggering statement is issued and the trigger condition turns out to be true.

A trigger has the following syntax:

```
CREATE [OR REPLACE] TRIGGER triggerName
{BEFORE | AFTER} {INSERT | UPDATE | DELETE }
ON tableName
[REFERENCING [NEW AS newRow]
              [OLD AS oldRow]]
FOR {EACH ROW | STATEMENT}
DECLARE
...
BEGIN
...
END triggerName;
```

An example of a trigger:

```
CREATE OR REPLACE TRIGGER js_trigger_grades_log
  BEFORE UPDATE ON js_grade
  REFERENCING NEW AS newRow
              OLD AS oldRow
  FOR EACH ROW
  BEGIN
    INSERT INTO js_grades_logTable
      VALUES( to_char(:oldRow.points, '99999999'),
              to_char(:newRow.points, '99999999'));
  END js_trigger_grades_log;
```

---

<sup>1</sup> Connolly & Begg, Database Systems 4<sup>th</sup> edition

### 6.3 Oracle PL/SQL Subprogram

In this section I will discussing Oracle PL/SQL subprogram designed for IAGS. This will include the code and comments detail how the subprogram works. Some of the view and stored procedures are made for a particular group of users. To limit there access and make the data more user friendly. The groups and permissions will be explained in more detail in the next section.

The three groups for IAGS are:

1. DB Admin – They have full access to all tables, they can change any table or value.
2. Instructor – They have limited access to section they are/have taught. They can modify some tables and only view others.
3. Student – They have very limited access. They can only view sections they have signed-up for and grades for assignments for those sections.

#### 6.3.1 View(s)

There were several views created for this project. They are used to control the way a user views the data in the database. Some of the views are create to customize the data output to a user group. They are listed below.

##### 1. js\_latest\_grade\_view

Because multiple grades for one student and one assignment is allowed, with different *entered\_date*. A view was needed to list only the latest grades.

PL/SQL Code:

```
CREATE OR REPLACE VIEW js_latest_grade_view
AS
  SELECT g1.*
  FROM js_grade g1
  WHERE
    NOT EXISTS (
      SELECT * from js_grade g2
      WHERE  g1.date_entered < g2.date_entered AND
             g1.assign_id = g2.assign_id AND
             g1.id = g2.id
    );
```

Example:

```
SQL> SELECT * FROM js_latest_grade_view WHERE lower(details) LIKE('%late%');
```

GRADE_ID	ASSIGN_ID	ID	POINTS	DATE_ENTE	DETAILS
3	35	440965598	117	15-OCT-05	turned in late
14	5	369219713	119	19-OCT-05	turned in late
42	37	330200714	25	04-NOV-05	turned in late
48	64	459692930	55	06-OCT-05	turned in late
52	36	272741205	17	10-NOV-05	turned in late



## 2. js\_section\_inst\_view

A custom section view for the instructors group to view section information, course information, and number of students signed-up for that section.

PL/SQL Code:

```
CREATE OR REPLACE VIEW js_section_inst_view AS
  SELECT section_id,
         iid,                                -- instructor id
         year, season_num, worth, crn,
         course_id, short_name,
         COUNT(sid) num_student, -- number of students signed-up
         max_students,
         MIN(grade_points) min_points,
         MAX(grade_points) max_points,
         AVG(grade_points) avg_points
  FROM (
    SELECT
      si.id sid,
      to_number(to_char(se.start_date, 'YYYY')) year,
      se.section_id,
      se.id iid,
      se.season_num,
      se.crn,
      ci.course_id,
      ci.short_name,
      se.max_students,
      si.grade_points grade_points,
      ci.worth
    FROM js_section se, js_course_info ci, js_signup si
    WHERE se.course_id = ci.course_id AND
          se.section_id = si.section_id(+)
  )
  GROUP BY section_id, iid, year,
         season_num, crn, course_id,
         short_name, max_students, worth;
```

Example:

```
SQL> SELECT * FROM js_section_inst_view WHERE iid=123 AND
2  season_num=0 AND year=2005;
```

SECTION_ID	IID	YEAR	SEASON_NUM	WORTH	CRN	COURSE_ID	SHORT_NAME
13	123	2005	0	5	41506	1	cs215
14	123	2005	0	5	41505	2	cs216

NUM_STUDENT	MAX_STUDENTS	MIN_POINTS	MAX_POINTS	AVG_POINTS
0	22			
0	22			

### 3. js\_signup\_inst\_view

A custom sign-up view for the instructor group, to view some section, some course, some person and some sign-up information.

PL/SQL Code:

```
CREATE OR REPLACE VIEW js_signup_inst_view
AS
  SELECT
    to_number(to_char(se.start_date, 'YYYY')) "YEAR",
    se.season_num,
    se.section_id,
    si.id,
    (p.fname || ' ' || p.mname || ' ' || p.lname) name,
    si.grade_letter,
    si.grade_points,
    js_calc_final_points(si.id, se.section_id) calc_points,
    si.details
  FROM js_section se, js_signup si, js_person p
 WHERE si.section_id = se.section_id AND
       si.id = p.id;
```

Example:

```
SQL> SELECT * FROM js_signup_inst_view WHERE year=2005 AND
      2  season_num=0 AND section_id=8;
```

YEAR	SEASON_NUM	SECTION_ID	ID	NAME	GRADE	GRADE_POINTS	CALC_POINTS	DETAILS
2005	0	8	150120557	Sam L. Johnson	D	63.4	32.2727273	
2005	0	8	459692930	Kris P. Warashky	C-	71	59.0909091	
2005	0	8	253221481	Lauren F. Rodregez		0	53.1818182	

#### 4. js\_assign\_inst\_view

A custom assignment view for the instructor group, to view all assignments and grades for all students signed up for a section.

PL/SQL Code:

```
CREATE OR REPLACE VIEW js_assign_inst_view AS
SELECT
    a.assign_id,
    a.section_id,
    a.name,
    a.type_num,
    a.is_extra,
    a.max_points,
    MIN(g.points) min_pts,
    MAX(g.points) max_pts,
    AVG(g.points) avg_pts,
    a.details
FROM js_assign a, js_grade g
WHERE a.assign_id = g.assign_id(+)
GROUP BY a.assign_id;
```

Example:

```
SQL> SELECT * FROM js_assign_inst_view WHERE section_id=8;
```

ASSIGN_ID	SECTION_ID	NAME	TYPE_NUM	IS_EXTRA	MAX_POINTS	MIN_PTS	MAX_PTS	AVG_PTS	DETAILS
2	8	Quiz	1	1	30	5	20	12.5	
3	8	Final	4	0	10				
27	8	Quiz	1	0	130	15	125	71.5	
50	8	Lab	0	0	10	4	4	4	
63	8	Lab	0	0	70				

### 5. **js\_signup\_section\_assign\_view**

A custom sign-up, section and assignment view used by the *js\_grade\_gen\_view* view. It simply makes the other view easier to read. This view should not be used by any group.

#### PL/SQL Code:

```
CREATE OR REPLACE VIEW js_signup_section_assign_view
AS
  SELECT
    si.id,
    se.section_id,
    a.assign_id,
    a.name aname, -- assignment name
    a.max_points,
    a.is_extra,
    (p.fname || ' ' || p.mname
     || ' ' || p.lname) sname -- student name
  FROM   js_signup si, js_section se, js_assign a, js_person p
 WHERE  si.section_id = se.section_id AND
        se.section_id = a.section_id AND
        si.id = p.id;
```

#### Example:

```
SQL> SELECT * FROM js_signup_section_assign_view
      2 WHERE section_id=8 ORDER BY id;
```

ID	SECTION_ID	ASSIGN_ID	ANAME	MAX_POINTS	IS_EXTRA	SNAME
150120557	8	2 Quiz	30	1	Sam L. Johnson	
150120557	8	3 Final	10	0	Sam L. Johnson	
150120557	8	27 Quiz	130	0	Sam L. Johnson	
150120557	8	50 Lab	10	0	Sam L. Johnson	
150120557	8	63 Lab	70	0	Sam L. Johnson	
253221481	8	2 Quiz	30	1	Lauren F. Rodregez	
253221481	8	3 Final	10	0	Lauren F. Rodregez	
253221481	8	50 Lab	10	0	Lauren F. Rodregez	
253221481	8	63 Lab	70	0	Lauren F. Rodregez	
253221481	8	27 Quiz	130	0	Lauren F. Rodregez	
459692930	8	2 Quiz	30	1	Kris P. Warashky	
459692930	8	27 Quiz	130	0	Kris P. Warashky	
459692930	8	63 Lab	70	0	Kris P. Warashky	
459692930	8	50 Lab	10	0	Kris P. Warashky	
459692930	8	3 Final	10	0	Kris P. Warashky	

## 6. js\_assign\_grade\_view

A custom assignment and grade view used by the *js\_grade\_gen\_view* view. It simple makes the other view easier to read. This view should not used by any group.

PL/SQL Code:

```
CREATE OR REPLACE VIEW js_assign_grade_view
AS
    SELECT
        s.section_id,
        s.assign_id,
        s.aname,
        s.id,
        s.sname,
        a.points,
        s.max_points,
        s.is_extra,
        a.details
    FROM   js_assign a, js_latest_grade_view g
    WHERE  a.assign_id = g.assign_id;
```

Example:

```
SQL> SELECT * FROM js_assign_grade_view WHERE section_id = 1 ORDER BY id;
```

SECTION_ID	ASSIGN_ID	ID	POINTS	DETAILS
1	24	185445705	75	
1	34	185445705	30	
1	64	330200714	121	
1	66	459692930	67	
1	64	459692930	55	turned in late

## 7. js\_grade\_gen\_view

A custom grade view for the instructor/student group, to view the grades for all students signed-up in a section. Some columns are only include in order to, later, filter by section and/or person.

### PL/SQL Code:

```
CREATE OR REPLACE VIEW js_grade_gen_view
AS
  SELECT
    s.id,
    s.section_id,
    s.assign_id,
    s.name,
    s.max_points,
    s.is_extra,
    a.points
  FROM   js_signup_section_assign_view s,
         js_assign_grade_view a
 WHERE  s.section_id = a.section_id(+) AND
        s.assign_id = a.assign_id(+) AND
        s.id = a.id(+);
```

### Example:

```
SQL> SELECT * FROM js_grade_gen_view WHERE section_id=1 AND assign_id=8;
```

SECTION_ID	ASSIGN_ID	ANAME	ID	SNAME	POINTS	MAX_POINTS	IS_EXTRA	DETAILS
1	24	Lab	185445705	Brenda E. Johnson	75	80	0	
1	24	Lab	330200714	Jack L. Kerry		80	0	
1	24	Lab	459692930	Kris P. Warashky		80	0	

### 6.3.2 Stored Procedure(s)

There are several stored procedure created for this project. In order to make inserting a grade easier and to calculate the final grade points a student. They are listed below.

#### 1. Insert Grade: **js\_insert\_grade**

This will automatically set the *date\_entered* to the system date, and increment the primary grade id key using the grade sequence.

#### PL/SQL Code:

```
CREATE OR REPLACE PROCEDURE js_insert_grade(  
    sid INTEGER,          -- student id  
    aid INTEGER,          -- assignment id  
    grade INTEGER,        -- grade points  
    details VARCHAR2      -- details  
) AS  
BEGIN  
    INSERT INTO js_grade VALUES(  
        js_grade_seq.nextval,  
        aid,  
        sid,  
        grade,  
        SYSDATE,  
        details);  
COMMIT;  
END;  
/
```

#### Example:

```
SQL> EXECUTE js_insert_grade(253221481, 27, 117, NULL);  
  
PL/SQL procedure successfully completed.
```

## 2. Calculate final grade points for a Student: **js\_calc\_final\_points**

This will return a float number, of the percentage of points a student was given in a section. The percentage will be based on the total points given for all assignments(in one section), plus extra credit. Divided by the total max points from all assignments(in one section), not including extra credit.

### PL/SQL Code:

```
CREATE OR REPLACE FUNCTION js_calc_final_points(  
    stu_id INTEGER,    -- student id  
    sec_id INTEGER     -- section id  
)  
RETURN FLOAT  
AS  
    mtPts js_assign.max_points%TYPE := 0;  
    stPts js_grade.points%TYPE := 0;  
    fnPts FLOAT := 0.00;  
    -- cursor for each grade for a student in a section  
    CURSOR allGrades IS  
        SELECT *  
        FROM js_grade_gen_view  
        WHERE section_id = sec_id AND  
              id = stu_id;  
BEGIN  
    -- add up grade points, and total points  
    FOR g IN allGrades  
    LOOP  
        IF (g.points IS NOT NULL) THEN  
            stPts := stPts + g.points;  
        END IF;  
  
        IF (g.is_extra < 1) THEN  
            mtPts := mtPts + g.max_points;  
        END IF;  
    END LOOP;  
  
    -- error check for no max points, prevent divide by zero  
    IF (mtPts > 0) THEN  
        fnPts := (stPts/mtPts)*100;  
    END IF;  
  
    RETURN fnPts;  
END;  
/
```

### Example:

```
SQL> SELECT js_calc_final_points(185445705, 1) FROM DUAL;  
  
JS_CALC_FINAL_POINTS(185445705,1)  
-----  
                                12.962963
```



### 6.3.3 Trigger(s)

There is only one trigger created for this project. It's listed below.

#### Check/Enforce Student Sign-up Rule: **js\_signup\_insert\_check\_trigger**

This stored procedure checks and/or enforces the rule that students are not allowed to sign-up for a section more than once.

#### PL/SQL Code:

```
CREATE OR REPLACE TRIGGER js_signup_insert_check_trigger
  BEFORE INSERT ON js_signup
  REFERENCING NEW AS nRow
  FOR EACH ROW
  DECLARE
    siCount NUMBER;
  BEGIN
    -- get number of signups
    SELECT COUNT(*) INTO siCount
    FROM js_signup si
    WHERE
      si.id = :nRow.id AND
      si.section_id = :nRow.section_id;

    -- signup count greater than zero, then error
    IF (siCount > 0)
    THEN
      RAISE_APPLICATION_ERROR(-20000,
        'Person Already Signed-up');
    END IF;
  END js_signup_check_trigger;
/
```

#### Example:

```
SQL> INSERT INTO js_signup VALUES(150120557, 1, NULL, NULL, NULL);

1 row created.

SQL> INSERT INTO js_signup VALUES(123456789, 1, NULL, NULL, NULL);
INSERT INTO js_signup VALUES(150120557, 1, NULL, NULL, NULL)
*
ERROR at line 1:
ORA-20000: Person Already Signed-up
ORA-06512: at "CS342.JS_SIGNUP_INSERT_CHECK_TRIGGER", line 12
ORA-04088: error during execution of trigger
'CS342.JS_SIGNUP_INSERT_CHECK_TRIGGER'
```

## Section 7: Users & the GUI Client Application

In this section I will discussing the user groups, development of, and using the graphical user interface (GUI) client application for IAGS.

### 7.1 User Groups

The user groups for the client application are split in to three groups, database administrator, instructor, and student. The client application adds another user type that was not discussed in section 1.5, the database administrator user group. This group was added to allow for easy full table access to the database(for details see below).

*Note: Only one database administrator user and the instructor user group was implemented in IAGS GUI client application, version (0.5 beta).*

#### 7.1.1 Database Administrator(s)

The database administrator user group, is given full access to all of the IAGS tables (see section 7.2.1 for the IAGS table list) in the database. This group usually only consist of one user. The client application was developed with this in mind, only allowing a single user name and a set password stored internally in the client application (see section 7.3.1, on how to login). This user type was added to allow for ease and user friendly interface of main tables management. Most data from the tables is shown in it's raw form, foreign and primary key for example. However there are a few columns that are displayed as window controls for ease of edit. For example the *season\_num* in the *js\_section* table is displayed as a drop list instead of a number. When viewing the table, the client application translate the *season\_num* number to string then selects the item in the drop list, and vise versa when updating or inserting back into the database.

On a daily bases a database administrator user could be adding courses, change an instructor's password, attached an instructor to a section... In the current version of the GUI client application this user is used to maintain the tables and connection between the student, section and instructor. In later versions this connection will try to be automated, thus reducing some of the load from this user. Also with the current version, this user must changing/adding data that other users groups don't have access to. For example the instructor's password. This will also be changed in proceeding versions or the GUI client application. Because, the database administrator user is given full access the IAGS tables, this user is given access to a “Export All data from DB”, “Import into DB” and “Create DB Schema” functions. To do backups of all IAGS data, easily import data into the database, and/or create the DB schema on a specific DBMS. See section 7.3.2, for details on how to use these features.

### 7.1.2 Instructor(s)

The instructor user group uses custom views to display parts of the database that they should have access to, instead of selecting the IAGS tables directly. In the current version of GUI client application, a user's information(name, email, phone...) can only be changed by the database administrator user. In a later version the user will be give a way to change this information.

The instructor user on a daily bases could be adding assignments, entering grades for assignments done by student(s), giving final grades for a section, and printing a grade report for a section. (see section 7.2 for a complete list of viewable/editable columns).

An instructor user will be able to view/edit any of the following:

- A list of sections, this user is teaching, for a particular year and quarter.  
(see section 7.2, “*js\_section\_inst\_view*” view, for details)
- A list of students and final grades, who are signed up for a particular section this user is teaching, year and quarter.  
(see section 7.2, “*js\_signup\_inst\_view*” view, for details)
- A list of assignment(s) for a particular section this user is teaching, year and quarter.  
(see section 7.2, “*js\_assign\_inst\_view*” view, for details)
- A list of grades with student information, for a particular assignment and section this user is teaching. Including all student's signed up for this section, even if they don't have a grade.  
(see section 7.2, “*js\_grade\_gen\_view*” view, for details)
- Generate a report of all students final grade information, for any particular section this user is teaching, year and quarter. Only showing the students last 4 numbers from there ID, final grade points, and final grade letter.  
(view only of the “*js\_signup\_inst\_view*” view, see section 7.2 for details)

### 7.1.2 Students(s)

The student user group uses custom views to display parts of the database that they have access to, instead of selecting the IAGS tables directly. The current specifications define this user as view only, meaning they can only read the data, they can not edit or add new rows in the database. In later versions of the GUI client application, this user group will be added and they may be able to sign-up for a section, drop from a section and/or change there contact information.

The student user on a daily bases could be viewing any of the following:

- A list of course sections and grades for this particular student user.
- A list of assignment grades for this student user, a particular section, year and quarter.

## 7.2 Relations, Views and Subprograms Related to the Group Activities.

This section describes, in details, what information a user group can view, modify and/or add to a table in the IAGS database.

User permissions abbreviation:

RO – User can only read this item.

RW – User can read and edit (UPDATE database data) this table or column.

AD – User can add a new row to this table. (INSERT into database)

NA – User can not view this table or this column directly from this table.

### 7.2.1 IAGS Tables

With the current version of the IAGS GUI client application, these tables are for the database administrator user group only. Instructor and student user group can not view these tables directly.

Name	Database Administrator User	Instructor User	Student User
<b>js_course_info</b>	AD, RW	NA	NA
<b>js_person</b>	AD, RW	NA	NA
<b>js_instructor</b>	AD, RW	NA	NA
<b>js_section</b>	AD, RW	NA	NA
<b>js_signup</b>	AD, RW	NA	NA
<b>js_assign</b>	AD, RW	NA	NA
<b>js_assign</b>	AD, RW	NA	NA

### 7.2.2 Instructor Views

With the current version of the IAGS GUI client application, these views are for the instructor user group only. Database administrator and student user group can not view these views. (see section 6.3.1, to see the views PL/SQL code)

*Note: The view names are in bold and the columns are tabbed in underneath.*

Name	Permissions	Description
<b>js_section_inst_view</b>		
section_id	RO	Section ID
iid	RO	Instructor user ID
year	RO	The year this section is though. Derived from the “start_date” column in the “js_section” table.
season_num	RO	The season number (0 = Fall, 1 = Winter, 2 = Spring, 3 = Summer), this section is though.
worth	RO	Credits or units this course is worth.
crn	RO	CRN
course_id	RO	Course ID
short_name	RO	The course short name.
num_student	RO	An aggregate count of the number of students signed up for this section.
max_students	RO	The maximum number of allowed students to sign-up for this section. This is only a guild line, a rule enforced by the DB Administrator.
min_points	RO	The minimum final grade points from all students final grade points, in this section.
max_points	RO	The maximum final grade points from all students final grade points, in this section.
avg_points	RO	The average final grade points from all students final grade points, in this section.

Name	Permissions	Description
<b>js_signup_inst_view</b>		
year	RO	The year this section is though. Derived from the “ <i>start_date</i> ” column in the “ <i>js_section</i> ” table.
season_num	RO	The season number (0 = Fall, 1 = Winter, 2 = Spring, 3 = Summer), this section is though.
section_id	RO	Section ID
id	RO	Student ID
name	RO	Student Name, composite of first, middle and last name.
grade_letter	RW	The final grade letter for a student for this section. This value comes from the “ <i>js_signup</i> ” table. So using the “ <i>section_id</i> ” and “ <i>id</i> ”, this value can be updated in the “ <i>js_signup</i> ” table.
grade_points	RW	The final grade points for a student for this section. This value comes from the “ <i>js_signup</i> ” table. So using the “ <i>section_id</i> ” and “ <i>id</i> ”, this value can be updated in the “ <i>js_signup</i> ” table.
calc_points	RO	This value comes from the “ <i>js_calc_final_points</i> ” stored procedure. (see Instructor Stored Procedures, for details)
details	RW	The details (or comments) for a student for this section. This value comes from the “ <i>js_signup</i> ” table. So using the “ <i>section_id</i> ” and “ <i>id</i> ”, this value can be updated in the “ <i>js_signup</i> ” table.

Name	Permissions	Description
<b>js_assign_inst_view</b>	AD	When the user adds a row it is inserted into the “ <i>js_assign</i> ” table, with the “ <i>assign_id</i> ” as a auto sequence value, and the “ <i>section_id</i> ” is selected before hand. (see section 7.4 for details)
assign_id	RO	Assignment ID
section_id	RO	Section ID, used to filter assignment, down to a particular section.
name	RW	The assignment name. This value comes from the “ <i>js_assign</i> ” table. So using the “ <i>assign_id</i> ”, this value can be updated in the “ <i>js_assign</i> ” table.
type_num	RW	The assignment type number (0 = Lab, 1 = Quiz, 2 = Test, 3 = Mid Term, 4 = Final). This value comes from the “ <i>js_assign</i> ” table. So using the “ <i>assign_id</i> ”, this value can be updated in the “ <i>js_assign</i> ” table.
is_extra	RW	States whether this assignment is extra credit. This value comes from the “ <i>js_assign</i> ” table. So using the “ <i>assign_id</i> ”, this value can be updated in the “ <i>js_assign</i> ” table.
max_points	RW	The maximum points for an assignment for this section. This value comes from the “ <i>js_assign</i> ” table. So using the “ <i>assign_id</i> ”, this value can be updated in the “ <i>js_assign</i> ” table.
min_pts	RO	The minimum grade points from all students grade points, for this assignment.
max_pts	RO	The maximum grade points from all students grade points, for this assignment.
avg_pts	RO	The average grade points from all students grade points, for this assignment.
details	RW	The details (or comments) for an assignment for this section. This value comes from the “ <i>js_assign</i> ” table. So using the “ <i>assign_id</i> ”, this value can be updated in the “ <i>js_assign</i> ” table.

Name	Permissions	Description
<b>js_grade_gen_view</b>	AD	Grades are never updated but always added. To add a grade, a stored procedure ( <i>“js_insert_grade”</i> ) was create to automate some parts. (see section 7.2.3, for details)
section_id	RO	Section ID
assign_id	RO	Assignment ID
aname	RO	Assignment name
id	RO	Student ID
sname	RO	Student's name, composite of first, middle and last name.
points	RW	The grade points for this assignment and student. This value comes from the <i>“js_grade”</i> table. So using the <i>“assign_id”</i> and <i>“id”</i> , this value can be updated in the that table.
max_points	RO	The maximum points for this assignment.
is_extra	RO	States whether this assignment is extra credit.
details	RW	The details (or comments) for a grade for this assignment and student. This value comes from the <i>“js_grade”</i> table. So using the <i>“assign_id”</i> and <i>“id”</i> , this value can be updated in the that table.



### 7.2.3 Instructor Stored Procedures

With the current version of the IAGS GUI client application, these stored procedures are for the instructor user group only. Database administrator and student user group can not access them. (see section 6.3.2, to see the stored procedures PL/SQL code)

#### 1) **js\_insert\_grade**

Input:           1. Student ID  
                  2. Assignment ID  
                  3. Grade Points  
                  4. Grade Details

Return:          None.

Description: This will insert into the “*js\_grade*” table, automatically incrementing the sequence number, and use the SYSDATE for the “date\_entered” column.

#### 2) **js\_calc\_final\_points**

Input:           1. Student ID  
                  2. Section ID

Return:          A floating point decimal value. Which is the calculated grade percentage for a student in a section.

Description: Using the “*js\_grade\_gen\_view*”, a selection is made filtering to the given section and student. Then in a loop the students points and total points are added up. If the assignment is extra credit then the “*max\_points*” are not added to the total points, but the grade points are still added to the student points. Then the student points are divided by the total points and multiplied by 100.

### 7.2.4 Other Views

There are several views that were created to make the previous views simpler to read. However they are not used directly by any user group. They are “*js\_latest\_grade\_view*”, “*js\_signup\_section\_assign\_view*”, and “*js\_assign\_grade\_view*”. (see section 6.3.1, to see the views PL/SQL code)

## 7.3 Using the GUI Client Application

This section describes all part of the IAGS GUI client application. This includes logging in/out, the database administrator & instructor user functions, help menu, and the end user license agreement (EULA).

### 7.3.1 Logging In/Out

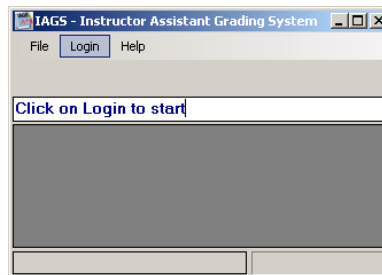
When you first run IAGS client application you are not logged in as a user, therefor you can only Login, Exit, Open the help document, and look at the about dialog.

---

#### Logging In

---

Simple click on the “Login” menu item, highlighted in the image below.

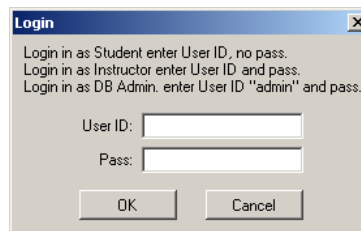


A dialog box will pop up asking for a “User ID” and “Pass”. Like the image below.

**Student:** The current version of application will not allow for a student to login. However in later version they will just enter there user/student ID number in the “User ID” text box.

**Instructor:** To login as an instructor simply enter your user/instructor ID number in the “User ID” text box and password in the “Pass” text box. The test data provided contains a sample instructor. To login as this user, enter “123” as the “User ID” and “test” for the “Pass”.

**DB Admin:** To login as the DB Administrator enter “admin” in the “User ID” text box and “test” in the “Pass” text box.



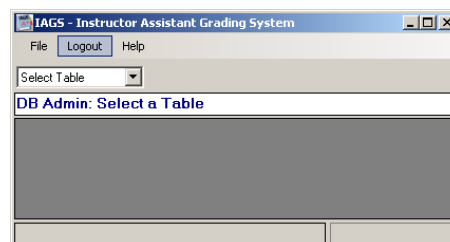
---

#### Logging Out

---

Once logged in the “Login” menu item will change to “Logout”.

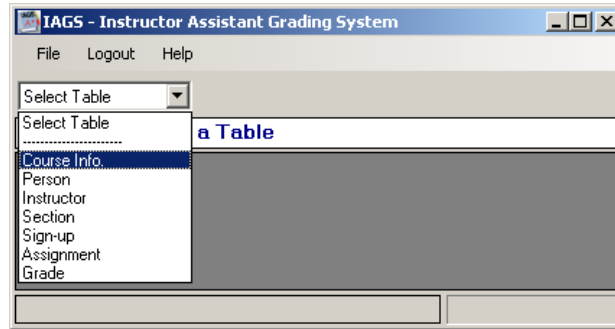
To Logout simple click on the “Logout” menu item, highlighted in the image below.



### 7.3.2 DB Administrator User

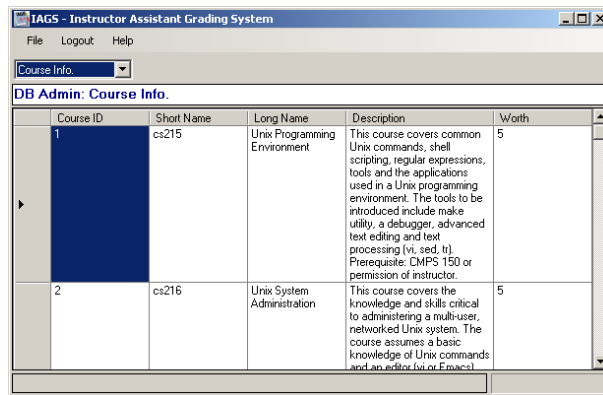
#### Selecting a Tables

Once logged in as the DB Administrator, a drop list of tables will now be visible. As shown in the image below. To manage a table simple click on the drop list then click on the table name.



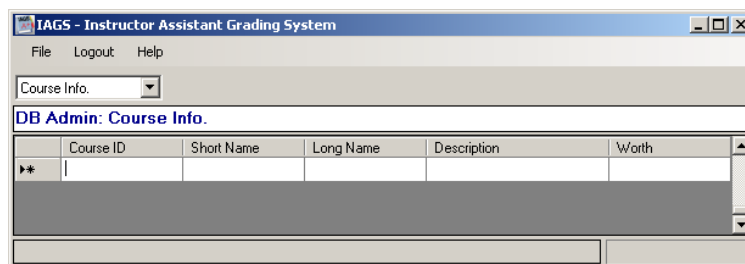
#### Managing the Data (Edit, Add, Delete)

The dark gray area in the middle will be filled with the table data. Looking something like the image below.



**Edit:** a value double click on the cell, and start typing. The row will now be marked as editing and once you click to another row, the change will be sent to the database.

**Add:** a new row, scroll to the bottom of the list you will see an empty row with a star in far left gray box. Click in the empty cell on that row and start editing, it will automatically create a new row. Once you click to another row the new row will be added to the database. The last row should look like the row in the image below.



**Cancel:** a change or new row, simple hit “ESC” on the keyboard.

**Delete:** a row, by selecting a whole row. To do this click on the far left light gray box. It should look something like the image below. Then hit “DEL” on the keyboard.

Course ID	Short Name	Long Name	Description	Worth
1	cs215	Unix Programming Environment	This course covers common Unix commands, shell scripting, regular expressions, tools and the applications used in a Unix programming environment. The tools to be introduced include make utility, a debugger, advanced text editing and text processing (vi, sed, n). Prerequisite: CMPS 150 or permission of instructor.	5
2	cs216	Unix System Administration	This course covers the knowledge and skills critical to administering a multi-user, networked Unix system. The course assumes a basic knowledge of Unix commands and an editor (vi or Emacs)	5

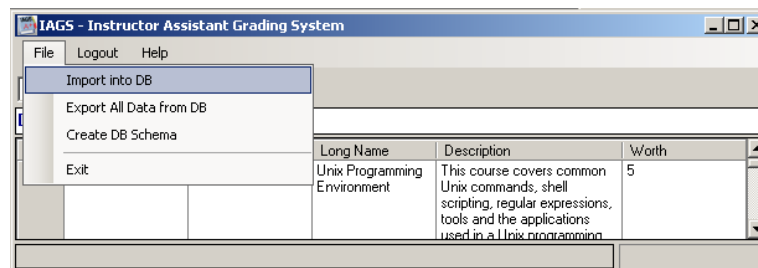
---

### Additional Features (Import, Export, and Create DB Schema)

---

To Import, export and/or create the DB schema, click on the “File” menu then the appropriate item. It should look something like the image below.

Note: All three features will disable all controls when executed, but the process will be display in the status bar at the bottom.



#### Import into DB:

This while first pop up a folder browser dialog box, asking you were to import from. The folder selected must contain a file for each table in the IAGS database. They are: “assign.csv” for the “js\_assign” table. “course\_info.csv” for the “js\_course\_info” table. “grade.csv” for the “js\_grade” table. “instructor.csv” for the “js\_instructor” table. “person.csv” for the “js\_person” table. “section.csv” for the “js\_section” table. “signup.csv” for the “js\_signup” table.

#### Export ALL Data from DB:

This while first pop up a folder browser dialog box, asking you were to export the data to. The application will create the same files for each table as in “Import into DB”, with the data from the database stored as comma separated values within.

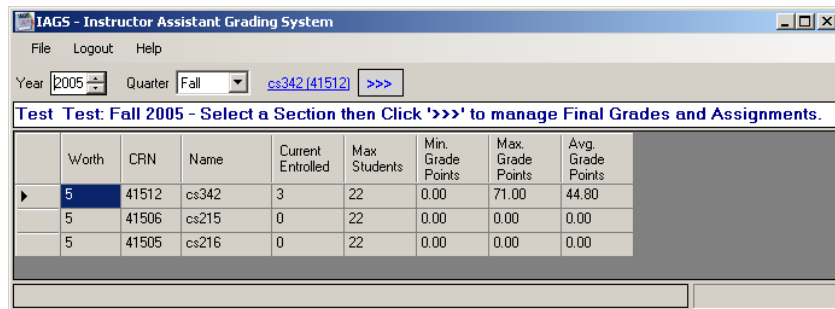
#### Create DB Schema:

This will drop all tables, views and stored procedures if it can. Then create all tables views and stored procedures. **Warning: This function will destroy ALL data in the database.** So use with caution.

### 7.3.3 Instructor User

#### Viewing the Sections and Selecting a Section

Once logged in as an Instructor, a year, quarter, and section link controls will now be visible. As shown in the image below. Base on the year and quarter the main body table will automatically filter to the sections the instructor is teaching for that year and quarter. All the cells in the table are gray, meaning that you can not edit any of the rows. This view also gives the instructor information about how many students are signed up for this section, there min, max, and average final grade points. To view/manage a section's assignments and/or final grades, select a section from the list (the link control just to the left of the “>>>” button will change to that section name) then click on the “>>>” button. To get back to this view at any time simply click on the section link.

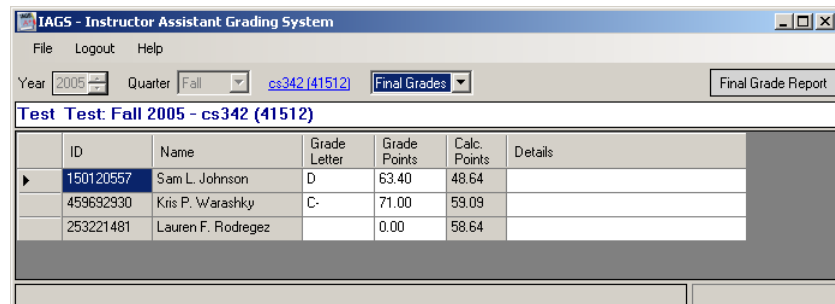


	Worth	CRN	Name	Current Enrolled	Max Students	Min. Grade Points	Max. Grade Points	Avg. Grade Points
▶	5	41512	cs342	3	22	0.00	71.00	44.80
	5	41506	cs215	0	22	0.00	0.00	0.00
	5	41505	cs216	0	22	0.00	0.00	0.00

#### Managing Final Grades, Assignments, and Select an Assignment

Once the instructor has selected a section to manage. The “>>>” will disappears and two more controls will be added. A drop list, allowing the instructor to switch the main table from “Final Grades” to “Assignments”, and a “Final Grade Report” button.

**Final Grade:** When selecting “Final Grade” from the drop list. It will display all the students, there final grade letter, grade point, calculated final points, and details. It would look something like the image below. The cells you can not edit are grayed out. For this view you can edit, the “Grade Letter”, “Grade Points” and “Details. The calculated points column is a percentage of points for each student's accumulated points for assignments over the maximum grade points for the assignments.



	ID	Name	Grade Letter	Grade Points	Calc. Points	Details
▶	150120557	Sam L. Johnson	D	63.40	48.64	
	459692930	Kris P. Warashky	C-	71.00	59.09	
	253221481	Lauren F. Rodregez		0.00	58.64	

**Assignments:** When selecting “Assignments” from the drop list. It will display all assignments for the selected section. Two more controls will now be visible, the assignment link and the “>>>” button. It should look something like the image below. The cells you can not edit are grayed out. They include min, max, and average grade points for each assignment. To edit/add and remove assignments simply follow the same procedure as in the “DB Administrator User, Managing the Data (Edit, Add, Delete)” section. To view/manage an assignment's grades, select an assignment from the list (the link control just to the left of the “>>>” button will change to that assignment name) then click on the “>>>” button. To get back to this view at any time click on the assignment link.

Name	Type No.	Is Extra	Max Points	Student Min. Points	Student Max. Points	Student Avg. Points	Details
Quiz 1	Quiz	<input checked="" type="checkbox"/>	30	5	20	12.50	
Final	Final	<input type="checkbox"/>	10	0	0	0.00	question 20 thr
Quiz 1	Quiz	<input type="checkbox"/>	130	15	125	71.50	
Lab 1	Lab	<input type="checkbox"/>	10	4	4	4.00	
Lab 2	Lab	<input type="checkbox"/>	70	0	0	0.00	

**Final Grade Report:** When you click on the “Final Grade Report” it will pop up a dialog box containing a “Print Ready” version of the final grade list for this section. See “Final Grade Report” section below for details.

## Managing Grades

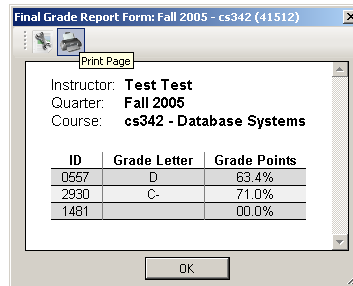
Once the instructor has selected an assignment to manage. The “>>>” will disappear. The main table in the middle will list all students signed up for the section selected and the grade they have for the assignment selected. All cells that can't not be edited will be grayed out. To edit points, and/or details, double click on the cell then, to apply the change click on another row.

Student ID	Student Name	Points	Max Points	Is Extra	Details
150120557	Sam L. Johnson	20	30	<input checked="" type="checkbox"/>	
253221481	Lauren F. Rodrez	0	30	<input checked="" type="checkbox"/>	
459692930	Kris P. Warashky	5	30	<input checked="" type="checkbox"/>	

---

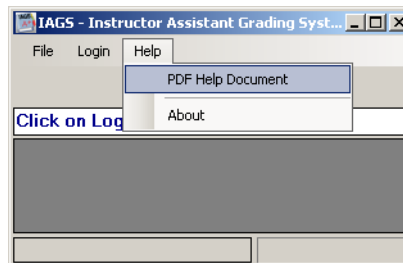
## Final Grade Report

The final grade report dialog box allows you to print the final grade report for a section. It should look something like the image below. At the top there is a tool bar containing two icon buttons, the first one opens a “Page Setup” dialog, and the second one opens the “Print” dialog. The middle is an web browser displaying an HTML generated report, ready for print. To print this page, click on the printer icon in the tool bar and then click “Print” in the print dialog box.



### 7.3.4 Help & About

The PDF help document and About dialog box can be found under the “Help” menu. Similar to the image below.



---

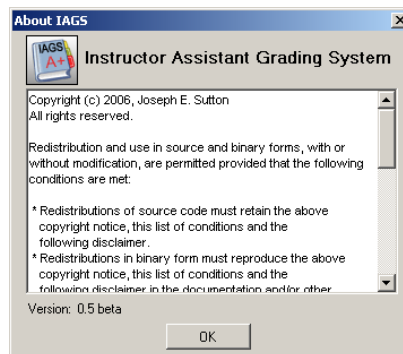
### Help (PDF Document)

If you have a PDF viewer installed, when you click on this menu item it will open the PDF help document using the default viewer.

---

### About Box

Contains a copy of the EULA and the current version number.



### 7.3.5 EULA

IAGS is licensed under the BSD license.

The full license agreement is as follows:

Copyright (c) 2006, Joseph E. Sutton  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- \* Neither the name of the PolyFaust nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



## 7.4 Implementing the GUI Client Application

This section describes how the GUI client application was implemented, designing the user interface, major class/interface description, major features, and the learning process.

### 7.4.1 Designing the User Interface

When designing a user interface, I always start by laying down controls adjusting them till they feel like other well established applications and/or they are easy to use, however providing some powerful features. I also like to customize the controls look and feel just enough to make it my own, and to stand out from the rest.

For this application, instead of old method of disabling controls, I decided to hide all controls that a user don't have access to. This is because C#'s control flow management is very easy to use and provides features that in the old MFC method was very difficult to implement. For example in MFC you hide a control, so it was not visible but the other controls around to didn't shrink or grow to fill in the gap. In C# it's very easy with a table, set to automatically shrink & grow, you add the controls to the table, and now the controls will slide, shrink or grow to fill in the gap.

Visual Studio 2003/2005 provide a very user friendly method of designing an application interface. Using a GUI designer, you simply drag the control from the toolbar, set the properties, and add events. Properties are things such as displayed text, color, font, width, height, alignment.... Events are functions that are triggered when that control changes in some fashion. For example, for a button you can set a “click” event, so when a user clicks on the button a function is called. Each event, sends a sender object, and event arguments to the function assignment to it. The sender object is the control that has the attached function. The arguments contain information about the event to occurred. Multiple events from different controls can be attached to the same function, but the function must have appropriate event signature. The even signature is like a function prototype for an event.

Beside simply dragging controls, setting properties, and events. One of the most important part of the designing the application interface is simple using it. Going thought the motions that a “normal” user would go though. This is also part of the testing process but influences the design. After years of experience of GUI software development, I learned one important thing. You can't please all the users all the time, so just do the best you can do and don't sweat the small stuff.

#### 7.4.2 Major Class/Interface Description

The GUI client application code structure is broken up in a large number of classes. This was done because, one of the main goals was to make it easily portable to other DBMS's and other user interfaces, such as a web interface. So the major classes are the generic bases classes that implement the features of the application, they are listed below. The major class sections will contain a description of the class and a list of import functions and variables. In the “Classes for User Groups” section, you will find a list of classes and descriptions for the particular user groups.

##### 1. cIAGS class

This is the main class that manages the entire application. There is only one instance of the class in the Program class, as a static variable. It provides methods and functions so that classes can talk to to each other. It manages what user group the current user is, what view state they are in and objects for all the lists, views and reports. It also updates some GUI controls such as the info box.

Major variables and functions:

Variable or Function prototype	Description
<code>int security_level</code>	Each user group has a defined security level number (Student = 0, Instructor = 1 and DB administrator = 2). If this value is “-1” then the user is not logged in. Once a user is authenticated for a group this variable is set. This variable is very important to security of the IASG data. It is also important for the fact that it defines that way the application look to the user. If you are an instructor then you are displayed a different set of controls from the DB admin. for instance.
<code>int sub_state</code>	In the current version only the Instructor group, uses this variable. Bases on it's value, it defines what controls and view the instructor can see. If it's “0”, then the user is in the Section View (“section_view”). If it's “1”, then the user is in the Sign-up View (“inst_signup_view”). If it's “2”, then the user is in Assignment View (“inst_assign_view”). If it's “3”, then then user is in Grade View (“inst_grade_view”).

<code>cDBItemList</code> <code>current_view</code>	This is a reference to one of the list or view objects below. See <code>cDBItemList</code> for details.
<code>cPersonList</code> <code>person_list</code> <code>cCourseInfoList</code> <code>course_info_list</code> <code>cInstructorList</code> <code>inst_list</code> <code>cSectionList</code> <code>section_list</code> <code>cAssignmentList</code> <code>assign_list</code> <code>cSignupList</code> <code>signup_list</code> <code>cGradeList</code> <code>grade_list</code>	These are the list objects for the DB Administrator user group. Each list object is for each table in the database. For example “ <code>cPersonList</code> ” class is for the “ <code>js_person</code> ” table. All of these classes derive from the <code>cDBItemList</code> class. See <code>cDBItemList</code> for details.
<code>cSectionView</code> <code>section_view</code> <code>cInstSignupView</code> <code>inst_signup_view</code> <code>cInstAssignView</code> <code>inst_assign_view</code> <code>cInstGradeView</code> <code>inst_grade_view</code>	These are the view objects for the Instructor user group. Each view object is for one or more instructor view(s) in the database. All of these classes derive from the <code>cDBItemList</code> class. See <code>cDBItemList</code> for details.
<code>cPerson</code> <code>current_login</code>	This is the current logged in user information. There name, ID...
<code>int</code> <code>current_sem</code> <code>int</code> <code>current_year</code> <code>cSectionViewRow</code> <code>current_section</code> <code>cInstAssignViewRow</code> <code>current_assign</code>	These variables are used for the Instructor user group in order to keep track of what season, year, section and assignment they selected.
<code>bool</code> <code>LoggedIn</code> <code>bool</code> <code>Login(string u, string p)</code> <code>void</code> <code>Logout()</code>	Log in and out functions.
<code>void</code> <code>SwitchView(string name)</code>	Based on the input string this function changes the “ <code>current_view</code> ” object reference to one of the list or view objects.
<code>void</code> <code>PopupReport()</code>	This is Final Grade Report function. It creates a new Report Form object then displays a generated html document from the “ <code>cFinalGradeReport</code> ” class.

<pre>int CreateAll() int LoadAll() int SaveAll()</pre>	<p>All these function create a thread to run a function with the same name but has “Proc” at the end. The purpose in using the thread was because these functions can take a long time and could appear to the user as the application has locked up. Create all, will drop all tables, views and stored procedures, if it can, then create them all.</p> <p>Load all, will import the comma separated files into the database. Similar to the Java data loader.</p> <p>Save all, will export all the tables to comma separated files.</p> <p>(See section 7.5, for details on problems that arouse from using a thread.)</p>
<pre>void dataGrid_CellValueNeeded(...) void dataGrid_CellValuePushed(...) void dataGrid_NewRowNeeded(...) void dataGrid_RowValidating(...) void dataGrid_CancelRowEdit(...) void dataGrid_UserDeletingRow(...)</pre>	<p>These functions pass there input to the “current_view”’s functions with the same name. These functions provide a generic interface for the DataGridView to pass event messages to, but be handled by one the specific classes.</p>

## 2. **cDBItemList** class

This class provides an abstract class like feature for the list, view, and report classes. Almost all the functions are are virtual, some do nothing and need to be overridden in the child class. This class derives from another class called “cDB” which add the DB functions. See “cDB” below for details.

Major variables and functions:

Variable or Function prototype	Description
ArrayList dlist	This is an array of “cBaseItem” objects. See “cBaseItem” below for details. Basically the rows from the selection.
<pre>int LoadFromFile() int SaveToFile()</pre>	Load takes a comma separated file and translate that to the “dlist”. Save takes the “dlist” and translate it to a comma separated file.

<code>int Create()</code>	Creates the table(s) and/or view(s) in the database.
<code>int DropTable()</code> <code>int DropView()</code>	Drops the table(s) and/or view(s) from the database.
<code>int SelectAll()</code>	Select all from the table or view.
<code>int InsertDB(...)</code> <code>int UpdateDB(...)</code> <code>int DeleteDB(...)</code>	Insert, update and/or delete from table. For a view this would obviously need to be overridden to insert, update and/or delete from a table.
<code>int dataGrid_Define(...)</code>	This defines the columns of the DataGridView.
<code>cBaseItem getNewRow()</code>	This function is intended to be overridden by a child class. Returning the child row class object which has “cBaseItem” as it's base class.
<code>void dataGrid_CellValueNeeded(...)</code> <code>void dataGrid_CellValuePushed(...)</code> <code>void dataGrid_NewRowNeeded(...)</code> <code>void dataGrid_RowValidating(...)</code> <code>void dataGrid_CancelRowEdit(...)</code> <code>void dataGrid_UserDeletingRow(...)</code>	DataGridView event functions. They control what data is placed in a cell. Adding new row, which adds a new row to the DB. Deleting a row, which deletes the row from the DB. Updating a row, which updates a row in the DB.
<code>object ReadData(object[] row)</code>	“DB_base” object calls this function to custom fill the data array list in the “cBaseItem”. See below for details on “cBaseItem” and “DB_Base”.

### 3. **cDB** class

This class provides functions to do DB functions. This is done though the “db” object which is a reference to a “cDB\_MySQL” or “cDB\_Oracle” object. See “cDB\_Base” for details.

Major variables and functions:

Variable or Function prototype	Description
<code>cDB_Base db</code>	In the current version this references a “cDB_MySQL” or “cDB_Oracle” object. Based on the static “type” value. See “cDB_Base” for details.
<code>void Connect()</code> <code>void Disconnect()</code>	These connect and disconnect from the DBMS using the “db” object.
<code>int ExecSQL(...)</code>	Connects, sends a statement to the DBMS, through the “db” object, then disconnects.
<code>int ExecProc(...)</code>	Connects, sends stored procedure statement to the DBMS, through the “db” object, then disconnects.
<code>int Select(...)</code> <code>int SelectAll(...)</code>	Adds the “SELECT..” command to the passed in string and then passes it to the “ExecSQL” function.
<code>int ExecProcedure(...)</code>	Calls the “ExecProc” function
<code>bool CheckTable(...)</code>	If the table exist then returns true, it not return false
<code>int Insert(...)</code> <code>int Update(...)</code> <code>int Delete(...)</code>	Adds the “INSERT...”, “UPDATE...”, or “DELETE...” command to the passed in string and then passes it to the “ExecSQL” function.
<code>int CreateSeq(...)</code> <code>int CreateTable(...)</code> <code>int CreateView(...)</code> <code>int CreateProc(...)</code>	Adds the “CREATE...” command to the passed in string and then passes it to the “ExecSQL” function.
<code>int DropSeq(...)</code> <code>int DropTable(...)</code> <code>int DropView(...)</code> <code>int DropProc(...)</code>	Adds the “DROP...” command to the passed in string and then passes it to the “ExecSQL” function.

#### 4. **cDB\_Base** interface

This is an interface for the real DB implementation functions. In the current version only the “cDB\_MySQL” (MySQL DBMS) and “cDB\_Oracle” (Oracle DBMS) classes implement this interface. There connection details are below.

Major variables and functions:

Variable or Function prototype	Description
<code>void Connect()</code> <code>void Disconnect()</code>	These function connect and disconnect from the DB server.
<code>int ExecProc(...)</code> <code>int ExecSQL(...)</code> <code>int ExecSQL_RowCount(...)</code>	These functions send a SQL statement to the DB server. ExecSQL_RowCount get the first row, first column data, converts it to a number then returns that value. This function is interned to be used with the “DB” class function “ExecSQL_RowCount” which is called by the “RowCount” function. “RowCount” was created to do a count on all rows returned by a selection (“count(*)”).

##### a) **cDB\_MySQL** class

This class implants the “cDB\_Base” interface for the MySQL DBMS. It uses the a “OdbcConnection” object to connect/disconnect and a “OdbcCommand” object to send a command to the DBMS. The “OdbcCommand” provides functions to read data returned from the DBMS. I used two different ways of executing SQL statements on the DBMS. One was to call “ExecuteReader” in the “OdbcCommand” object and the other was “ExecuteNonQuery” from the “OdbcCommand” object. The first method is used with “select” statements were the DBMS returns rows of data. The second method just sends the statement and only cares if it failed or not. The second function was used for all statements that were not “select” statements.

Here are the step IAGS goes through in order to execute a SQL statement

1. Build SQL statement
2. Create OdbcConnection object
3. Create OdbcCommand object using connection object
4. Set command text to SQL statements
5. Open connection  
Selection:
  1. ExecuteReader
  2. Read each row
  3. Add row data to dlist in cDBItemList objectInsert, Update, Delete...:
  1. ExecuteNonQuery
6. Close connection
7. Deallocate all

b) cDB\_Oracle class

This class implants the “cDB\_Base” interface for the Oracle DBMS. It uses the “OleDbConnection” object to connect/disconnect, a “OleDbCommand” object to send a command to the DBMS, “DbDataAdapter” and “DataTable”. I used two different ways of executing SQL statements on the DBMS. One was to call “ExecuteNonQuery” from the “OleDbCommand” object and the other method uses a “DbDataAdapter” object. The first method just sends the statement and only cares if it failed or not. The first function was used for all statements that were not “select” statements. The second method is used with “select” statements where the DBMS returns rows of data

Here are the steps IAGS goes through in order to execute a SQL statement

1. Build SQL statement
  2. Create OleDbConnection object
- Selection:
1. Create DbDataAdapter from connection object
  2. Set command text to SQL statements
  3. Pass data adapter to DB object's fill function
  4. In DB fill function
    1. Create DataTable
    2. Fill data table using data adapter
    3. Add row data to dlist in cDBItemList object

Insert, Update, Delete...:

1. Open connection
  2. Create OleDbCommand object from connection object
  3. Set command text to SQL statements
  4. ExecuteNonQuery
3. Close connection
  4. Deallocate all

The Ole method of reading data from the server is slightly different than the Odbc method, but they both result in filling the “dlist” with the table/view data.

Note: With both DBMS's the server connection information is stored within the class as a string. Include the database name, DBMS login user name and password. In later versions this should be moved to a configuration file.



##### 5. **cBaseItem** class

This class provides an abstract class, like features for the list, and view row classes. This class is intended to store a single row's column data and provide default cell update and display functions.

See “Classes for User Groups” for a list of classes that use this class as a base.

Major variables and functions:

Variable or Function prototype	Description
<code>ArrayList data</code>	A list of column cell values from a selection.
<code>int offset</code>	This is used when getting the values for a row. To offset which column to display. For example, with some of the instructor views the primary and foreign keys should to be hidden. So this variable is used to specify what column the “render” function should start at.
<code>object RenderCell(int index)</code>	Returns the column data at index plus offset.
<code>bool UpdateCell(...)</code>	This updates the column data at index plus offset.

## 6. **cSqlValueList** class

This allows for an easy and safe way to build a list of values to add to an SQL statement. For use with the “UPDATE”, and “INSERT” SQL statements. In SQL numbers are sent with no special precautions, but text and date & time values on the other hand needs special treatment. Text needs to be enclosed in single quotes and for security reasons all the special characters should be escaped. Date & Time values need to be formatted in a specific way, and with Oracle need to be in a function. This class solves all of these issues.

Major variables and functions:

Variable or Function prototype	Description
<code>string SQL</code>	This is properties that returns the internal SQL formatted string
<code>static string Implode(...)</code>	This is a static function that simple returns a SQL formatted string of the input ArrayList. Each item in the array list it is converted to a string then and separated by commas.
<code>void Add(string n, object d)</code>	This function is intended to be used with the “UPDATE” SQL command. The “name” string is the table column name, and the object is it's value. For example, Add(“details”, “test”), would add “details=’test’” to the SQL statement string. Note: a comma would be add to the previous SQL statement string is another value was added before it.
<code>void Add(object o)</code>	This is a generic Add object function. It uses the “is a” check to determine the real object type. This function was created because the ArrayList is a list of objects, not that type you add. So when reading the ArrayList you have to check the type before adding it to the list.

<code>void Add(string s)</code>	This function encloses the string in quotes and escapes all special characters in the string. The escaping of special characters was added to prevent a user from injecting there own SQL statement.
<code>void Add(DateTime d)</code>	This formats the date & time for a particular DBMS.
<code>void Add(int d)</code> <code>void Add(uint d)</code> <code>void Add(byte d)</code> <code>void Add(bool d)</code> <code>void Add(short d)</code> <code>void Add(ushort d)</code> <code>void Add(float d)</code> <code>void Add(double d)</code>	These functions simple add the number to the list. This is because numbers don't need any special formating.

## 7. Classes for User Groups

Below is a list of class used by particular user groups, the base classes they derive from and a description of purpose. All the user group classes derive from a base class and overriding functions for it's purpose.

User Group	Class Name	Base Class	Description
DB Admin.	cAssignmentList	cDBItemList	This class represent the “js_assign” table. Allowing the user to insert, update and delete from the table.
DB Admin.	cCourseInfoList	cDBItemList	This class represent the “js_course_info” table. Allowing the user to insert, update and delete from the table.
DB Admin.	cGradeList	cDBItemList	This class represent the “js_grade” table. Allowing the user to insert, update and delete from the table.
DB Admin.	cInstructorList	cDBItemList	This class represent the “js_instructor” table. Allowing the user to insert, update and delete from the table.
DB Admin.	cPersonList	cDBItemList	This class represent the “js_person” table. Allowing the user to insert, update and delete from the table.
DB Admin.	cSectionList	cDBItemList	This class represent the “js_section” table. Allowing the user to insert, update and delete from the table.
DB Admin.	cSignupList	cDBItemList	This class represent the “js_signup” table. Allowing the user to insert, update and delete from the table.
DB Admin.	cAssignment	cBaseItem	This class contains all the column data from a single row, from the “js_assign” table in the DB.

DB Admin.	cCourseInfo	cBaseItem	This class contains all the column data from a single row, from the “js_course_info” table in the DB.
DB Admin.	cGrade	cBaseItem	This class contains all the column data from a single row, from the “js_grade” table in the DB.
DB Admin.	cInstructor	cBaseItem	This class contains all the column data from a single row, from the “js_instructor” table in the DB.
DB Admin.	cPerson	cBaseItem	This class contains all the column data from a single row, from the “js_person” table in the DB.
DB Admin.	cSection	cBaseItem	This class contains all the column data from a single row, from the “js_section” table in the DB.
DB Admin.	cSignup	cBaseItem	This class contains all the column data from a single row, from the “js_signup” table in the DB.
Instructor	cInstSectionView	cDBItemList	This class represent the “js_section_inst_view” view. Allowing the user to only view the table data but update it, add rows or delete rows.
Instructor	cInstSignupView	cDBItemList	This class represent the “js_signup_inst_view” view. Allowing the user to view and update some values. But they can't add or delete rows.
Instructor	cInstAssignView	cDBItemList	This class represent the “js_assign_inst_view” view. Allowing the user to add, update and delete rows.

Instructor	cInstGradeView	cDBItemList	This class represent the “js_grade_gen_view” view. Allowing the user to only add rows, this is because of the “can't update grades policy”. So they can't delete or update rows. For this view rows are displayed even if entries don't exist and if they exist only the latest ones are displayed. This class uses the “js_insert_grade” stored procedure (see section 7.2.3 for details) to insert a grade.
Instructor	cInstSectionViewRow	cBaseItem	This class contains all the column data from a single row, from the “js_section_inst_view” view, when selected.
Instructor	cInstSignupViewRow	cBaseItem	This class contains all the column data from a single row, from the “js_signup_inst_view” view, when selected.
Instructor	cInstAssignViewRow	cBaseItem	This class contains all the column data from a single row, from the “js_assign_inst_view” view, when selected.
Instructor	cGradeViewRow	cBaseItem	This class contains all the column data from a single row, from the “js_grade_gen_view” view, when selected.
Instructor	cFinalGradeReport	cDBItemList	This class selects all rows from the “js_signup_inst_view” view. Then generates a HTML document for display and/or print. In later versions of the IAGS GUI client application, a report class should be created.

## Instructor Assistant Grading System



#### 7.4.3 Major Features

The major features of the IAGS GUI client application are as follows:

- a) MySQL 5.0+ and Oracle 8i DBMS's support. In the current version this feature is only supported by changing a single line of code then rebuilding the application. But in later versions a configuration file will contain the DBMS information and which one it should use.
- b) Custom user group experience
  - 1. DB Administrator
    - i. Import into the DB from comma separated value (CSV) files.
    - ii. Export all data from the DB to CSV files.
    - iii. Create IAGS database schema on a specific DBMS.
    - iv. Manage all tables in the IAGS database.
  - 2. Instructor
    - i. Custom/filtered views, with useful student grade/assignment statistics.
    - ii. Generate an HTML document report (for display or print) of the final grade for a section.
- c) Clean design, free of extraneous/confusing controls.
- d) BSD user license. Meaning that it's free to share and modify without the developers consent.

#### 7.4.4 The Learning Process

For me the learning process include lots of action, doing. I usually don't just sit down and read a book. I jump in, full steam, trying to do what I have done before in other languages or applications. If I can't discover the solution on my own, I do a Google search. Finally if all else fails I ask a fellow classmate or co-worker. I like to discover the solution myself or figure out the solution from an on line example, over ask for help. Mainly for the reason that it helps me remember how to solve the problem again, when it arises. For a new languages, of course, I go over the basic how to tutorials, but after that I just start trying things.

This process only works when I'm trying to learn a new application or language. If I'm developing a new application that's a totally different game, all together. I plan (what the app. will do), do small tests, design class relationship (on paper or diagram), test again, revise plan, sketch out basic user interface and then code. With a well developed plan, coding should be quicker and easier.



## 7.5 The Path to Success

In order to successfully develop a working useful database application. Design and implementing a database application is very similar to other application development. However there is one major difference, database applications usually manage large amounts of data. So making the interface simpler is better.

For this application I did the following:

1. **Designing.**

I design the class relationship structure and layout user interface. I did this on paper, and using the MS VS 2005 form designer. I learned that MS VS 2005 form designer makes GUI application development very easy and user friendly.

2. **Testing.**

I create a small test application to see how to make a DB connection to both Oracle and MySQL. I discovered that Oracle did not work well with the Odbc class. There was a problem when selecting, it would not return the list of row. So I used the Ole class instead. Then I create the base interface for both class to make the connection, disconnection and queries. The base class hides the underlying working, allowing for other DBMS's to be supported in the future. I learned that Odbc does not work well with all DBMS's

3. **Implementation.**

I proceeded to implement the classes, functions, properties that I outlined in the design phase. With little to no problems. It was quite strait forward, like dozens of other application I developed in the past. The DB connection was very easy, and strait forward. So, I'm not going to outline all the steps in the part, because most were outlined in the 7.4 section.

However, there is one major thing that was an eye opener, something that I was not aware of before writing this application. It was GUI control handling between threads. Thread handling in C# is very easy, create a thread, say what function is going to be in the thread, and finally start the thread. However, if you try to change a control from that thread, VS 2005 will throw an exception. This is because the controls can only be updated on the main thread. In the past this was not that case, it would let you change a control on another thread, but could result in controls flashing or being corrupted. My goal was to spawn a thread, do DB connection/queries, update the status and progress bar, all in that thread, then close the thread. A typically refereed to as a “ worker” thread.

I have to give my thanks to Casey Langen, for the solution to this problem. He told me that I have to use an event defined my a delegate prototype. In the event handler function I check to see if it's on the main thread if it's not, then invoke a new event to the main thread and return. Otherwise if the even is called from the main thread then it updates the control based on the arguments passed in. So I call the event from the worker or main thread and the event will make sure the message is called from the correct thread before doing any GUI control updates.

I learned that some languages are better suited to some tasks better then other languages, simply by design. C# is defiantly designed for GUI application development